Lightspeed Black Box SDK

API Specification

Version 1.12
September 7, 2010

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

To obtain additional copies of this document, contact:

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

Copyright © 2010 Lightspeed Financial, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form by any means without the written permission of Lightspeed Financial Inc.

Other third party product names used herein are used to identify such products and for descriptive purposes only. Such names may be marks and/or registered marks of their respective owners.

About this Document

This document describes the API for Lightspeed's Black Box Software Development Kit (SDK) Library, referred to as the SDK Library. It is intended to give the reader all the information required to use the SDK Library.

Who Should Read this Document

This document is intended for software developers that will be using Lightspeed's SDK Library to develop a Black Box Trading System.

Terminology

Throughout this document the following terms are used:

Term	Definition
SDK	Software Developers Kit. A set of tool that allow Lightspeed customers to quickly develop Black Box Trading Systems.
SDK Library	A compiled form of the SDK software components common to all Black Box 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 Black Box Trading System.
Remote Console	A console that is running on a computer that is physically located at a different location than the computer the Black Box Trading System application is running on.
Lightspeed's Black Box Trading Environment	A set of services provided by Lightspeed that allow customers to develop Black Box Trading Systems. These services include: market data service, order entry service, risk management, short availability service, etc..
Black Box Trading System	A software application developed for the purpose of trading for a profit with little human intervention. Throughout this document the term "Black Box Trading System" is used to refer to an application that is created by compiling the customer's software and liking it with the SDK Library.
Trading Strategy	A component of a Black Box Trading System that will analyze market

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved L

Logic	conditions, and when conditions are met, will send buy and sell orders to the market.
Position Management Logic	A component of a Black Box 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 routine, protocols and tools for building software applications.
Lightspeed Gateway	Lightspeed's order entry server. Black Box Trading Systems connect to the Lightspeed Gateway to send orders to the market and receive status about orders form 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 SDK Library.
Callback Function	Refers to the SDK 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 SDK Library to create a Black Box Trading System.
Console operator or Operator	The person responsible for starting, stopping, configuring and monitoring the Black Box Trading System.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved L

Revision History

The table below records the revision history of this document:

Revision	Date	Changes Made
1.00	January 25, 2010	Initial Document
1.01	May 4, 2010	Added an account parameter to the following callback functions: - update_global_trading_state - query_global_trading_state - update_single_stock_trading_state - query_single_stock_trading_state - update_allowed_trading_side - query_allowed_trading_side - enable_trading_letter_range - query_trading_letter_state - disable_trading_letter_range - trailing_stop_loss_exceeded - trailing_stop_loss_amount_modified - start_time_event - stop_time_event - add_one_stock - update_console_down_mode - query_console_down_mode - update_trading_mode - query_trading_mode Change parameter type from long to double in the following callback functions: - trailing_stop_loss_exceeded - trailing_stop_loss_amount_modified Change the description of the buying_power parameter in the callback funciton sod_buying_power to be more clearn
1.02	May 6, 2010	No longer using long long. All long long changed to double.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved L

		Removed the function call asctoll
1.03	May 11, 2010	Added query_account_numbers() and query_account_mode()
1.04	May 13, 2010	Changed query_account_numbers() to get_account_numbers(). Changed query_account_mode() to get_account_mode().
1.05	June 4, 2010	Changed order_type_status() codes to 'U' and 'D'.
1.06	June 17, 2010	Added get_fee_info() function call
1.07	July 14, 2010	Added Network Interface section
1.08	August 6, 2010	Added Futures and Indices secion
1.09	August 9, 2010	Added event_time parameter to quote_event_received() Added market_center_code and trade_time parameters to trade_received() Add execution_time parameter to execution_report_received() Change parameter type form long to char for mode parameter in updata_trading_mode() callback function
1.10	August 17, 2010	Change the parameters value and net_change to double in the callback function index_data_update()
1.11	August 26, 2010	EDGA data was added to the SDK platform. This document was updated to reflect that EDGA is now a supported data source.
1.12	September 7, 2010	Add reference to Imbalance feed (IMBL). Added the callback function lrp_received(). Added the callback functino imbalance_received()

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved L

Contents

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved L

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved L

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved L

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved L

1. Introduction

Lightspeed's Black Box Trading System SDK Library consists of a number of C function calls and C callback functions. Function calls are C functions that can be called by the customer's software. Callback functions are C functions provided by the customer's software and will be called by the SD Library. Stubs for all the callback functions are provided in the customer source code template.

This document defines the function calls and callback functions. The function calls and callback functions organized by category to assist the reader in locate information quickly. The following categories are used and each category is discussed in a separate section of this document.

- System Functions
- Start/Stop Trading Functions
- Market Data Functions
- Order Processing Functions
- Timer Functions
- Alarm Clock Timer Functions
- Short Availability Functions
- Console Interface Functions
- Utility Functions
- Reading Configuration Parameters From a File

1.1. Price Representation

Unless otherwise specified, prices in the SDK Library are implemented as integers with four digits of precision to the right of the decimal point. For example, the price of 25.98 is represented as the integer value 259800.

1.2. Data Sources

Lightspeed received market data from many sources and makes the data available to Black Box Trading Systems. Lightspeed provides the following methods for redistributing the data.

Prints and Exchange Quotes servers receive data from the NYSE CTS, CQS and Alerts feeds, and the NASDAQ UTDF and UQDF feeds. The data is then reformatted and sent to Black Box Trading System using a protocol define by Lightspeed.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

Multicast data is available from many sources and is made available to Black Box Trading Systems. The multicast data is in the format specified by the data source.

The SDK Library has access to the following data sources:

- Prints and Exchange Quotes servers (PEQS)
- INET ECN-MD servers (INET)
- ARCA OTC ECN-MD servers (ARCA)
- ARCA Listed ECN-MD servers (ALST)
- ARCA ETF ECN-MD servers (AETF)
- BATS ECN-MD servers (BATS)
- EDGX ECN-MD servers (EDGX)
- EDGA ECN-MD servers (EDGA)
- Futures Market Data Server (INDX)
- Imbalance Data (IMBL). Note, Imbalance Data is obtained from the Prints and Exchange Quotes server. Imbalance data is treated as a separate data stream by the SDK Library. This allows the customer to register to receive only quotes data, only Imbalance data, or both from the Prints and Exchange Quotes server.

1.3. Order Types

Lightspeed can route orders to several Exchange and ECN. Lightspeed can also process several different types of orders. Version 1.00.00 of the SDK Library allows the customer's software to specify the following order types when entering an order.

Value	Description
'I'	INET order
'A'	ARCA order
'R'	Rash order
'E'	AMEX order
'T'	BATS order
'H'	EDGA order
'G'	EDGX order
'O'	Boston Ouch order
'Y'	NYSE Hidden (SuperDot, DirectPlus)

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

1.4. Console Display Information

The console is use to do the following:

- Display information about the Black Box Trading System
- Change configuration parameters
- Manual enter orders and cancel orders

Some of the information displayed on the console is provided by the SDK Library and some is provided by the customer's software. The SDK Library maintains state about orders and positions and will provide this information to the console. The SDK Library will provide the console with the following information:

- Total P&L (net and gross)
- Total Exposure
- Commission and fees
- Shares traded
- Number of orders entered
- Number of live orders
- Top 10 winners and losers (P&L and shares traded)
- Position summary (number of open positions, long shares, short shares)
- Open positions (symbol, price, size, exposure, and realized Net P&L)
- Market data status
- Order type status
- Trailing stop loss amount
- Market data for the security in the Trade Window
- Short Availability flag for the security in the Trade Window
- Total daily volume for the security in the Trade Window
- Position price and size for the security in the Trade Window

The customer's software is responsible for providing some of the information displayed on the console. The customer's software will be queried periodically for state information. The information returned from the query will be displayed on the console. The customer's software will be queried for the following information:

- Global trading state (Trade, Don't Trade, Liquidate Only, Bail Out, Not Available)
- Trading state for a single security (Trade, Don't Trade, Liquidate Only, Not Available)
- Trading side (Either, Long Only, Short Only, Not Available)
- Auto or Manual start/stop trading mode (Auto, Manual, Not Available)
- Run with all consoles down mode (Run, Stop, Liquidate Only, Not Available)
- Trading Letter state (Enabled, Disabled, Not Available)
- Version of the customer's software

Customers are responsible for writing their trading strategy code and position management code. This code is referred to as the customer's software. The customer may need to display information on the console specific to their trading strategy, and change trading strategy configuration parameters via the console. The console source code is provided with the SDK and the customer must write the console code to display their information and write the code to allow the operator to change their configuration parameters by sending messages from the console to the

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

customer's software in the Black Box Trading System. The SKD Library API provides functions to allow the customer's software to receive messages from a console, send a message to all console, and respond to a console request message by sending a response message to the same console that sent the request message. See the Console Interface Functions section for more information.

1.5. Account Modes

The SDK Library provides access to the Lightspeed trading environment. The Lightspeed trading environment provides a set of services that allow customers to deploy Black Box Trading Systems. These services include: market data service, order entry service, risk management, short availability service, etc.. Lightspeed's order processing server is referred to as the Lightspeed Gateway or just Gateway.

The Gateway 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 Black Box 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.x). When Multi-Account mode is used, all account numbers that will be traded must also be specified in the configuration file. When the SDK Library connects to the Gateway, it will receive all account numbers the Gateway is configured with. The SDK 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 SDK Library will log an error and exit. The Black Box Trading System cannot be used to trade until the account numbers have been configured correctly.

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

When the Black Box Trading System is configured for Multi-Account mode, the console provided with the SDK will display information for a single account, or the aggregation of all account information. The term "Aggregate Account" is used to describe when the console is displaying the aggregation of all account information. For

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

example, if there are two accounts and one account's P&L is $1000 and the other account's P&L is $500, then $1500 will be displayed when displaying the aggregate account (the aggregation of all account information).

The console supports the concept of an "active account". The active account can be an individual account or the aggregate account. When the active account is an individual account, then only the information for the individual account is displayed and manual orders entered using the console will be associated with the active account

Some console functions are not allowed when the active account is the aggregate account. For example, entering manual orders is not allowed when the active account is set to the aggregate account. An order must have a valid account number associated with it; therefore, orders can only be entered when the active account is an individual account.

When the SDK Library queries the customer's software to determine what to display on the console, the customer's software will be passed the account number of the active account. The customer's software should respond to the query with the information for the specified account. If the console's active account is set to the aggregate account, then the customer's software will be passed an account number that identifies the aggregate account. The account number 9999999999 is used to identify the aggregate account. The customer's software should respond to the query with the aggregation of information for all accounts. The customer's software can chose to respond to a query for the aggregate account with "Not Available" if the aggregation of account information cannot be expressed as a single value. For example, if there are two accounts and one account is configured to only acquire long position and the other account is configured to only acquire short positions, then when queried for the Trading Side of the aggregate account a meaningful response is not possible. In this case the customer's software may chose to respond with "Not Available". Responding with "Not Available" will cause the console to display "N/A". For example, (Trading Side: N/A).

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

2. System Functions

2.1. initialize() – Callback Function

void initialize(char process_id)

This function is called by the SDK Library when the Black Box Trading System is started. The customer's software should perform any required initialization. At a minimum the customer software must initialize all variables that can be queried by the SDK Library using **query** functions defined below. The SDK Library will immediately start to query the customer software for its states so that it can update all trading consoles.

The following parameter is passed when this function is called.

process_id: The process identifier that is entered on the command line when the Black Box is started. Valid values are '0' through '9'.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

2.2. update_global_trading_state() – Callback Function

void update_global_trading_state(char state, double account)

This function is called by the SDK Library when the operator has changed the global trading state using the console. The global trading state affects all securities. The following trading states are defined by the SDK Library:

TRADE – Indicates that automated trading is enabled.

DON'T_TRADE – Indicates that automated trading is disabled.

LIQUIDATE_ONLY – Indicates that automated trading should only attempt to cover existing positions.

BAIL_OUT – Indicates that the operator wants to cover all positions quickly. The SDK Library will attempt to cover all open positions by sending one order for each open position. The SDK software will send a routable RASH order five cents through the market in an attempt to get filled. Only one order will be sent for each position. The console operator may need to instruct the Black Box Trading System to "Bail Out" more than once to fill all positions. This is NOT a market order and will not chase the market.

The following parameters are passed when this functions is called.

state Specifies the desired global trading state.

Value	Description
'T'	TRADE
'D'	DON'T_TRADE
'L'	LIQUIDATE_ONLY
'B'	BAIL_OUT

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

2.3. query_global_trading_state() – Callback Function

char query_global_trading_state(double account)

This function is called by the SDK Library to determine the Global Trading State of the customer software. The value returned by this function will be displayed on the console. The console will use this value to determine what to display for Global Trading State. The customer's software should maintain this state for trading. **The customer software should initialize this state immediately so that the console can be properly updated.**

The following table indicates the values that can be returned by this function.

Value	Description
'T'	TRADE
'D'	DON'T_TRADE
'L'	LIQUIDATE_ONLY
'B'	BAIL_OUT
'N'	NOT AVAILBLE The Console will display N/A.

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

2.4. update_single_stock_trading_state() – Callback Function

void update_single_stock_trading_state(char symbol[], char state, double account)

This function is called by the SDK Library when the console operator has changed the trading state for a single security. The following trading states are defined by the SDK Library:

TRADE – Indicates that automated trading is enabled.

DON'T_TRADE – Indicates that automated trading is disabled.

LIQUIDATE_ONLY – Indicates that automated trading should only attempt to cover an existing position.

The following parameters are passed when this function is called.

symbol: Specifies the security whose trading state is being updated.

state Specifies the global trading state.

