eSpeed C/C++ Application Program Interface

REFERENCE GUIDE

System Version: 1.4.16

Revision Date: November 30, 2006

Document Version: ESPD – 1

Table of Contents

1. Introduction

This document describes the C interface and the message flow of the eSpeedAPI.

Other documents relating to the eSpeedAPI:

eSpeedAPI Install and Build Guide describes how to install the eSpeedAPI on the development platform and how to build your application with the eSpeedAPI.

eSpeedAPI Release Notes contains the changes for a particular release version of the eSpeedAPI.

eSpeedAPI FHLB Trade Confirm presents the mechanism for accessing and interpreting the additional information available in an trade confirmation from the FHLB system that is not described in the eSpeedAPI Reference Guide.

1.1 Overview

The eSpeed Electronic Trading Interface is a system that provides the necessary functionality to effectively trade and post markets with eSpeed trading systems and provides users with the basic information suitable for formulating trading decisions. The eSpeed API allows third parties to develop their own applications that use the eSpeed system. This document describes the usage of the eSpeed API.

Certified applications that have been built using the eSpeed API are connected to the eSpeed system through a Session Manager, which provides authentication and management of the connections made by those applications.

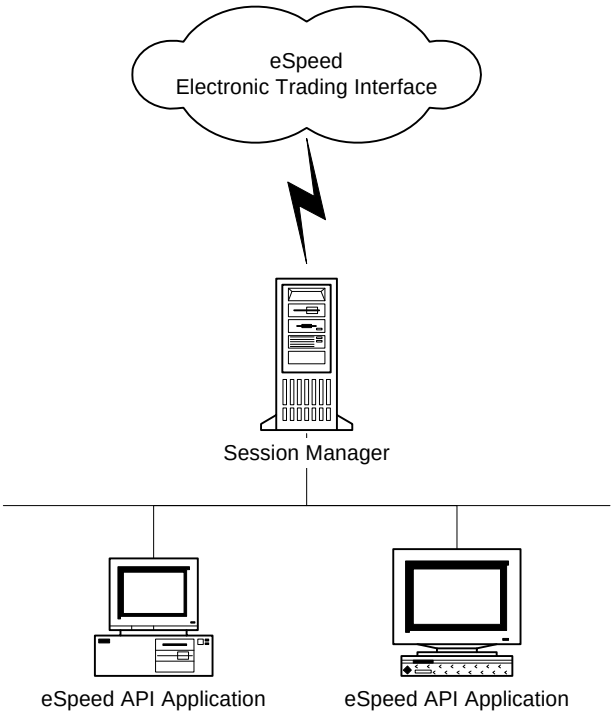

Figure 1 - System Overview

The eSpeed API requires an application first to establish a physical connection to the eSpeed Session Manager and create one or more user logins. This is achieved using the `CFETIOpenSession` and

`CFETILogin` interfaces provided in the API. Once a session has been established a second level of authentication is required to permit the user to create a connection to an eSpeed Trading System. Once connected the user is then able to perform market and trading operations as required using the interfaces supplied. Facilities are also provided to allow the user to submit simple queries (e.g. to identify instruments of interest) and to subscribe to market data for those instruments.

Each eSpeed operation is made through a provided interface in the eSpeed API. Each action is validated and then delivered to the Session Manager. If an error occurs during this processing an error code is returned by the interface to the application to indicate the nature of the problem. Otherwise, a success code is returned. A full list of eSpeed API error codes is given in Appendix A.

The success or failure of the requested operation is delivered to the application through callback functions specified by the application. For example, if a login request can be processed successfully the `CFETILogin` interface will generate the return code `CFETI_SUCCESS`. Login has not been completed, however, until the command `CFETI_LOGIN_ACCEPTED` (or `CFETI_LOGIN_REJECTED`) has been delivered to the application callback that was specified in the `CFETILogin` call. Any data delivered by the eSpeed API to an application through one if its supplied callback functions should be copied if it is to be preserved by the application. In most cases, memory allocated for such data is released once the callback function has completed. A full list of eSpeed API data types is given in 2.12.3.

For every command and action in the interface specification, the eSpeed API returns a response to the user that submitted the request. This includes login and connection management as well as all trading, query and subscription commands. Where necessary, subsequent responses and global notifications are distributed to all trading participants. This command response is delivered asynchronously to the application. In order to receive these messages it is incumbent upon eSpeed API applications to use the message processing functions provided by the eSpeed API. Facilities are provided to process messages forever using a message loop, to wait for a single inbound message, as well as facilities to poll for incoming message queues. Full details of the message processing functions provided in the API are discussed in section 2.16.

Log files are created automatically by the eSpeed API when the application is run. By default, the log files are written to the current directory. This location can be modified by setting the environment variable `CFETI_LOG_PATH`. The first is a transaction log and is used to record requests generated by eSpeed API applications and the responses that are received. The file is named `cfeti.nn` where `nn` is the day of the month on which the file was created. A new transaction log file is created each day. The second file is an information log used by the eSpeed API to record system events. The file is named `cfetilog.nn` where `nn` is the day of the month on which the file was created. A new information log file is created each day. The names of the log files are changed in libESPD to be ESPD_transaction.nn for the transaction log file and ESPD_log.nn for the information log.