Value	Description

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

‘T’	TRADE
‘D’	DON’T_TRADE
‘L’	LIQUIDATE_ONLY

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

2.5. query_single_stock_trading_state() – Callback Function

char query_single_stock_trading_state(char symbol[], double account)

This function is called by the SDK Library to determine the customer software’s Trading State for a single security. The value returned by the function will be displayed on the console. The console will use this value to determine what to display for the security’s Trading State. **The customer software should initialize the trading state of all stocks so the console can be properly updated.**

The following parameter is passed when this functions is called.

symbol: Specifies the security whose trading state is being queried.

The following table specifies the values that can be returned by this function.

Value	Description
‘T’	TRADE
‘D’	DON’T_TRADE
‘L’	LIQUIDATE_ONLY
‘N’	NOT AVAILBLE The Console will display N/A.

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

2.6. update_allowed_trading_side() – Callback Function

void update_allowed_trading_side(char allowed_side, double account)

This function is called by the SDK Library when the console operator has changed the allowed trading side. The trading side affects all securities. The following trading sides are defined by the SDK Library:

EITHER – Indicates that both long and short positions can be acquired.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

LONG_ONLY – Indicates that only long positions can be acquired.

SHORT_ONLY - Indicates that only short positions can be acquired.

The following parameter is passed when this function is called.

allowed_side: Specifies the allowed trading side.

Value	Description
'E'	EITHER
'L'	LONG_ONLY
'T'	SHORT_ONLY

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

2.7. query_allowed_trading_side() – Callback Function

char query_allowed_trading_side(double account)

This function is called by the SDK Library to determine the Trading Side the customer's software is allowing. The value returned by the function will be displayed on the console. The console will use this value to determine what to display for the Allowed Trading Side. **The customer software should initialize this value immediately so the console can be properly updated.**

The following table specifies the values that can be returned by this function.

Value	Description
'E'	EITHER
'L'	LONG_ONLY
'T'	SHORT_ONLY
'N'	NOT AVAILBLE The Console will display N/A.

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

2.8. enable_trading_letter_range() – Callback Function

void enable_trading_letter_range(char start, char end, double account)

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

This function is called by the SDK Library when the operator has enabled trading for securities that start with the specified letter range. For example, the operator may want to enable trading for securities that start with letters 'A' through 'D'. To enable only one letter, the start and end parameter will be set to the same letter.

The following parameters are passed when this function is called.

start: Specifies the starting letter in the range.

end: Specifies the ending letter in the range.

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

2.9. disable_trading_letter_range() – Callback Function

void disable_trading_letter_range(char start, char end, double account)

This function is called by the SDK Library when the operator has disabled trading for securities that start with the specified letter range. For example, the operator may want to disable trading for securities that start with letters 'A' through 'D'. To disable only one letter, the start and end parameter will be set to the same letter.

The following parameters are passed when this function is called.

start: Specifies the starting letter in the range.

end: Specifies the ending letter in the range.

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

2.10. query_trading_letter_state() – Callback Function

void query_trading_letter_state(char state[], double account)

This function is called by the SDK Library to determine if the customer's software is enabled or disabled for trading for securities that start with each letter. This function returns a 26 byte string where each byte represents the state for one letter. For example, element 0 contains the trading state for securities that start with 'A', element 1 contains the trading state for securities that start with 'B', etc. The following trading states are defined.

Value	Description
'E'	Trading is enabled.
'D'	Trading is disabled.
'N'	NOT AVAILBLE The Console will display will display the Trading Letters in Blue if those letters where the state is 'N'.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

If securities that start with A, B and C are disabled, and D through Z are enabled, then the following string would be returned by this function.

DDDEEEEEEEEEEEEEEEEEEEEEEE

The customer software should immediately initialize the trading letter array so the console can be properly updated.

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

2.11. num_console_update() – Callback Function

void num_console_update(long num_consoles)

This function is called by the SDK Library to inform the customer's software of the number of consoles connected to the Black Box Trading System. This callback function is called when the SDK Library accepts a console connection, which will increase the number of consoles, or when a console connection is terminated, which will decrease the number of consoles. A typical use of this function is to allow the customer's software to determine if it wants to continue trading when all console connections are terminated.

The following parameter is passed when this function is called.

num_consoles: The number of console connections

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

2.12. update_console_down_mode() – Callback Function

void update_console_down_mode(char mode, double account)

This function is called by the SDK Library when the operator has changed the "Console Down Mode". Console Down Mode refers to whether or not automated trading should continue when no consoles are connected. The following Console Down Modes are defined by the SDK Library:

TRADE – Continue to trade when no consoles are connected.

STOP_TRADING – Stop trading when no consoles are connected.

LIQUIDATE_ONLY – Continue trading out of positions only.

The following parameter is passed when this function is called.

mode: Specifies the Console Down Mode.

Value	Description
'T'	Continue to trade when no consoles are connected.
'S'	Stop trading when no consoles are connected.
'L'	Liquidate current positions only.

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

2.13. query_console_down_mode() – Callback Function

char query_console_down_mode(double account)

This function is called by the SDK Library to determine the Console Down Mode the customer's software is currently supporting. The value returned by the function will be sent to the console. The console will use this value to determine what to display for Console Down Mode. **The customer software should initialize this value immediately so the console can be properly updated.**

The following table indicates the values that can be returned by this function.

Value	Description
'T'	Continue to trade when no consoles are connected.
'S'	Stop trading when no consoles are connected.
'L'	Liquidate current positions only.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

'N'	NOT AVAILBLE The Console will display N/A.

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

2.14. trailing_stop_loss_exceeded() – Callback Function

void trailing_stop_loss_exceeded(double high_pnl, double dpnl, double account)

The SDK configuration file contains a configuration parameter called "**trailing-stop-loss-amt**". The SDK Library will monitor booked P&L and exposure, and will call this function periodically if that value exceeds the day's high booked P&L amount minus the trailing-stop-loss-amt value specified in the configuration file. The stop loss function works as a stop loss mechanism that uses an account's high booked P&L mark as the basis for the stop loss. The high P&L mark is REALIZED P&L only and starts the day at $0 and will only move **UP** from there. If the high mark booked PNL minus the "current booked P&L minus current exposure" is lower than the stop-loss-amt then this callback function will inform the customer software that the P&L has dropped to the point of warranting this call. It is up to the customer software to make any necessary trading adjustments. This call is simply informative and will not result in any SDK generated system trading modifications.

Using the console, the operator can change the **trailing-stop-loss-amt** value. Changing the **trailing-stop-loss-amt** value will affect whether or not this function is called. For example, assume the **trailing-stop-loss-amt** value specified in the configuration file is $1,000. The high mark booked P&L climbs to $3000, then the trading account either loses or has a bad position of $1,100 dollars, so the current day account value (booked P&L - exposure) is $1900. This will cause the SDK Library to periodically (eg, every 5 seconds) call this function to inform the customer's software that the **trailing-stop-loss-amt** has been exceeded. The operator may change the trailing-stop-loss-amt value to $2,000 using the console. This would cause the SDK Library to stop calling the function because the loss ($1,100) is now less than the stop-loss-amt value ($2,000).

Exposure is defined as the value that would be **SUBTRACTED** from the account value if the current positions were covered by paying the spread. By this definition a negative exposure value would add P&L to the account.

The following parameters are passed when this function is called.

high_pnl: This is the high water mark for booked P&L. Note, this number is represented as a double. If this parameter contains 234.67, then the high water mark for booked P&L is two hundred thirty four dollars and sixty seven cents.

dpnl: The day's current booked P&L minus the current exposure. Note, this number is represented as a double. If this parameter contains 25.45, then the current booked P&L is twenty five dollars and forty five cents.

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

2.15. trailing_stop_loss_amout_modified() – Callback Function

void trailing_stop_loss_amount_modified(double new_amount, double account)

This callback function is called by the SDK library to inform the customer's software that the trailing stop loss amount has been modified. Please see **trailing_stop_loss_exceeded()** for a detailed explanation of the trailing stop loss amount mechanism.

The following parameter is passed when this function is called.

new_amount: This value is a negative number representing the amount of drop from the booked P&L high water mark the operator has specified before the SDK library alerts the customer software. Note, this number is represented as a double. If this parameter contains -500.00, then the stop loss amount is negative five hundred dollars.

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

2.16. add_one_stock() – Callback Function

void add_one_stock(char symbol[], char type, double account)

This function is called by the SDK Library to inform the customer software that the console operator has added a security using the "Add 1 Stock" console feature. Note, the "Add 1 Stock" console feature informs the SDK Library that a security has been added. The SDK Library needs to know that a new symbol has been added so that it can register to receive market data for the new security. Typically, securities are specified in the symbol.list file. However, the symbol.list file is only read when the Black Box Trading System is started. The "Add 1 Stock" console feature can be used to add securities after the Black Box Trading System has been started. The add_one_stock() function does NOT add stocks to the symbol_list file. The operator must manually add the stock to the file if the stock is to be permanently added.

This callback function is used to inform the customer software that a symbol has been added. However, this callback function only provides the security symbol and symbol type. To actually trade this symbol, the customer software may need additional information. If the customer software needs additional trading related information, then the customer must write the code to send the trading information from the console to the customer Black Box software.

The following parameters are passed when this function is called.

symbol: Specifies the security that is being added.

type: Specifies the type of security. The following security types are defined:

Value	Description
'O'	OTC Security

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

‘L’	Listed Security
‘E’	ETF

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

2.17. query_version() – Callback Function

void query_version(char version[])

This function is called by the SDK Library to determine the version of the customer software. This function will return a string that represents the version of the customer software. The value returned by the function will be displayed on the console as the customer’s software version. **There is limited space to display the version on the console, so this function should return a version that is 8 characters or less**. **This value should be immediately initialized so that the console can be properly updated.**

2.18. system_shutdown() – Callback Function

void system_shutdown(void)

This function is called by the SDK Library when the operator has requested that the Black Bock Trading System be shutdown (terminate program). The customer’s software should perform any processing required to shutdown.

2.19. get_account_numbers() – Function Call

void get_account_numbers(double account_array[], long array_size)

This function can be called by the customer software in order to get a list of account numbers that are active on this gateway connection if the gateway is configured in multi-account mode. This function will return an account number in each array element. If the array specified by the customer software is larger than the number of active accounts, the value negative one (-1) will be placed in unused element. If the gateway is configured to single-account mode, all array elements returned will be negative one (-1).

The following parameters are passed when this function is called.

account_array: Specifies the array of doubles where the SDK software will place the account numbers.

array_size: Specifies the number of elements (doubles) in the array passed to this routine.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

2.20. get_account_mode() – Function Call

char query_account_mode()

This function can be called by the customer software in order to find out if the configured gateway is set for single account mode or multi-account mode.

This function **returns** '**S**' for single account mode or '**M**' for multi-account mode.

2.21. get_fee_info() – Function Call

double get_fee_info(char fee_type[])

This function can be called by the customer's software to determine the fees that are defined in the SDK configuration file. The SDK configuration file contains the fees that a customer is charged. This function is passed a parameter that specifies the fee type that the being requested, and returns the value read from the sdk_cfg.x file. If an unknown value is passed in, the routine returns ERROR_SDK (-1). The table below define the valid fee_type values.

The following parameter is passed when this function is called.

fee_type: Specifies the fee type that is to be returned. The fee type is represented as a string. The following fee type strings can be passed to this function.

Value	Description
"sec-fee-rate"	The SEC fee rate.
"commission-rate"	Lightspeed commission rate.
"inet-add-liquidity-fee"	INET add liquidity fee rate.
"inet-remove-liquidity-fee"	INET remove liquidity fee rate.
"arca-add-liquidity-fee"	ARCA add liquidity fee rate.
"arca-remove-liquidity-fee"	ARCA remove liquidity fee rate.
"rash-add-liquidity-fee"	RASH add liquidity fee rate.
"rash-remove-liquidity-fee"	RASH remove liquidity fee rate.
"amex-add-liquidity-fee"	AMEX add liquidity fee rate.
"amex-remove-liquidity-fee"	AMEX remove liquidity fee rate.
"nyse-add-liquidity-fee"	NYSE add liquidity fee rate.
"nyse-remove-liquidity-fee"	NYSE remove liquidity fee rate.
"bats-add-liquidity-fee"	BATS add liquidity fee rate.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

“bats-remove-liquidity-fee”	BATS remove liquidity fee rate.
“edgx-add-liquidity-fee”	EDGX add liquidity fee rate.
“edgx-remove-liquidity-fee”	EDGX remove liquidity fee rate.
“edga-add-liquidity-fee”	EDGA add liquidity fee rate.
“edga-remove-liquidity-fee”	EDGA remove liquidity fee rate.
“bost-add-liquidity-fee”	BOST add liquidity fee rate.
“bost-remove-liquidity-fee”	BOST remove liquidity fee rate.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

3. Start/Stop Trading Functions

The SDK Library provides tools that allow the customer software to automatically start and stop trading based on start and stop times defined in the configuration file (**svr_cfg.cfg.x, where x is the black box runtime parameter value. Eg. 6**). The SDK Library will call the customer software using a callback function at the start and stop times defined in the configuration file. This will allow the customer software to automatically start or stop trading if desired. The SDK Library will simply inform the customer software that the start and stop times have been reached. It is up to the customer's software to decide whether to actually start or stop trading when informed.

The console has a button that allows the operator to inform the customer's Black Box software whether or not to start and stop trading when informed that the start and stop times have occurred. Two trading modes are defined, Auto and Manual. Using the console, the operator can inform the customer's software, via a callback function, of the desired trading mode. Auto mode indicates the operator would like the customer's Black Box software to start and stop trading when informed that the start and stop times have occurred. Manual mode indicates the operator would like the customer's Black Box software to NOT start and stop trading when informed that the start and stop times have occurred. In Manual mode the operator would prefer to manually start and stop trading.

The following callback functions are used to assist the customer in implementing automated trading.

3.1. start_time_event() – Callback Function

void start_time_event(double account)

This function is called by the SDK Library at the start time specified in the configuration file.

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

3.2. stop_time_event() – Callback Function

void stop_time_event(double account)

This function is called by the SDK Library at the stop time specified in the configuration file.

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

3.3. update_trading_mode() – Callback Function

void update_trading_mode(char mode, double account)

This function is called by the SDK Library when the operator has change the trading mode using the console. Two trading modes are defined, Auto and Manual. See the discussion above for more information.

The following parameter is passed when this functionsis called.

mode: Specifies the trading mode desired by the operator.

Value	Description
'M'	Manual
'A'	Auto

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

3.4. query_trading_mode () – Callback Function

char query_trading_mode(double account)

This function is called by the SDK Library to determine the Trading Mode the customer's software is currently supporting. See the discussion above for more information about Trading Mode. The value returned by the function will be used by the console to determine what to display for Trading Mode.

The following table specifies the values that can be returned by this function.

Value	Description
'M'	Manual
'A'	Auto

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

4. Market Data Functions

4.1. quote_event_received() – Callback Function

void quote_event_received(char symbol[], long event_type, long bid_price, long bid_size, long ask_price, int ask_size, long event_time)

This function is called by the SDK Library every time a market data message is received. The market data protocol used by most venues has three message types: New Order, Modify Order, and Delete Order. This function is called every time one of these message is received. This gives the customer's software an opportunity to make trading decisions every time a market data message is received.

This function passes the current inside price and quantity as well as an event_type that indicates how the inside price and quantity have changed. In many cases, the inside price and quantity will not change. For example, if a new buy order is received below the inside bid price, then the inside price and quantity will not have changed since the last time this function was called.

The following parameters are passed when this function is called.

symbol: The security the market data message is associated with. For example, AAPL.

event_type: Indicates how the inside price and quantity have changed.

Value	Description
0	NO_INSIDE_CHANGE
1	INSIDE_BID_PRICE_CHANGE_ONLY
2	INSIDE_BID_SIZE_CHANGE_ONLY
3	INSIDE_BID_PRICE_AND_SIZE_CHANGE
4	INSIDE_ASK_PRICE_CHANGE_ONLY
5	INSIDE_ASK_SIZE_CHANGE_ONLY
6	INSIDE_ASK_PRICE_AND_SIZE_CHANGE

bid_price: The current inside bid price.

bid_size: The current inside bid quantity.

ask_price: The current inside offer price.

ask_size: The current inside offer quantity.

event_time: The time the event occurred at the Exchange or ECN, expressed in milliseconds past midnight.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

4.2. get_book_depth() – Function Call

long get_book_depth(char symbol[], data_book_t **bid_book, data_book_t **ask_book)

This function can be called by the customer's software to obtain access to the depth of book. To improve performance, the customer's software obtains pointers to the book maintained by the SDK 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 following parameters are passed when this function is called.

symbol: The security the customer's software want to obtain access to the book for.

bid_book: A pointer to a pointer. The SDK Library will return a pointer to the bid book in this parameter.

ask_book: A pointer to a pointer. The SDK Library will return a pointer to the offer book in this parameter.

Returns

The depth of book is returned by this function.

If an error occurs, then –1 is returned.

Example Code

The SDK Library maintains the 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 book using array indexes. The following code sample demonstrates how to do this. The code sample will log the bid and offer prices for the entire depth of the book.

```
depth = get_book_depth(symbol, &bid_book, &ask_book);
for(i=0; i<depth; i++)
{
        log_data(0,"Bid Price: %ld   Ask Price: %ld\n",  bid_book[i].i_price, ask_book[i].i_price);
}
```

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

4.3. trade_received() – Callback Function

void trade_received(char symbol[], long trade_price, long trade_size, char sale_condition, long tot_daily_volume, long daily_high, long daily_low, char market_center_code, long trade_time)

This function is called by the SDK Library to inform the customer's software that a trade reported by the NASDAQ via the UTDF feed or a trade reported by the NYSE via the CTS feed has occurred. The customer's software is also informed of the total daily trade volume, and the daily high and low trade price.

The following parameters are passed when this function is called.

symbol: The security the trade occurred on.

trade_price: The price of the last trade.

trade_size: The size of the last trade.

sales_condition: This is taken directly form the data feed and indicates the type of trade. See table below:

Value	Description
	The following conditions apply to all participants:
'@'	Regular Sale
'A'	Acquisition
'B'	Bunched Trade
'C'	Cash Trade
'D'	Distribution
'K'	Rule 155 Trade
'L'	Sold Last
'N'	Next Day
'O'	Opened
'P'	Prior Reference Price
'R'	Seller
'T'	Form-T Trade
'W'	Average Price Trade
'Z'	Sold (Out of Sequence)
	The following conditions apply only to Nasdaq participants (including UTP):

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

‘G’	Bunched Sold Trade
‘S’	Split Trade
	The following conditions apply only to UTP participants within Nasdaq:
‘M’	Market Center Close Price
‘1’	Stopped Stock – Regular Transaction
‘2’	Stopped Stock – Sold Last
‘3’	Stopped Stock – Sold (Out of Sequence)
	The following conditions apply only to NYSE & AMEX:
‘E’	Automatic Execution
‘F’	Burst Basket Execution
‘G’	Opening / Reopening Trade Detail
‘H’	Intraday Trade Detail
‘I’	Basket Index on Close Transaction

tot_daily_volume: The total shares traded in the current day.

daily_high: The highest price the security has traded at in the current day.

daily_low: The lowest price the security has traded at in the current day.

market_center_code: Specifies where the trade occurred. See table below:

Value	Description
‘A’	American Stock Exchange
‘B’	Boston Stock Exchange
‘C’	Cincinnati Stock Exchange
‘D’	National Association of Securities Dealers Alternative Display Facility
‘E’	No associated morket center (handler generated)
‘M’	Chicago Stock Exchange
‘N’	New York Stock Exchange
‘P’	Archipelago / Pacific Exchagne
‘Q’	NASDAQ
‘T’	National Association of Securities Dealers

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

‘U’	OTC Bulletin Board
‘u’	OTC Non-Nasdaq Issue
‘X’	Philadelphia Stock Exchange
‘Z’	BATS Exchange Inc.

trade_time: The time the traded occurred at the Exchange or ECN, expressed in milliseconds past midnight.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

4.4. execution_report_received() – Callback Function

void execution_report_received(char symbol[], long price, long size, char venue[], char visible, long execution_time)

This function is called by the SDK Library to inform the customer's software that an execution report message was received. Some venues such as INET and BATS send a message when an order is executed. ARCA does not report when an order is filled. ARCA simply reports that the order is dead, and does not differentiate between a canceled order and an executed order. The execution reported in this message will also be reported via the NASDAQ UTDF feed or the NYSE CTS feed, which the customer will be informed of via the trade_received() function.

Some venues support "hidden" orders, which are orders that are not visible on the book. The visible parameter indicates whether or not the execution was for an order that was visible. A value of 'N' indicates the execution occurred on an order that is not visible on the venue's book. A value of 'V' indicates the execution occurred on an order that is visible on the venue's book.

The following parameters are passed when this function is called.

symbol: The security the execution report occurred on.

price: The price the execution occurred at.

size: The number of shares executed.

venue: The venue where the execution occurred. For example, INET.

visible: A value of 'N' indicates the execution occurred on an order that is not visible on the venue's book. A value of 'V' indicates the execution occurred on an order that is visible on the venue's book.

Value	Description
'V'	Visible
'N'	Not visible

execution_time: The time the execution occurred at the Exchange or ECN, expressed in milliseconds past midnight.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

4.5. mkt_data_status_event() - Callback Function

void mkt_data_status_event(long event_type, char venue[], char symbol[])

This function is called by the SDK Library to inform the customer's software of "events" related to the Market Data System. Examples of events are a connection to a market data source has been made or dropped, the SDK Library is registering to receive market data, inactivity has been detected on a connection, etc. Some event types require additional information to be meaningful. For some event types the venue (eg, INET) is important. For other event types the venue and symbol are important. This function passes three parameters, event type, venue and symbol. The following table describes the event types and indicates for which events the venue and symbol are meaningful.

For example, a CONNECTION_DROPPED event with the venue field equal to "INET" indicates that the connection to INET data source was dropped. A REGISTERING event with the venue field equal to "BATS" and the symbol equal to "AAPL" indicates the SDK Library has register to receive market data for AAPL from the BATS data source.

Event Type	Description	Venue	Symbol
CONNECTION_MADE	A connection has been established with a data source.	Meaningful	Not Meaningful
CONNECTION_DROPPED	The connection to a data source has been dropped	Meaningful	Not Meaningful
REGISTERED	The SDK Library has registered with the data source to receive market data.	Meaningful	Meaningful
DEREGISTERED	The SDK Library has deregistered with the data source and no longer want to receive market data.	Meaningful	Meaningful
INACTIVITY_DETECTED	Data has not been received from a data source for a period of time. The SDK Library suspects there may be a problem and will remove that venue's data from the book.	Meaningful	Not Meaningful
EXCESSIVE_LATENCY	Data latency refers to the amount of time it takes for a message to travel from the data source to the Black Box Trading System. The data source will put a time stamp in the message indicating the time the message was sent. Latency is determined by comparing the time the Black Box Trading System received the message with the time stamp in the message. If the latency becomes "excessive", then data from that data source is removed from the book. Removing late or stale data from the book ensures that the book is kept as accurate	Meaningful	Not Meaningful

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

	as possible. The amount of time that is considered excessive is specified as a configuration parameter. When the SDK Library detects excessive latency it will remove all data for the specified venue from the book and notifiy the customer's software by calling this function and setting event_type to EXCESSIVE_LATENCY.		
ACCEPTABLE_LATENCY	When the SDK Library removes a venue's data from the book due to excessive latency, it will continue to monitor the latency. When the latency becomes acceptable, the data will be restored to the book. The SDK Library will notify the customer's software by calling this function and setting event_type to ACCEPTABLE_LATENCY.	Meaningful	Not Meaningful

The following parameters are passed when this function is called.

event_type: Specifies the type of event (see the table above). Valid values include:

Value	Description
1	CONNECTION_MADE
2	CONNECTION_DROPPED
3	REGISTERED
4	DEREGISTERED
5	INACTIVITY_DETECTED
6	EXCESIVE_LATENCY
7	ACCEPTABLE_LATENCY

venue: The venue the event pertains to (eg, "INET", "ARCA", "ALST", "AETF", "BATS", "EDGX", "EDGA", "INDX").

symbol: The security the event is associated with.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

4.6. get_mkt_data_connection_state() – Function Call

long get_mkt_data_connection_state(char venue[])

This function can be called by the customer's software to obtain the state of a connection with a data source. A connection can be in one of the following states: up (a connection has been established with the data source), down (a connection has not been established with the data source), or in progress (the SDK Library is in the process of establishing a connection with the data source).

The following parameters is passed when this function is called.

venue: A four character identifier for the data source. The following venue identifiers are used:

Value	Description
"PEQS"	Prints and Exchange Quotes data feed.
"INET"	INET ECN-MD data feed..
"ARCA"	ARCA OTC ECN-MD data feed.
"ALST"	ARCA Listed ECN-MD data feed.
"AETF"	ARCA ETF ECN-MD data feed.
"BATS"	BATS ECN-MD data feed.
"EDGX"	EDGX ECN-MD data feed.
"EDGA"	EDGA ECN-MD data feed.
"INDX"	Futures (and Indices) data feed.
"IMBL"	Imbalance data. Note, Imbalance data is obtained from the Prints and Exchange Quotes data server. Specifying "IMBL" will retrun the same value as specifying "PEQS".

Returns

1 – UP, a connection has been established with the data source.

2 – Down, a connection has not been established with the data source.

3 – In Progress, the SDK Library is in the process of establishing a connection with the data source.

-1 – An error was encountered. For example, an unrecognized venue was passed as a parameter.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

4.7. get_symbol_registered_state() – Function Call

long get_symbol_registered_state(char symbol[], char venue[])

This function can be called by the customer's software to determine whether or not the SDK Library has registered to receive data from the specified venue for the specified security. For example, to determine if the SDK Library has register to receive IBM data from the ARCA Listed data source, then the symbol parameter should be set to "IBM", and the venue parameter should be set to "ALST".

The following parameters are passed when this function is called.

symbol: The security the customer wants to determine if it has been registered with the data source.

venue: A four character identifier for the data source. The following venue identifiers are used:

Value	Description
"PEQS"	Prints and Exchange Quotes data feed.
"INET"	INET ECN-MD data feed..
"ARCA"	ARCA OTC ECN-MD data feed.
"ALST"	ARCA Listed ECN-MD data feed.
"AETF"	ARCA ETF ECN-MD data feed.
"BATS"	BATS ECN-MD data feed.
"EDGX"	EDGX ECN-MD data feed.
"EDGA"	EDGA ECN-MD data feed.
"INDX"	Futures (Indices) data feed.
"IMBL"	Imbalance data. Note, registering to receive Imbalance data is not done on an individual symbol basis. When the SDK Library register to received Imbalance data, it will received Imbalance data for all symbols. This function will ignore the symbol parameter when the venue parameter is set to "IMBL".

Returns

1 – Registered

2 – Not Registered

3 – Registration in progress

-1 - An error was encountered. For example, an unrecognized symbol or venue was passed as a parameter.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

-2 – An invalid combination of parameters was specified. For example, setting the symbol to "IBM" and the venue to "ARCA" will result in –2 being returned. IBM is a listed security and cannot be registered on the ARCA OTC data source. IBM can only be registered on the ARCA Listed data source, which is identified as "ALST".

4.8. get_daily_high_price() – Function Call

long get_daily_high_price(char symbol[])

This function can be called by the customer's software to obtain the highest price traded for the current day.

The following parameter is passed when this function is called.

symbol: The security the customer want to obtain the daily high price for.

Returns

The highest price traded for the current day.

If an error occurs, then –1 is returned.

4.9. get_daily_low_price() – Function Call

long get_daily_low_price(char symbol[])

This function can be called by the customer's software to obtain the lowest price traded for the current day.

The following parameter is passed when this function is called.

symbol: The security the customer want to obtain the daily low price for

Returns

The lowest price traded for the current day.

If an error occurs, then –1 is returned.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

4.10. get_daily_volume() – Function Call

long get_daily_volume(char symbol[])

This function can be called by the customer's software to obtain the daily volume for the current day.

The following parameter is passed when this function is called.

symbol: The security the customer want to obtain the daily volume for.

Returns

The total daily volume traded for the current day.

If an error occurs, then –1 is returned.

4.11. get_previous_day_close_price() – Function Call

long get_previous_day_close_price(char symbol[])

This function can be called by the customer's software to obtain the closing price for the previous day.

The following parameter is passed when this function is called.

symbol: The security the customer want to obtain the previous day closing price for.

Returns

The previous day's closing price.

If an error occurs, then –1 is returned.

4.12. trading_halted() – Callback Function

void trading_halted(char symbol[])

This function is called by the SDK Library to inform the customer's software that trading for a security has been halted.

The following parameter is passed when this function is called.

symbol: The security that trading has been halted.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

4.13. trading_resumed() – Callback Function

void trading_resumed(char symbol[])

This function is called by the SDK Library to inform the customer's software that trading has resumed for a security.

The following parameter is passed when this function is called.

symbol: The security that has resumed trading.

4.14. quotation_only_received() – Callback Function

void quotation_only_received(char symbol[])

This function is called by the SDK Library to inform the customer's software that a security has entered the Quotation Only Period. Before a halted security will resume trading, there may be a period of time when market participants can enter orders. However, orders will not be filled until trading resumes.

The following parameter is passed when this function is called.

symbol: The security that has entered the Quotation Only Period.

4.15. get_halt_resume_state() – Function Call

long get_halt_resume_state (char symbol[])

This function can be called by the customer's software to determine whether a security is trading normally, halted, or in quotation only period.

The following parameter is passed when this function is called.

symbol: The security the customer wants to determine the trading state for.

Returns

1 – Trading normally.

2 – Trading is halted

3 – Quotation Only Period

If an error occurs, then –1 is returned.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

4.16. set_inactivity_time() – Function Call

void set_inactivity_time(long time)

This function can be called by the customer's software to set the market data inactivity time. The SKD Library monitors message received from data sources. If no messages are received for a specified amount of time, then the SDK Library suspects there may be a problem and will remove that data source's data from the book. Several factors should be considered when choosing a timeout value. If the customer's software has registered for a lot of high volume securities, then more messages will be received and the timeout value can be set to a smaller value. If only a few low volume securities have been register for, then a larger timeout value is required to prevent timing out during periods of low activity.

The following parameter is passed when this function is called.

time: The inactivity time in seconds. If no message are received for the number of seconds specified in this parameter, then the SDK Library will remove the data source's data from the book.

4.17. lrp_received() – Callback Function

void lrp_received(char symbol[], long low_lrp, long high_lrp)

This function is called by the SDK Library to inform the customer's software that an NYSE Liquidity Replenishment Points messages have been received.

Liquidity Replenishment Points (LRP) are pre-determined price points at which electronic trading briefly converts to auction market trading. LRP's may be triggered by an electronic sweep or if electronic trading results in rapid price movement over a short period of time. Refer to the NYSE Liquidity Replenishment Points Customer Interface Specification for more details.

The following parameters are passed when this function is called.

symbol: The security the LRP information is for.

low_lrp: Low LRP Price.

high_lrp: High LRP Price.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

4.18. imbalance_received() – Callback Function

**void imbalance_received(char symbol[], char action, char market, char cross_type,
char regulatory_imbalance, long buy_volume, long sell_volume,
long tot_volume, long price_reference, long clearing_price,
long near_price, long far_price)**

This function is called by the SDK Library to inform the customer's software that an Imbalance messages have been received. Refer to the NYSE Imbalance feed Customer Interface Specification and the NASDAQ NOIView Specification for more information.

The following parameters are passed when this function is called.

symbol: The security the Imbalance information is for.

action: Specifies whether or not the Imbalance data is valid. Valid values include:

Value	Description
'+'	Indicates the Imbalance data is valid and can be used
'-'	Indicates the Imbalance data in should no longer be used. For example, after 4:00pm EST or after the stock opens in the morning, this callback function will be called with a '–' indicating that the Imbalance information should no longer be used.

market: Specifies the source of the Imbalance information. Valid values include:

Value	Description
'N'	Indicates the Imbalance data is from NYSE.
'Q'	Indicates the Imbalance data is from NASDAQ.

cross_type: Indicates the Cross Type. Valid values include:

Value	Description
'O'	Open cross
'C'	Closing cross.
'H'	Intraday opening cross for IPO

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

regulatory_imbalance: This parameter is used to differentiate between NYSE regulatory and informational Imbalance data. Valid values include:

Value	Description
‘0’	NYSE stocks at the open
‘1’	NYSE stock at the close, regulaory
‘0’	NYSE stock at the close, informational
‘0’	NASDAQ stocks at the open and close

buy_volume: Shares required to be paired off on the buy side.

sell_volume: Shares required to be paired off on the sell side

tot_volume: The total volume traded for the day.

price_reference: Price at which the Imbalance is being calculated.

clearing_price: The closest price to the reference price (price_reference) where the Imbalance is zero.

near_price: Hypothetical auction clearing price for cross orders only.

far_price: Hypothetical auction clearing price for cross orders as well as continuous orders.

Notes:

When the source of the Imbalance data is the NASDAQ, the values passed in the buy_volume and sell_volume parameters are calculated as follows:

```
If (imbalanceDirection == ‘B’)
{
        buy_volume = pairedShares + ImbalanceShares;
        sell_volume = pairedShares;
}
else If (imbalanceDirection == ‘S’)
{
        sell_volume = pairedShares + ImbalanceShares;
        buy_volume = pairedShares;
}
else
{
        sell_volume = 0;
        buy_volume = 0;
}
```

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

5. Futures Data and Indices Functions

The SDK Library has access to the Lightspeed Futures Market Data Server. The Lightspeed Futures Market Data Server receives data feeds directly from Globex and other aggregators such as Reuters and S&P. The Lightspeed Futures Market Data Server distributes Futures and Indices data to clients with minimal latency and high throughput.

The SDK Library will connect to the Lightspeed Futures Market Data Server and make the data available to the customer's software using the callback functions described below. Customers should refer to Appendix A of the Lightspeed Futures Market Data Server specification for a list of the available Indices and Futures contracts.

Note, at this time, customers cannot trade Futures contracts using the SDK. Only the Futures data is provided. Futures data and Indices are provided and can be used by customers in their equities trading strategies.

5.1. fut_data_update() – Callback Function

void fut_data_update(char symbol[], long bid_price, long bid_size, long ask_price, long ask_size)

This function is called by the SDK Library to inform the customer's software that a Futures market data message has been received

The following parameters are passed when this function is called.

symbol: The symbol representing the Futures data received.

bid_price: The best bid price for the Futures contract.

bid_size: The number of contracts available at the best bid price.

ask_price: The best offer price for the Futures contract.

ask_size: The number contracts available at the best offer price.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

5.2. fut_trade_received() – Callback Function

void fut_trade_received(char symbol[], long trade_price, long trade_size)

This function is called by the SDK Library to inform the customer's software that a Futures trade message has been received

The following parameters are passed when this function is called.

symbol: The symbol representing the Futures contract traded.

trade_price: The price the trade occurred at.

trade_size: The number of contracts traded.

5.3. index_data_update() – Callback Function

void index_data_update(char symbol[], double value, double net_change)

This function is called by the SDK Library to inform the customer's software that the value of an Index has changed.

NOTE: the parameters **value** and **net_change** are of type double.

The following parameters are passed when this function is called.

symbol: The symbol that represents the Index.

value: The value of the Index (double).

net_change: Net change from the previous days close (double).

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

6. Order Processing Functions

The Order processing functions consist of an order creation call by the customer's software to initiate an order and several order state update callback functions provided by the SDK library. Each order created by the customer's software will cause one or more state changes that will induce callbacks by the SDK library. The customer software should update its own internal states when making calls into or receive calls to the SDK library order functions. The SDK library will maintain full state of all orders, positions, and P&L for each account being traded.

The Black Box Trading System must maintain accurate state information when the Black Box Trading System is being shutdown and restarted for whatever reason. The SDK library builds a recovery log file that is read upon startup. When first started, the file does not exist. However, if the black box needs to be reset (brought down) then subsequent restarts will read the recovery file and build state. As the recovery file is read, each order state update and system update will cause the callback functions in the customer's software to be called. This is the opportunity for the customer software to rebuild its internal state. Customers should familiarize themselves with the Lightspeed Gateway specification from which the SDK order processing functions are based.

6.1. launch_new_order() – Function call

```
long launch_new_order(char return_order_token[],
                      char symbol[],
                      char b_s_ind,
                      long shares,
                      long price,
                      char odr_type,
                      long tif,
                      char display,
                      long display_shares,
                      long discretion_offset,
                      char use_dpls,
                      char sdot_conversion,
                      char routing_strategy,
                      char smart_algorithm,
                      char rash_outbound,
                      double account)
```

This function is called by the customer's software when a new order is to be sent to the Gateway. The SDK library will create and send this order to the Gateway. This function returns a day unique 5 byte character token (char return_order_token[]) that will be used on all subsequent order transactions relating to this order. This function also returns an error code that will tell the caller if the new order was actually sent or if an error occurred. **The customer software needs to keep this token as the identifier of this order. It will be passed in all order functions.**

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

The following parameters are passed when this function is called.

return_order_token: returned to the caller, contains the 5 byte order token as the identifier of this order. The customer's software must allocate 5 bytes for this parameter. This is NOT a string (i.e. not terminated with a 0).

symbol: specifies the security for the new order.

b_s_ind: specifies the side of the new order. Buy, Sell, Short

Value	Description
'B'	Buy order
'S'	Sell order
'T'	Short order

shares: specifies the number of shares required for this order. valid values: 1-999999.

price: The order price. Valid values: 0001 – 99999999. Refer to the "Price Representation"

odr_type: Specifies the type of order the customer's software wishes to use. Valid values:

Value	Description
'I'	INET order
'A'	ARCA order
'T'	BATS order
'Y'	NYSE order, SuperDot order, Directplus order
'H'	EDGE A order
'G'	EDGE X order
'R'	RASH order
'E'	AMEX order
'O'	BOST order

tif: Specifies the time the order should live in seconds. Valid values: 0 – 99999. Special meaning time in force values are contained in the following table. Please refer to the Lightspeed Gateway protocol specification for the specific handling of these values for different order types.

Value	Description
0	Immediate or Cancel

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

99994	Opening Cross
99997	Market on Close
99998	Times out at market close of primary exchange
99999	Good until exchange trading day ends

display: Specifies if the order should be hidden or displayed. Please refer to the Lightspeed Gateway protocol specification for the specific handling of these values for different order types. Valid values:

Value	Description
‘Y’	Anonymous: Price to Display
‘N’	Do Not Display the order
‘A’	Attributable: Price to Display
‘C’	Anonymous: Price to Comply
‘I’	Imbalance only
‘P’	Post Only

display_shares: Specifies the number of shares to display when using some form of iceberg order. 100 or greater.

discretion_offset: Specifies the disrectionary offset from the order price.

use_dpls: Specifies if this order should go to DirectPlus ‘Y’ or SuperDot ‘N’. This parameter is ignored on OTC orders.

sdot_conversion: Specifies that if **use_dpls** is set to ‘Y’ and the order is not DirectPlus eligible, ‘Y’ causes a convertion to SuperDot or ‘N’ causes the order to be rejected. If **use_dpls** is set to ‘N’, **sdot_conversion** is ignored. This parameter is ignored on OTC orders.

routing_strategy: Specifies the routing strategy to be added to the order being sent. Not all orders have a routing strategy and therefore this parameter will be ignored for those orders that do not. Please refer to the Lightspeed Gateway protocol specification for the specific handling of these values for different order types. Most of these strategy values are for BATS, but a few are for EDGX.

Value	Description
‘A’	BATS order routable to Arca
‘B’	BATS only
‘C’	BATS order routable to NSX Blade only

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

‘D’	BATS order routable to EDGA
‘E’	BATS order routable to ISE
‘G’	BATS order routable to EDGX
‘K’	BATS order routable to BOST
‘L’	BATS order routable to LavaFlow
‘M’	BATS order routable to Chicago Stock Exchange
‘N’	BATS order routable to NASDAQ
‘P’	BATS order Post Only
‘R’	BATS order routable to ALL
‘T’	BATS order routable to TRAC
‘U’	BATS order routable to AMEX
‘V’	BATS order routable to DATA
‘W’	BATS order routable to CBOE
‘X’	BATS order routable to Philly Stock Exchange
‘Y’	BATS order routable to NYSE
‘Z’	BATS order routable to NSX first then sweep BATS
‘1’	EDGX order, EDGX only
‘2’	EDGX order, external to all (ROUT)
‘3’	EDGX order, eligble to be routed to display liquidity but not IOI destinations (ROUX)
‘4’	EDGX order, will be routed to IOI destinations but not to displayed liquidity. (ROUZ)
‘5’	EDGX order, if listed order is marketable, eligible to be routed to DOT. (RDOT)

smart_algorithm: Specifies the RASH smart algorithm. For a complete description of each of the algorithm type please refer to the nasdaq trader website for Routing Strategies and Order Types Guide.
http://www.nasdaqtrader.com/content/ProductsServices/Trading/Workstation/rash_strategy.pdf

Value	Description
‘A’	SCAN
‘B’	DOTA
‘C’	STGY

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

‘D’	DOTM
‘E’	TFTY
‘F’	QTFY
‘G’	SKIP
‘H’	SAVE
‘I’	QSAV
‘J’	DOTI
‘K’	DOTZ

rash_outbound: Specifies the outbound flag.

Value	Description
‘Y’	Outbound flag is set
‘N’	Outbound flag is not set

account: This field is used to specify the account when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

Returns:

Value	Description
0	NO ERROR
2	ERROR BAD SYMBOL
3	ERROR PRICE TOO HIGH
4	ERROR GATEWAY CONNECTION DOWN
5	ERROR TOO MANY OPEN ORDERS ON THIS SYMBOL
6	ERROR CANNOT CREATE THIS ORDER
7	ERROR BAD ORDER TYPE
13	ERROR BAD ACCOUNT

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

6.2. sdk_restart_order() - Callback Function

```
void sdk_generated_order(char order_token[],
                         char symbol[],
                         char b_s_ind,
                         long shares,
                         long price,
                         char odr_type,
                         long tif,
                         char display,
                         long display_shares,
                         long discretion_offset,
                         char use_dpls,
                         char sdot_conversion,
                         char routing_strategy,
                         char smart_algorithm,
                         char rash_outbound,
                         double account)
```

This function is called by the SDK library when a new order is generated by the SDK library during a restart playback. As the system replays the day's events, it will come across new orders and use this routine to alert the customer's software that a new order has been generated. During playback, the customer software can expect to receive order updates as they occurred during the live session. This will allow the customer's software to build order state, P&L, and positions.

The following parameters are passed when this function is called.

order_token: Specifies the day unique 5 byte token generated by the SDK library when the new order was created.

symbol: specifies the security for the new order

b_s_ind: specifies the side of the new order. Buy, Sell, Short

Value	Description
'B'	Buy order
'S'	Sell order
'T'	Short order

shares: specifies the number of shares for this order. valid values: 1-999999.

price: The price of this order. Valid values: 0001 – 99999999.

odr_type: Specifies the type of order. Valid values:

Value	Description

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

'I'	INET order
'A'	ARCA order
'T'	BATS order
'Y'	NYSE order, SuperDot order, DirectPlus order
'H'	EDGE A order
'G'	EDGE X order
'R'	RASH order
'E'	AMEX order
'N'	NITE order
'O'	BOST order

tif: Specifies the time the order should live in seconds. Valid values: 0 – 99999.

display: Specifies the order as hidden or displayed. Valid values:

Value	Description
'Y'	Display the order
'N'	Do Not Display the order

display_shares: Specifies the number of shares to display when using some form of iceberg order. 100 or greater.

discretion_offset: Specifies the discretionary offset from the order price.

use_dpls: Specifies if this order should go to DirectPlus 'Y' or SuperDot 'N'. This parameter is ignored on OTC orders.

sdot_conversion: Specifies that if **use_dpls** is set to 'Y' and the order is not DirectPlus eligible, 'Y' causes a convert to SuperDot or 'N' causes the order to be rejected. If **use_dpls** is set to 'N', **sdot_conversion** is ignored. This parameter is ignored on OTC orders.

routing_strategy: Specifies the order routing set up in the original order. See detailed descriptions in above in launch_new_order() under routing_strategy.

smart_algorithm: Specifies the smart algorithm passed into the original order. See detailed descriptions in above in launch_new_order() under routing_strategy.

rash_outbound: Specifies the rash outbound flag from the original order.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

account: This field will specify the account originally supplied when the Gateway is set to multi-account mode. This field is set to 0 if the Gateway is set to single account mode.

6.3. console_order_request() – Callback Function

This callback function is called by the SDK library when the console operator sends an order to the customer's software. It contains the same parameters that the customer's software needs to send an order. The customer's software will need to use the parameter data from this callback and generate a **launch_new_order()** call to inititate the order. Please refer to the **launch_new_order()** documentation for specific descriptions of the parameters for this function.

```
void console_order_request(char symbol[],
                           char b_s_ind,
                           long shares,
                           long price,
                           char odr_type,
                           long tif,
                           char display,
                           long display_shares,
                           long discretion_offset,
                           char use_dpls,
                           char sdot_conversion,
                           char routing_strategy,
                           char smart_algorithm,
                           char rash_outbound,
                           double account)
```

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

6.4. console_cancel_all_orders() – Callback Function

void console_cancel_all_orders(double account)

This function is called by the SDK library when the console operator has requested that all live orders for the specified account be canceled. If the Trading Mode is set to Single Account Mode, then the account parameter will be se to 0 and the customer software should cancel all live orders. If the Trading Mode is set to Multiple Account Mode, then the customer's software should examine the account parameter and should cancel all orders associated with the specified account. If the Trading Mode is set to Multiple Account Mode and the account parameter specified the Aggregate Account Number, then the customer's software should cancel all orders.

The following parameter is passed when this function is called.

account: Specifies the account number for which all orders are to be canceled. If the Gateway is in single account mode, this value will be set to 0. If the Gateway is in multi-account mode, and the console operator wants all orders cancelled this account value will be set to 9999999999.

6.5. console_cancel_symbol_all_orders() – Callback Function

void console_cancel_symbol_all_orders(char symbol[], double account)

This function is called by the SDK library when the console operator has requested that all live orders for the security and the specified account be canceled. If the Trading Mode is set to Single Account Mode, then the account parameter will be set to 0 and the customer's software should cancel all live orders for the specified security. If the Trading Mode is set to Multiple Account Mode, then the customer's software should examine the account parameter and should cancel all orders for the specified symbol that are associated with the specified account. If the Trading Mode is set to Multiple Account Mode and the account parameter specified the Aggregate Account Number, then the customer's software should cancel all orders for the specified security..

The following parameters are passed when this function is called.

symbol: Specifies the security that orders should be canceled. The symbol and account parameters are used to determine which orders are to be canceled.

account: Specifies the account number for which orders are to be canceled. The symbol and account parameters are used to determine which orders are to be canceled.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

6.6. console_cancel_one_order() – Callback Function

void console_cancel_one_order(char order_token[], char symbol[], double account)

This function is called by the SDK library when the console operator has requested that one order be canceled.

The following parameters are passed when this function is called. If the Gateway is configured for single account mode the account value is 0.

order_token: Specifies the day unique 5 byte token generated by the SDK library when the new order was created.

symbol: Specifies the security.

account: Specifies the account for this order.

6.7. order_accept() – Callback Function

void order_accept(char order_token[], char symbol[], double account)

This function is called by the SDK library when a previous order sent by the customer's software or the SDK is received from the Gateway. The 5 byte order token is used by the customer's software to identify the order.

The following parameters are passed when this function is called.

order_token: Specifies the day unique 5 byte token generated by the SDK library when the new order was created.

symbol: Specifies the security.

account: This field is used to specify the account. When the Gateway is set to single account mode, this value is 0.

6.8. launch_order_cancel() – Function call

long launch_order_cancel(char order_token[], char symbol[], double **account)**

This function call will cause an attempt to cancel an existing order.

The following parameter is passed when this function is called.

order_token: Specifies the day unique 5 byte order token received from one of the two new order routines.

symbol: Specifies the security.

account: This field is used to specify the account. When the Gateway is set to single account mode, this value is 0.

Returns:

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

Value	Description
0	NO ERROR
2	ERROR, BAD SYMBOL
4	ERROR, GATEWAY CONNECTION DOWN
8	ERROR, ORDER NOT FOUND
9	ERROR, ORDER NO LONGER LIVE
10	ERROR, CANCEL ALREADY PENDING
11	ERROR, ORDER NOT YET ACCEPTED
12	ERROR, MAXIMUM CANCEL ATTEMPTS

6.9. order_cancelled() – Callback Function

void order_cancelled(char order_token[], char symbol[], double account)

This function is called by the SDK library when a previous order has been cancelled by the Gateway. The 5 byte order token is used by the customer's software to identify the order.

The following parameters are passed when this function is called.

order_token: Specifies the 5 byte day unique token generated by the SDK library when the new order was created.

symbol: specifies the security.

account: This field is used to specify the account. When the Gateway is set to single account mode, the value is 0.

6.10. order_rejected() - Callback Function

void order_rejected(char order_token[], char symbol[], char reason_code, double account)

This function is called by the SDK library when a previously sent order has been rejected by the Gateway. The 5 byte order token is used by the customer's software to identify the order.

The following parameters are passed when this function is called.

order_token: Specifies the day unique 5 byte token generated by the SDK library when the new order was created.

symbol: specifies the security.

reason_code: Specifies the reason the order was rejected. See reason codes below.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

Value	Description
‘A’	Odd lot to venue.
‘B’	Dotx 30 second rule.
‘C’	Destination for order is closed or currently down
‘D’	Bid tick violation
‘E’	Max order size rule
‘F’	Max position size rule
‘G’	Rule update in progress
‘H’	Stock is halted on the primary exchange
‘I’	Price not available
‘J’	Short order with long position
‘K’	Sell order without long position
‘L’	Potential overselling
‘M’	Sell shares more than long
‘N’	Nonshortable
‘O’	Other error
‘P’	Insufficient day-trading buying power
‘Q’	One way BP rule
‘R’	Protection price
‘S’	Invalid symbol
‘T’	Test mode: can only send order on test stocks
‘U’	Marked PNL cutoff rule
‘V’	Overselling
‘W’	Not well formed, one or more field
‘X’	Supershort in multiple sells account
‘Y’	Invalid account number
‘Z’	Max order size
‘0’	Exceed Throttle Limit
‘1’	Rash Algorithm Block
‘2’	System Error

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

‘9’	System Error

account: This field is used to specify the account. When the Gateway is set to single account mode, the value is 0.

6.11. cancel_rejected() - Callback Function

void cancel_rejected(char order_token[], char symbol[], char reason_code, double account)

This function is called by the SDK library when a previous cancel request sent by the customer’s software has been rejected by the Gateway. The 5 byte order token is used by the customer’s software to identify the order.

order_token: Specifies the day unique 5 byte token generated by the SDK library when the new order was created.

symbol: specifies the security.

reason_code: Specifies the reason the cancel was rejected.

Value	Description
‘L’	Token malformed
‘N’	Token unknown
‘C’	Destination for order is closed or currently down
‘O’	Other error
‘0’-‘9’	System error

account: This field is used to specify the account. When the Gateway is set to single account mode, this value is 0.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

6.12. **order_execution() - Callback Function**

void order_execution(char order_token[], char symbol[], char b_s_ind, long shares, long price, long ref_num, char contra[], char liquidity_flag, char venue_code, double account)

This function is called by the SDK library when an execution occurs on an order. The 5 byte order token is used by the customer's software to identify the order. This will cause a position state update.

The following parameters are passed when this function is called.

order_token: Specifies the day unique 5 byte token generated by the SDK library when the new order was created.

symbol: specifies the security.

b_s_ind: Specifies if this is a Buy = 'B', Sell = 'S', or Short = 'T'.

shares: Specifies the number of shares in this execution.

price: Specifies the price of this execution.

ref_num: Specifies the day unique integer reference number of this execution.

contra: Specifies the identifier of the execution's contra ECN or market-maker.

liquidity_flag: Specifies 'A' for added liquidity or 'R' for removed liquidity.

venue_code: Specifies the venue where the trade executed.

account: This field is used to specify the account. When the Gateway is set to single account mode, this value is 0.

6.13. **journal_position() - Callback Function**

void journal_position(char symbol[], char b_s_ind, long shares, long price, double account)

This function is called by the SDK library when a journal entry position (from the console) has been received. This will cause a position state update.

The following parameter is passed when this function is called.

symbol: specifies the security.

b_s_ind: Specifies if this is a Buy = 'B', Sell = 'S', or Short = 'T'.

shares: Specifies the number of shares in this position change.

price: Specifies the price of this position change.

account: This field is used to specify the account. When the Gateway is set to single account mode, this value is 0.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

6.14. doneaway_trade() - Callback Function

void doneaway_trade(char symbol[], char b_s_ind, long shares, long price, long ref_num, double account)

This function is called by the SDK library when a doneaway trade occurs for this account. This doneaway will act as an order execution. However, it will not be associated with a previously entered order. Doneaway trades are commonly used for fixing clearly erroneous trades or fixing problems with incorrect positioning. Doneaway trades will effect buying power and position.

The following parameters are passed when this function is called.

symbol: specifies the security.

b_s_ind: 'B'= bought, 'S'= sold long, 'T'=sold short

shares: The number of shares in the doneaway

price: The price of the doneaway.

ref_num: The numeric reference number of the doneaway.

account: This field is used to specify the account. When the gateway is set to single account mode, this value is 0.

6.15. sod_position() - Callback Function

void sod_position(char l_s_ind, long shares, char symbol[], long price, double account)

This function is called by the SDK library with start of day positions. One call will be made for each position.

The following parameters are passed when this function is called.

l_s_ind: 'L'= Long position, 'S'= short position

shares: The number of shares for this Start Of Day position.

symbol: The stock symbol of the position.

price: The price of the SOD position.

account: This field is used to specify the account. When the Gateway is set to single account mode, this value is 0.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

6.16. sod_buying_power() - Callback Function

void sod_buying_power(char scope, double buying_power, double account)

This function is called by the SDK library with the start of day buying power.

The following parameters are passed when this function is called.

scope: ‘S’ = Stock, ‘F’ = Futures

buying_power: The buying power value. Note, this number is represented as a double. If this parameter contains 500.5, then the buying power would be five hundred dollars and fifty cents.

account: This field is used to specify the account. When the Gateway is set to single account mode, this value is 0.

6.17. order_type_status() - Callback Function

void order_type_status(char order_type, char status)

This function is called by the SDK library when the status of an order type has changed. A common way to change an order type status is for the operator to use the console to change the status. For example, if the operator decides they no longer want to route orders to BATS, they can use the console to disable the BATS order type. This will result in this callback function being called with order_type equal to ‘T’ (BATS) and the status equal to ‘D’ (Disabled). Note, the customer’s software is responsible to actually stop sending orders with an order type of BATS (‘T’). The customer’s software cannot assume that the SDK Library will reject order sent with a disabled order type.

The following parameters are passed when this function is called.

order_type: Specifies the order type whose status is being updated. Valid values:

Value	Description
‘I’	INET
‘A’	ARCA
‘T’	BATS
‘Y’	NYSE, SuperDot, DirectPlus
‘H’	EDGE A
‘G’	EDGE X
‘R’	RASH
‘E’	AMEX

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

‘O’	BOST

status: Indicates the status of the order type. Valid values:

Value	Description
‘D’	Disabled
‘U’	Enabled or UP

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

7. Timer Functions

The SDK Library provides functions to allow the customer's software to set a timer, cancel a timer, and be notified when the specified time has elapsed.

When the customer's software sets a timer, they are instructing the SDK Library to notify them at a specified time in the future. For example, the customer's software can instruct the SDK library to notify them 3 seconds from the time the request is made.

When the customer's software sets a timer, it specifies the time to be notified in microseconds. For example, 3 seconds is 3000000 microseconds.

The largest timer that can be set is 1800000000 microseconds (1800 seconds, or ½ hour).

The customer's software will call the set_timer() function to set a timer. The set_timer() function will return an integer that is a unique identifier for the timer. This identifier is chosen by the SDK Library and is used to identify the timer if it needs to be canceled by calling cancel_timer();

The set_timer() function has 5 parameters. The first parameter specifies the time to be notified in microseconds. The second parameter is a customer chosen identifier that can be used by the customer's software to determine the appropriate action to take when the timer has elapsed. The customer's software must choose an identifier that is greater than or equal to 100,000 (see the example code below). The next three parameters are 4 byte values (represented as integers) that the customer software can populate however they like. The SDK Library will simply return the value of these three parameters to the customer software when it is notified the timer has elapsed.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

Example Code

```
#define  TIMER_CANCL_ORDERS             100001
#define TIMER_CHECK_SOMETHING           100002



// code example to set timer
// set the timers
// set a timer to cancel all orders in 5 seconds
cancel_order_timer_id = set_timer(5000000, TIMER_CANCL_ORDERS, 0, 0, 0);

// set a timer to check something in 10 milliseconds
check_something_timer_id = set_timer(10000, TIMER_CHECK_SOMETHING  , 1, 2, 3);



// code example to cancel a timer before the specified time has elapsed.
cancel_timer(check_something_timer_id);



// code example to process timer when notified by the SDK Library
// the code to process timer notifications is done in the timer_event() callback function

void
timer_event(id, value1, value2, value3)
{
        switch(id)
        {
        case TIMER_CANCL_ORDERS:
                cancel_all_live_orders();
                break;

        case TIMER_CHECK_SOMETHING:
                check_something(value1, value2, value3);
                break;
        }
}
```

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

7.1. set_timer() – Function Call

long set_timer(long call_back_time, long id, long value1, long value2, long value3)

This function can be called by the customer’s software to set a timer

The following parameters are passed when this function is called.

call_back_time: Specifies the time in microseconds to be notified.

id: A customer chosen identifier that can be used by the customer’s software to determine the appropriate action to take when the specified timer has elapsed. This parameter must be greater than or equal to 100,000.

value1: Any value chosen by the caller. This value will be passed back to the caller in the timer_event() function.

value2: Any value chosen by the caller. This value will be passed back to the caller in the timer_event() function.

value3: Any value chosen by the caller. This value will be passed back to the caller in the timer_event() function.

Returns

An identifier chosen by the SDK Library. Can be used by the caller to cancel a timer.

If the timer cannot be set, then –1 is returned.

7.2. cancel_timer() – Function Call

long cancel_timer(long timer_id);

This function can be called by the customer’s software to cancel a previously set timer.

The following parameter is passed when this function is called.

timer_id: Used to identify the timer to be canceled. This must be the same value returned by set_timer().

Returns

0 is returned if the timer is cancelled successfully.

-1 is returned if the timer could not be canceled. For example, the timer_id passed as a parameter was invalid.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

7.3. timer_event() – Callback Function

void timer_event(long id, long value1, long value2, long value3)

This function is called by the SDK Library to inform the customer's software that a timer has elapsed.

The following parameters are passed when this function is called.

id: The identifier the customer chose when setting the timer with the set_timer() function. This value can be used by the customer's software to determine the appropriate action to take (see the example code above).

value1: Specified by the customer's software when calling the set_timer() function, and simply returned here.

value2: Specified by the customer's software when calling the set_timer() function, and simply returned here.

value3: Specified by the customer's software when calling the set_timer() function, and simply returned here.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

8. Alarm Clock Timer Functions

An alarm clock timer is the term used to describe a timer function in which the customer's software specifies a system time to be notified. For example, the customer can instruct the SDK to notify them at 342,000,000 milliseconds past midnight. Note, this differs from the timer functions described in the section above where the customer specifies the number of microseconds from the current time to be notified.

The SDK Library provides functions to allow the customer's software to set an Alarm Clock timer, cancel an Alarm Clock timer, and be notified at the specified time.

When the customer's software sets an alarm clock timer, they are instructing the SDK Library to notify them at a specified time in the future expressed in milliseconds past midnight. For example, the customer's software can instruct the SDK library to notify them 10 milliseconds before the 9:30am market open by specifying the time 34,199,990.

The customer's software will call the set_alarm_clock_timer() function to set a timer. The set_alarm_clock_timer() function will return an integer that is a unique identifier for the timer. This identifier is chosen by the SDK Library and used to identify the timer to be canceled when calling cancel_alarm_clock_timer();

The set_ alarm_clock_timer() function has 5 parameters. The first parameter specifies the time to be notified in milliseconds past midnight. The second parameter is a customer chosen identifier that can be used by the customer's software to determine the appropriate action to take when the specified time has elapsed. The customer's software must choose an identifier that is greater than or equal to 100,000. The next three parameters are 4 byte values (represented as integers) the customer's software can specify however they like. The SDK Library will simply return the value of these three parameters to the customer's software when it is notified the specified time has been reached.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

8.1. set_alarm_clock_timer() – Function Call

long set_alarm_clock_timer(long time, long id, long value1, long value2, long value3)

This function can be called by the customer's software to set an alarm clock timer.

The following parameters are passed when this function is called.

time: Specifies the time in milliseconds past midnight to be notified.

id: A customer chosen identifier that can be used by the customer software to determine the appropriate action to take when the specified time has been reached. This parameter must be greater than or equal to 100,000.

value1: Any value chosen by the caller. This value will be passed back to the caller in the alarm_clock_timer_event() function.

value2: Any value chosen by the caller. This value will be passed back to the caller in the alarm_clock_timer_event() function.

value3: Any value chosen by the caller. This value will be passed back to the caller in the alarm_clock_timer_event() function.

Returns

An identifier chosen by the SDK Library. Can be used by the caller to cancel a timer.

If the timer cannot be set, then –1 is returned.

8.2. cancel_alarm_clock_timer() – Function Call

long cancel_alarm_clock_timer(long timer_id);

This function can be called by the customer's software to cancel a previously set alarm clock timer.

The following parameter is passed when this function is called.

timer_id: Used to identify the timer to be canceled. This must be the same value returned by set_alarm_clock_timer().

Returns

0 is returned if the timer is cancelled successfully.

-1 is returned if the timer could not be canceled. For example, the timer_id passed as a parameter was invalid.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

8.3. alarm_clock_timer_event() – Callback Function

void alarm_clock_timer_event(long id, long value1, long value2, long value3)

This function is called by the SDK Library to inform the customer's software that the time specified by an alarm clock timer has been reached.

The following parameters are passed when this function is called.

id: The identifier the customer chose when setting the timer with the set_alarm_clock_timer() function. This value can be used by the customer's software to determine the appropriate action to take.

value1: Specified by the customer's software when calling the set_alarm_clock_timer() function, and simply returned here.

value2: Specified by the customer's software when calling the set_ alarm_clock_timer() function, and simply returned here.

value3: Specified by the customer's software when calling the set_ alarm_clock_timer() function, and simply returned here.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

9. Short Availability Functions

9.1. short_avail_event() – Callback Function

void short_avail_event(char symbol[], char flag)

This function is called by the SDK Library to inform the customer's software that the short availability flag has been updated. The short availability flag indicates whether a security is eligible for short sale and any conditions associated with selling short.

The following parameters are passed when this function is called.

symbol: The security whose short availability flag has been updated.

flag: Short availability flag. This field will contain the following values:

Value	Description
'Y'	Available for short sale.
'H'	Hard to borrow – available for short sale when located is available.
'X'	Not available for short sale.
'T'	Threshold security.
'N'	Unknown – call for locate.

9.2. get_short_avail() – Function Call

char get_short_avail(char symbol[])

This function can be called by the customer's software to obtain the short availability flag.

The following parameter is passed when this function is called.

symbol: The security the customer wants to obtain the short availability flag for.

Returns

Returns the short availability flag (see the table above in the short_avail_event() description).

A 'U' is returned if the security is not recognized by the SDK Library.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

9.3. short_avail_status_event() – Callback Function

void short_avail_status_event(long event_type)

This function is called by the SDK Library to inform the customer's software of "events" related to the Short Availability Server. For example, a connection to the short availability server has been made or dropped. The following table describes the event types.

Event Type	Description
CONNECTION_MADE	A connection has been established with the Short Availability server.
CONNECTION_DROPPED	The connection to the Short Availability server has been dropped

The following parameter is passed when this function is called.

event_type: Specifies the type of event. Valid values include:

Value	Description
1	CONNECTION_MADE.
2	CONNECTION_DROPPED

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

10. Console Interface Functions

Customer's can develop software to send message to the console and receive messages from the console. Messages sent to the console typically contain information to be displayed. Messages sent from the console typically instruct the customer's Black Box software to take some action, such as stop trading or change a trading parameter.

Before discussing the functions it's important to understand the message format used by the SDK Library to exchange messages with the consoles. The following illustrates the message format.

Field Name	Field Length	Field Type	Description
Length	4	Hex Numeric	This field contains the length of the message. The length of the message does not include this field. This field is representated as a four character hexidicmal number, padded with zeros on the left. For example, a packet length of 25 bytes is 0019.
Message Type	4	Numeric 1 - 5998	This field identifies the type of message and is represented as a decimal number. The customer's software can use message types in the range 1 to 5998. Note, message type 5999 is used to send commands entered by the command line interface to the customer's Black Box software. See the command line discussion below.
Customer Information	0 – 65,530	Cutomer defined	This is the message body and contains any inforamtion the customer wants to exchange between the customer's Black Box software and the console. The customer is responsible for defining the content of this field.
ETX	1	0x03	End of message

The send_message_to_console() function take a message created by the customer's software and sends it to all consoles connected to the Block Box Trading System. When sending message to the consoles, it is important to use the message structure defined above.

The console_msg_received() function is a callback function and is called by the SDK Library when a console message have been received. If the message received from the console is requesting the customer's software to respond with a message, then the customer's software will create the response message in console_msg_received() callback function. A pointer to the response message and the length of the response message are returned from console_msg_received(). Upon return form console_msg_received(), the SDK Library will send the response message to the same console that the request message was received from. Inside the console_msg_received() function, the customer's software must allocated (using malloc) memory for the message. Once the SDK Library has sent the response message to the console it will free the memory the response message occupies. See the console_msg_received() description below for more details.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

10.1. Console Command Line

There are two methods the customer can use to send messages from the console to the customer's Black Box software. The console source code is provided with the SDK. One method is for the customer to develop software to send messages to the customer's Black Box software. Using this method the customer can create dialog boxes, buttons, text boxes and other Windows input methods to allow an operator to input information. This information can then be put in a message and sent to the customer's Black Box software.

The second method is to use the console command line interface. The console application that is provided with the SDK contains a command line interface. The command line interface allows the operator to type in "commands". The commands will be put into the message format described above and sent to customer's Black Box software. The message type will be 5999. The customer's Black Box softare can assume that messages received with message type 5999 contain information entered using the command line interface. For example, if the operator types "stop trading" in the command line, then the following message will be send to the Black Box.

Field Name	Field Length	Field Type	Description
Length	4	0011	Length of message in hexidecimal notation. Does not include the size of this field. The message is 17 bytes which is 11 hex.
Message Type	4	5999	Message type 5999 is use to send commands entered via the command line interface to the Black Box.
Customer Information	12	stop trading	Command entered by operator.
ETX	1	0x03	End of message

The following is the string the console would send to the customer's Black Box software: 0011599stop trading^ (^ represents ETX).

Note, this is just an example. The command "stop trading" is not a functioning command. The customer's Black Box software is responsible for parsing the body of the message and taking action based on the contents of the message.

The command line interface allows the customer to send information to the customer's Black Box software without having to develop any console code. Note, the customer could choose to develop console code to create messages and send them to the customer's Black Box software using message types that range from 1 to 5998. One reason the customer may want to develop console software to send messages to the customer's Black Box software, is to allow them to create messages that are easier to parse. Fix field messages can be easier to parse. Again, this is not required. If the customer does not want to develop any console code, they can define a command syntax and use the command line interface.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

10.2. send_msg_to_console() – Function Call

long send_msg_to_console(char msg[], long len)

This function can be called by the customer's software to send a message to all consoles. The customer's software must build messages in the format described above. This function will send the message to all consoles that are currently connected to the Black Box Trading System.

The following parameters are passed when this function is called.

msg: A pointer to the message to be sent to all consoles.

len: The length of the entire message. This value specified in this parameter must include all fields (message length, message type, message body, and ETX).

Returns

0 is returned is the message was sent to all consoles

-1 is returned if no consoles are currently connected to the Black Box Trading System.

10.3. console_msg_received() – Callback Function

long console_msg_received(char msg[], char **rtn_msg_ptr)

This function is called by the SDK Library to inform the customer's software that a message from a console has been received. The message type will be in the range 1 to 5999. Message type 5999 indicates the message contains information entered via the console command line interface. Message types 1 through 5998 are messages created by console software that was developed by the customer.

The message received will be in the format described above and the pointer passed to this function (msg) will point to the Message Length field.

When the customer's software returns from this function call, it can no longer access the message pointed to by msg. In other words, it cannot save the pointer to the message and later attempt to access the message after it returns from this call. If the customer's software needs to access the message after returning, it must make a copy of the message before returning.

If the message (msg) received from the console is requesting the customer's software to respond with a message, then the customer's software must create the response message in this function. The response message must be in the console message format described above. The customer's software must allocate space for the message. Note, the maximum message size is 64K bytes. This includes the length field, message type, message body (customer's information) and the ETX. A pointer to a pointer (rtn_msg_ptr) is passed to this function, and the customer's software must return the pointer to the allocated memory that contains the message. For example, the following code will allocate memory and return the pointer.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

```
*rtn_msg_ptr = malloc(sizeof(customer_response_msg));
```

The customer’s software must not de-allocate (free) the memory. This will be done by the SDK Library.

The customer’s software must return the total length of the response message. The message length returned must include the length field, message type, message body (customer’s information) and the ETX. If the customer’s software does not want a response message sent to the console, then 0 (zero) must be returned.

When this function returns control to the SDK Library, the SDK Library will check the message length returned. If the message length is greater than 0 (zero), then the response message will be sent to the same console that the request message was received from. Once the SDK Library has sent the response message to the console it will free the memory the response message occupied.

The following parameters are passed when this function is called.

msg: Pointer to the message received from the console.

rtn_msg_ptr: Pointer to a pointer. The customer’s software can return a pointer to a response message that is to be sent to the console.

Returns

The length of the response message to be sent to the console.

0 - If no response message is to be sent to the console, then 0 (zero) is returned.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

11. Utility Functions

11.1. log_data – Function Call

void log_data(FILE *fp, char *fmt, ...)

This function can be called by the customer's software to have information displayed on the console scroll window, and optionally written to a file.

The first parameter specifies a file descriptor that indicates the file the information will be written to. If the customer's software does not want the information written to a file, then a NULL (0) should be passed in this parameter. The log file used by the SDK Library can be written to by using the function call get_log_file_handle() to obtain the file descriptor used by the SDK Library. Below are two example of logging information. In the first example, the information not written to a file and only displayed on the console scroll window. In the second example, the information is written to the file and displayed on the console scroll window.

```
Log_data(0, "Hello There. I am %d years old\n", age);          // console display only
Log_data(fp, "Hello There. I am %d years old \n", age);    // write to file and console display
```

The second parameter is the same type of "control_string" that is used with printf. The control_string consists of two types of items. The first type is composed of characters to that will be printed on the screen or written to a file. The second type contains format specifiers that define the way the subsequent arguments are displayed. A format specifier begins with a percent sign and is followed by the format code. For example, %d displays signed integers.

11.2. get_log_file_handle() – Function Call

FILE *get_log_file_handle(void)

The SDK Library maintains a log file for logging important system information. The customer's software may also want to put information into a log file. The customer's software can create its own log file, or the customer's software can put information in the SDK Library's log file. If the customer's software want to put information in the SDK Library's log file, then it can call is function to obtain the file descriptor for the SDK Library's log file.

Returns

The file descriptor for the SDK Library's log file.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

11.3. get_company_name() – Function Call

long get_company_name(char symbol[], char company_name[])

This function can be called by the customer's software to obtain the company name associated with a symbol.

The following parameters are passed when this function is called.

symbol: The security the customer want to obtain the company name for.

company_name: This parameter provided the storage space that the SDK Library will put the company name into. The customer's software must provide 200 bytes of storage space. The company name will not exceed 200 bytes.

Returns

0 is returned if successful

-1 is returned if the security is not recognized by the SDK Library.

11.4. aftol() – Function Call

long aftol(char buf[], long len)

This function can be called by the customer's software to convert a buffer containing an ascii **price** in decimal notation to the internal integer representation. Prices in the SDK Library are implemented as integers with four digits of precision to the right of the decimal point. For example, the price of 25.98 is represented as the integer value 259800.

If this function is passed a buffer containing the characters "25.98", it will convert it to the integer 259800.

The **len** parameter indicates the maximum number of character to process when performing the conversion. For example, if the buffer contains "123456789.12", and this function is called with len equal to 5, then the resulting integer value will be 123450000.

This function will stop processing the buffer when a character other than '0' through '9' or a period ('.') is encountered. For example, if this function is passed the null terminated string "12.34" and **len** is equal to 20, then the resulting integer is 123400. Note, the conversion will stop when the null after the '4' is encountered even though **len** was set to 20.

The following parameters are passed when this function is called.

buf: The buffer containing the ascii price in decimal notation to be converted to integer representation.

len: Maximum number of characters to process when performing the conversion

Returns

This functions returns the integer representation of the price.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

11.5. asctol() – Function Call

long asctol(char buf[], long len)

This function can be called by the customer's software to convert an ascii buffer containing a whole number to a long data type. The len parameter indicates the maximum number of character to process when performing the conversion. For example, if the buffer contains "123456789", and this function is called with len equal to 5, then the resulting integer value will be 12345.

The following parameters are passed when this function is called.

buf: The buffer containing the ascii whole number to be converted to integer representation (long).

len: Maximum number of characters to process when performing the conversion

Returns

This function returns the integer (long) representation of the whole number.

11.6. get_time_string() – Function Call

void get_time_string(char time[])

This function can be called by the customer's software to obtain a string that contains the system time with microsecond accuracy. The time is returned in the following format hh:mm:ss.nnnnnn. For example, 12:35:45.124534. The caller must provide a buffer large enough to accommodate the time string (16 bytes or more).

The time string is returned in the time parameter.

11.7. get_secs_past_midnight() – Function Call

long get_secs_past_midnight(void)

This function can be called by the customer's software to obtain the system time expressed in the number of seconds past midnight.

Returns

The system time in seconds past midnight.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

11.8. get_millisecs_past_midnight() – Function Call

long get_millisecs_past_midnight (void)

This function can be called by the customer's software to obtain the system time expressed in the number of milliseconds past midnight.

Returns

The system time in milliseconds past midnight.

11.9. get_symbol_length() – Function Call

long get_symbol_length(char symbol[])

This function can be called by the customer's software to obtain the number of characters in the security symbol.

The following parameters are passed when this function is called.

symbol: The security whose symbol length is to be returned.

Returns

The number of characters in the security symbol.

11.10. send_audible() – Function Call

void send_audible(int tone)

This function can be called by the customer's software to cause the console to make a sound. This is typically used to alert the operator of an event that need attention.

The following parameter is passed when this function is called.

tone: Specifies the type of sound to be made. Currently on one sound is available and this parameter should be set to 0.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

11.11. Hashing Function

The SDK Library provides a hashing function that maps a string of character to an array index. The following code sample illustrates the use of the hashing function.

```
#define TABLE_SIZE          (1<<8)
#define HASH_MASK           (TABLE_SIZE – 1)
#define MAX_SYMBOL_SIZE  6

typedef _sym_t {
    sturct _sym_t  *next_ptr;
    long           symbol[MAX_SYMBOL_SIZE ];
    long           other_stuff;
    long           more_stuff;
} sym_t;

sym_t    *hash_table[TABLE_SIZE[;
char     symbol[MAX_SYMBOL_SIZE   ];
long      index;
sym_t    *sym_ptr;

sym_ptr = (sym_t *)malloc(sizeof(sym_t));

memcpy(sym_ptr->symbol, “MSFT  “,MAX_SYMBOL_SIZE);
sym_ptr->other_sutff = 1;
sym_ptr->more_stuff = 2;
sym_ptr->next_ptr = NULL;

index = hashCrc32(symbol, MAX_SYMBOL_SIZE   ) & HASH_MASK;

if (hash_table[index] == NULL)
{
    hash_table[index] = sym_ptr;
}
else
{
    sym_ptr->next_ptr = hash_table[index];
    hash_table[index] = sym_ptr;
}
```

In the code sample above, a data structure is used to store information about a single security. A hash table is used to quickly locate the data structure associated with a security.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

Line 1 defines the size of the hash table. The size of the hash table must be a power of two. In the example above, the size of the hash table is 256 (1<<8).

Line 2 defines the hash mask. This value must be the size of the hash table minus 1. In the example above, the hash mask is 255 or 11111111 in binary. The hash mask is used to ensure that the index produced by the hash function does not exceed the size of the hash table array.

Line 3 defines the maximum size of a symbol, which is 6 in the code sample.

Lines 4 through 9 define the data structure that will be used to store information about a security.

Line 10 declares the memory to be used as the hash table. An array of 256 elements is declared, and each array element is a pointer to a data structure of type sym_t.

Lines 11 through 13 defines variables used in the code sample.

Line 14 calls malloc to allocate memory for one data structure that will contain information about a security.

Lines 15 through 18 fill in the security information data structure.

Line 18, the hashCrc32 hash function is called to map the symbol string "MSFT " to an index into the hash table array. The hashCrc32 function is passed two parameters, the pointer to a string containing a symbol (eg, MSFT) and the number of character to perform the hash function on. A bitwise AND operation of the results returned from hashCrc32 and the hash mask is performed. This will truncate the index and ensure that it does not exceed the size of the hash table array.

Lines 19 through 26 are used to store the pointer to the security data structure in the hash table. Each hash table array element is a pointer to a linked list of security data structures. Line 19 tests to see if the list is empty. If the list is empty, then the security data structure is simply put on the empty list (line 21). If the list is not empty, then the security data structure is inserted at the head of the list (lines 25 and 26).

11.12. hashCrc32() – Function Call

unsigned long hashCrc32(const void *voidPtr, long len)

This function can be called by the customer's software to perform a hash function on a string of character. The result of the hash function is returned as an unsigned long.

The following parameters are passed when this function is called.

voidPtr: Pointer to the string of characters that the hash function will be performed.

len: The number of characters that the hash function will be performed on.

Returns:

The results of the hash function is returned as an unsigned long.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

12. Reading Configuration Parameters from a File

The SDK Library contains tools to allow the customer's software to easily read configuration information from a file. The customer can create a configuration file and the name of the file can be any valid file name. The contents of the configuration file must use the format tag = value. The following is an example of the contents of a configuration file that contains two configuration parameters.

```
parameter1 = 12345
parameter2 = abcd
```

There can be any number of spaces between the tag and the equal sign, and any number of spaces between the equal sign and the value.

The function read_config() is called to read the contents of the configuration file. The read_config() function takes two parameters. The first parameter is the name of the configuration file. The second parameter is a pointer to a data structure that defines the tags to be read and where the values are to be stored. For example, read_config("config_file", config_list) will read the file config_file and extract the information as defined by config_list.

The following code sample will read the contents of a configuration file that contains the two parameters shown above (parameter1 and parameter2).

```
typedef struct {
   char  *tag;              //  contains the address of the Tag
   char  *buf_addr;         //  contains the address of the buffer to store the value into
   long   len;              //  length of the buffer
} config_def_t;

config_def_t  config_list[] = {
   {"parameter1",  &g_config_values.parameter1[0], PARM1_SZ},
   {"parameter2", &g_config_values.parameter2[0], PARM2_SZ},
   {NULL, NULL, 0}
};

//  data type used to store configuration items
typedef struct {
   char parameter1[PARM1_SZ +1];
   char parameter2[PARM2_SZ +1];
} config_value_t;

// declare variable to store configuration items into
config_value_t  g_config_values;

// call the SDK Library function to read the configuration file and store the values in g_config_values
read_config("config_file", config_list);
```

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

The typedef config_def_t is used to define one configuration parameter. It contains a pointer to a string that contains the tag, the address of a buffer to store the value, and the length of the buffer.

The variable config_list is an array of type config_def_t, and each element represents one configuration parameter. Each array element contains the tag (eg, "parameter1"), the address of a buffer to store the value (eg, &g_config_values.parameter1[0]), and the length of the buffer (eg, PARM1_SZ). The last element of the array must contain a NULL pointer for the tag string. The read_config() function will stop processing array elements when it encounters a NULL tag pointer.

The typedef config_value_t is used to define how configuration parameter values will be stored. For example, an array of size PARM1_SZ plus 1 has been defined to store the value of parameter1. The size of the buffer should be one larger than the length of the buffer specified in the config_list array. The configuration parameter values are stored as strings which require NULL termination. The extra byte ensures there is space for a NULL to terminate the string.

The statement (config_info_t g_config_values;) is used to declare memory to store the configuration parameter value strings. g_config_values is of type config_info_t.

The customer's software will call the function read_config() to read the contents of the configuration file. The read_config() function takes two parameters. The first parameter is the name of the configuration file (config_file). The second parameter is a pointer to a data structure that defines the tags to be read and where the values are to be stored (config_list).

When read_config() returns, the data structure g_config_values will have been populated with the configuration parameter value that were extracted form the configuration file. For example, g_config_values.parameter1 will contain the string "12345", and g_config_values.parameter2 will contain the string "abcd".

12.1. Adding a Configuration Parameter

Once the code has been written to read configuration parameters, adding a new parameter requires only two lines of code to be written. Below is the same code sample described above, but a new configuration parameter has been added. The new configuration parameter is called "new-parm". The two lines of code required to read the new parameter are highlighted in blue text in the example code below.

If the following record has been added to the configuration file, then once read_config() has been called g_config_values.new_parm will contain the string "xyz123".

```
new-parm = xyz123
```

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

```
typedef struct {
    char  *tag;             //  contains the address of the Tag
    char  *buf_addr;        //  contains the address of the buffer to store the value into
    long    len;            //  length of the buffer
} config_def_t;

config_def_t  config_list[] = {
    {"parameter1",  &g_config_values.parameter1[0], PARM1_SZ},
    {"parameter2", &g_config_values.parameter2[0], PARM2_SZ},
    {"new-parm", &g_config_values.new_parm[0], NEW_PARM_SZ},    // *** New Code ***
    {NULL, NULL, 0}
};

//  data type used to store configuration items
typedef struct {
    char parameter1[PARM1_SZ +1];
    char parameter2[PARM2_SZ +1];
    char new_parm[NEW_PARM_SZ +1];     // *** New Code ***
} config_value_t;

// declare variable to store configuration items into
config_value_t  g_config_values;

// call the SDK Library function to read the configuration file and store the values in g_config_values
read_config("config_file", config_list);
```

12.2. read_config() – Function Call

long read_config(char filename[],config_def_t config_definitions[])

This function can be called by the customer's software to read the contents of a configuration file. This function takes two parameters. The first parameter is the name of the configuration file. The second parameter is a pointer to a data structure that defines the tags to be read and where the values are to be stored. See the discussion above.

The following parameters are passed when this function is called.

filename: The name of the configuration file to be read.

config_definitions: Pointer to a data structure that define the tags to be read and where the values are to be stored.

Returns

1 is returned if successful.

-1 is returned if the configuration file could not be opened.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

13. Network Interface

The SDK Library contains tools to allow the customer's software to do the following:

- Create a stream socket (i.e., establish a TCP connection)
- Create a datagram socket
- Join a multicast group
- Accept connections on stream sockets (i.e., accept TCP connections)

13.1. Creating a Stream Socket

To create a stream socket, the customer's software will first call the SDK Library function **make_tcp_connection()** passing the IP address and Port Number of the remote peer. This function will create a stream socket and attempt to establish the connection. The SDK Library will perform a non-blocking connection attempt. Later the SDK Library will call the customer's software to inform it whether the connection attempt was successful or if the connection attempt failed. The SDK Library will call the callback function **socket_status()** with the results of the connection attempt.

Once the connection is established, the customer's software will be informed when the SDK Library has read data from the connection (socket). The SDK Library will read data from the connection (socket) and call the **data_read()** callback function to inform the customer's software. The data is passed as a parameter to **data_read().** If the customer's software needs to access the data after it returns from **data_read(),** it must make a copy of the data before returning.

When the customer's software wants to write data to a connection (socket), it will call the **write_tcp_data()** function. The **write_tcp_data()** function will attempt to write the data. The **write_tcp_data()** function will return information regarding the attempt to write the data. There are 4 possible returns.

- The SDK Library successfully wrote the data to the socket.
- The socket buffer was full and the SDK Library queued the data. The SDK Library will write the queued data to the socket buffer when space becomes available.
- The socket buffer was full and the SDK Library's queue limit has been reached. The SDK Library will only queue a finite amount of data. If this occurs, the customer's software is writing data too fast and should throttle back sending data.
- There was an error when attempting to write the data to the socket.

If the status of the connection (socket) changes, the customer's software will be informed. The SDK Library will inform the customer's software by calling the **socket_status()** callback function. For example, if the peer drops the connection, the SDK Library will inform the customer's software by calling the **socket_status()** callback function.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

13.2. Creating a Datagram Socket

To create a datagram socket, the customer's software will first call the SDK Library function **make_datagram()** passing the IP address and Port Number of the customer's application. The **make_datagram()** function will create the datagram socket and bind the socket to the IP address and Port Number specified by the customer's software.

Once the datagram socket has been created, the customer's software will be informed when data has been received. The SDK Library will read data from the socket and call the **data_read()** callback function to inform the customer's software. The data is passed as a parameter to **data_read().** If the customer's software needs to access the data after it returns from **data_read(),** it must make a copy of the data before returning.

When the customer's software wants to write data (send a datagram), it will call the **write_datagram()** function and pass the IP address and port number the datagram is to be sent to. The **write_datagram()** function will attempt to write the data. The **write_datagram()** function will return the number of bytes actually written to the socket. Note, the SDK Library will not queue data if the datagram socket buffer is full. A datagram service is a "best effort" service. The SDK Library will simply attempt to write the data to the datagram socket. The customer's software must take appropriate action if the data cannot be written to the datagram socket.

If the status of the datagram socket changes, the customer's software will be informed. The SDK Library will inform the customer's software by calling the **socket_status()** callback function.

13.3. Joining a Multicast Group

To join a multicast group, the customer's software will first call the SDK Library function **join_mcast_group()** passing the IP address and Port Number assigned to the multicast group, and the IP address of the interface that the multicast data will be received on. The **join_mcast_group()** function will attempt to join the multicast group and will return whether the attempt was successful or failed.

Once the SDK Library has successfully joined the multicast group, the customer's software will be informed when data has been received. The SDK Library will read the data and call the **data_read()** callback function to inform the customer's software that data has been read. The data is passed as a parameter to **data_read().** If the customer's software needs to access the data after it returns from **data_read(),** it must make a copy of the data before returning.

If the status of the multicast socket changes, the customer's software will be informed. The SDK Library will inform the customer's software by calling the **socket_status()** callback function.

13.4. Accepting Connections on Stream Sockets

To accept connections on a stream socket, the customer's software will first call the SDK Library function **create_listen_socket()** passing the Port Number that the customer's software wants to accept connections on. The **create_listen_socket()** function will create the listen socket and bind the socket to the Port Number specified by the

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

customer’s software. The **create_listen_socket()** function will return whether creating the listen socket was successful or failed.

Later when a connection has been accepted by the SDK Library, the customer’s software will be informed via the **connection_accepted()** callback function.

Once a connection is established, the customer’s software will be informed when data has been read from the connection (socket). The SDK Library will read data from the connection (socket) and call the **data_read()** callback function to inform the customer’s software. The data is passed as a parameter to **data_read().** If the customer’s software needs to access the data after it returns from **data_read(),** it must make a copy of the data before returning.

When the customer’s software wants to write data to a connection (socket), it will call the **write_tcp_data()** function. The **write_tcp_data()** function will attempt to write the data. The **write_tcp_data()** function will return information regarding the attempt to write the data. There are 4 possible returns.

- The SDK Library successfully wrote the data to the socket.
- The socket buffer was full and the SDK Library queued the data. The SDK Library will write the queued data to the socket buffer when space becomes available.
- The socket buffer was full and the SDK Library’s queue limit has been reached. The SDK Library will only queue a finite amount of data. If this occurs, then the customer’s software is writing data too fast and should throttle back sending data.
- There was an error when attempting to write the data to the socket.

If the status of the connection (socket) changes, the customer’s software will be informed. The SDK Library will inform the customer’s software by calling the **socket_status()** callback function. For example, if the remote peer drops the connection, the SDK Library will inform the customer’s software by calling the **socket_status()** callback function.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

13.5. make_tcp_connection() – Function Call

int make_tcp_connection(char ip[], int port)   // Linux Sockets

SOCKET make_tcp_connection(char ip[], int port)   // Windows Sockets

This function is called by the customer's software to create a stream socket and establish a TCP connection. The IP address and port number of the remote peer are passed as parameters.

NOTE: When this function returns, the connection has not been established. Only the socket for the connection has been created. The SDK Library will perform a non-blocking connection attempt. Later, the customer's software will be informed whether the connection attempt was successful or failed. The **socket_status()** callback function is used to inform the customer's software whether the connection attempt was successful or failed.

The following parameters are passed when this function is called.

ip: the IP address of the remote peer.

port: the port number of the remote peer.

Returns

If the socket can be created, this function returns an identifier for the socket that is associated with the connection. This value will be passed to other functions to identify the socket (connection). For example, this value may be passed to **write_tcp_data()** to identify which socket the data is to be written.

Minus one (-1) is returned if the socket cannot be created on a Linux platform. On a Windows platform, INVALID_SOCKET is returned if the socket cannot be created.

13.6. data_read() – Callback Function

void data_read(int sock, char data[], int len)   // Linux Sockets

void data_read(SOCKET sock, char data[], int len)   // Windows Sockets

This function is called by the SDK Library to inform the customer's software that data has been read from a socket.

The following parameters are passed when this function is called.

sock: specifies the socket the data has been read from.

data: contains the data read from the socket.

len: specifies the amount of data read.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

13.7. write_tcp_data() – Function Call

int write_tcp_data(int sock, char data[], int len)   // Linux Sockets

int write_tcp_data(SOCKET sock, char data[], int len)   // Windows Sockets

This function is called by the customer's software to write data to a socket.

The following parameters are passed when this function is called.

sock: specifies the socket the data is to be written to.

data: contains the data to be written.

len: specifies the amount of data to be written.

Returns

Return Value	Description
0	The data was successfully written to the socket
1	The socket buffer was full and the data was queued by the SDK Library
2	The socket buffer was full and the SDK Library queue limit has been exceeded. In other words, the data was not queued by the SDK Library.
-1	Socket error. There was an error writing to the socket.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

13.8. make_datagram()– Function Call

int make_datagram(char ip[], int port) // Linux Sockets

SOCKET make_datagram(char ip[], int port) // Windows Sockets

This function is called by the customer's software to create a datagram socket. The IP address and Port Number parameters identify the application (customer's software). If the customer's software does not care what IP address is assigned to it, then it can pass a null string in the ip parameter.

The following parameters are passed when this function is called.

ip: the IP address that identifies the application (customer's software). A null string can be passed if the customer's software does not care what address is assigned.

port: the port number that identifies the application (customer's software).

Returns

If the datagram socket can be created, this function returns an identifier for the socket. This value will be passed to other functions to identify the socket. For example, this value may be passed to **write_datagram()** to identify which socket the datagram is to be written.

Minus one (-1) is returned if the socket cannot be created on a Linux platform. On a Windows platform, INVALID_SOCKET is returned if the socket cannot be created.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

13.9. write_datagram() – Function Call

int write_datagram(int sock, char data[], int len, char ip[], int port) // Linux Sockets

int write_datagram(SOCKET sock, char data[], int len, char ip[], int port) // Windows Sockets

This function is called by the customer's software to write data to a datagram socket.

The following parameters are passed when this function is called.

sock: specifies the socket the data is to be written to.

data: contains the data to be written.

len: specifies the amount of data to be written.

ip: specifies the IP address the data is to be sent to.

port: specifies the Port Number the data is to be sent to.

Returns

The number of bytes written to the datagram socket.

Minus one (-1) is returned if the socket passed is not recognized by the SDK Library or if an error occurs while writing to the datagram socket.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

13.10. join_mcast_group() – Function Call

int join_mcast_group(char ip[], int port, char iface_ip[])   // Linux Sockets

SOCKET join_mcast_group(char ip[], int port, char iface_ip[])   // Windows Sockets

This function is called by the customer's software to join a multicast group. The IP address and Port Number parameters identify the multicast group. The iface_ip parameter identifies that network interface that multicast data is to be received on

The following parameters are passed when this function is called.

ip: the IP address of the multicast group to be joined.

port: the port number of the multicast group to be joined.

iface_ip: the IP address of the network interface that multicast data is to be received on.

Returns

If the multicast socket can be created, this function returns an identifier for the socket. This value will be passed to other functions to identify the socket.

Minus one (-1) is returned if the multicast socket cannot be created on a Linux platform. On a Windows platform, INVALID_SOCKET is returned if the socket cannot be created.

13.11. create_listen_socket() – Function Call

int create_listen_socket(int port)   // Linux Sockets

SOCKET create_listen_socket(int port)   // Windows Sockets

This function is called by the customer's software to create a listen socket that will be used to accept TCP connections from remote peers. The port number that the customer's software wants to accept connections on is passed as a parameter.

The following parameter is passed when this function is called.

port: the Port Number the customer's software wants to accept connections on.

Returns

If the listen socket can be created, this function returns an identifier for the socket.

Minus one (-1) is returned if the socket cannot be created on a Linux platform. On a Windows platform, INVALID_SOCKET is returned if the socket cannot be created.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

13.12. connection_accepted() – Callback Function

void connection_accepted(int port, int sock)   // Linux Sockets

void connection_accepted(int port, SOCKET sock)   // Windows Sockets

This function is called by the SDK Library to inform the customer's software that a TCP connection has been accepted on a listen socket, that was previously created with a **create_listen_socket()** call. The socket identifier used to identify the connection is passed as a parameter.

The following parameter is passed when this function is called.

port: the port number assigned to the listen socket that the connection was accepted on.

sock: identifies the TCP connection that was accepted.

13.13. socket_status() – Callback Function

void socket_status(int sock, int status)   // Linux Sockets

void socket_status(SOCKET sock, int status)   // Windows Sockets

This function is called by the SDK Library to inform the customer's software about the status of a socket.

The following parameters are passed when this function is called.

sock: specifies the socket the status is associated with.

status: specifies the status of the socket. The following are valid values:

Value	Description
1	TCP connection has been established
2	TCP connection attempt failed
3	TCP connection has been deleted
4	Datagram stockt has been deleted.
5	Multicast group socket has been deleted.
6	Listen socket has been deleted.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved

13.14. delete_socket() – Function Call

int delete_socket(int sock)   // Linux Sockets

int delete_socket(SOCKET sock)   // Windows Sockets

This function is called by the customer's software to request that the SDK Library delete a socket. The sock parameter specifies the socket to be deleted.

The following parameter is passed when this function is called.

sock: specifies the socket to be deleted.

Returns

Zero (0) is returned if the socket was deleted.

Minus one (-1) is returned if the socket could not be deleted. For example, the socket passed is not recognized by the SDK Library.

Copyright ©2010 Lightspeed Financial, Inc. Proprietary Information. All rights reserved