The eSpeed API provides a level of recovery. If for any reason the connection to the eSpeed Session Manager should fail then the eSpeed API will send notification to user defined callbacks and then commence recovery. When reconnection to an eSpeed Session Manager is complete further notifications are sent and then the API will attempt to automatically restore any logins and trading system connections that were ‘live’ when the connection first failed. Similarly, if some other component of the eSpeed system should fail notifications are delivered to the application and the API will automatically attempt to restore trading system connections when the system is recovered.

1.2 Customer Support

Customer Support is available between 7:00am and midnight U.S. Eastern time, Monday through Friday, excluding holidays. When calling, please assist us by being ready with your product and account information.

The eSpeed Customer Support group has a series of phone numbers and e–mail addresses to meet various user needs. They are as follows:

If you are a Customer with questions about possible trading scenarios or specific trading features please contact your account representative.

If you are a Customer experiencing technical difficulty, have questions on how to use the system, please call or write:

eSpeed Call Center (US) — (+1) (212) 610-2300 or support@espeed.com
eSpeed Help Desk (Europe) — (0)20-7894–8600 or support@espeed.co.uk

If you are a Customer or Salesperson requesting information on how to make changes to your electronic account, or require a new access account, please call or write:

eSpeed Customer Access (US) — (212) 610-2300 or customeraccess@espeed.com
eSpeed Customer Access (Europe) — (0)20-7894-8886 or customeraccess@espeed.co.uk

If you are a Customer with questions regarding API development issues, downloading the latest version of the SDK or JNI, or testing a trade feed or market–making application, please call (between the hours of 9:00am and 6:00pm EST) or write:

eSpeed Customer Integration — (+1) (212) 610-3560 or customerintegration@espeed.com

If you are a Customer with questions regarding specific trades, verification of a trade, or are experiencing delivery problems with a trade, please call:

eSpeed Trade Support (US) — (212) 610-2300
eSpeed Trade Support (Europe) — (0)20-7894-8600

2. Application Interface

2.1 Open eSpeed Session

The first action of any eSpeed API application must be to establish a session with the eSpeed Session Manager. Initialization creates a persistent connection to the Session Manager and will manage the state of the connection. Status concerning the established connection is not delivered to the application until at least one login has successfully been established. Creation of a second session is prohibited by the eSpeed API.

```
CFETI_RC CFETIOpenSession(
                        const char *primarySessMgr,
                        const char *secondarySessMgr,
                        CFETI_IDENTIFICATION pIdent
);
```

`primarySessMgr`	The name and instance of the primary session manager. (HostName:Instance)
`secondarySessMgr`	The name and instance of the secondary session manager. (HostName:Instance)
`pIdent`	Application identification information

2.1.1 Application Identity

The application should create and populate an instance of the CFETI_IDENTIFICATION_DESC structure to identify the application name, its version and the organization that created it as well as details of the platform and operating system and then pass a pointer to this structure in the call to `CFETIOpenSession`. The information supplied will be copied by the eSpeed API when `CFETIOpenSession` is invoked.

```
typedef struct CFETI_IDENTIFICATION_DESC
                        CFETI_IDENTIFICATION_DESC;
struct CFETI_IDENTIFICATION_DESC {
    const char* Company;
    const char* Application;
    const char* Version;
    const char* Platform;
    const char* OperatingSystem;
};
```

2.1.2 Return Codes

A successful return code, `CFETI_SUCCESS`, indicates that the authentication request has been successfully processed. (Not that the login has been successful – the success or failure of the login is delivered to the application through the supplied system callback). Failure indicates that the eSpeed API request has not been processed. In this case the reason is indicated in the return code. Return codes that may be delivered are listed below.

`CFETI_SUCCESS`	Successful return code indicates that the eSpeed API routine has been successfully processed.
`CFETI_SESSMGR_SPEC`	Invalid session manager name specification in call to `CFETIOpenSession`.

`CFETI_CONNECT_FAIL`	Connection to the session manager(s) specified in the primary and secondary session manager names was not successful.
`CFETI_DIRECTORY_READ_ERROR`	The directory optionally specified by environment variable `CFETI_LOG_PATH` for log-files does not exist.
`CFETI_FILE_WRITE_ERROR`	A log file could not be opened for writing.

2.2 Close eSpeed Session

It is recommended that an eSpeed API application should close an eSpeed Session before terminating. Once a session has been closed a new one can be created by calling `CFETIOpenSession()`.

```
void CFETICloseSession();
```

2.3 Login

eSpeed authentication is completed at two levels: system and trading system. System authentication validates the user credentials, including location and active logins.

```
CFETI_RC CFETILogin(
                        const char *userName,
                        const char *userPassword,
                        CFETI_PREF systemPreferences,
                        CFETI_SYSTEM_CB systemCallback,
                        CFETI_UD userData,
                        char *userTag );
```

`userName`	The name of the User/Organization connecting to the system.
`userPassword`	Password for the User/Organization.
`systemCallback`	Callback routine that will be invoked for all system related messages for this login session.
`systemPreferences`	System User preferred settings that can be specified to control the behavior of the eSpeed API. Preferences can be combined using logical OR (|).
`userData`	Data specified by the user, which will be returned to the `systemCallback` on each invocation for this login.
`userTag`	Tag to associate with the User connecting to the system. Your customer service representative will notify you if you need to supply a value in this field. Otherwise, you can supply NULL (0) for this parameter. The parameter is optional if you are using a C++ compiler.

The eSpeed API encrypts login information before transmission. Upon successful authentication, the eSpeed system returns a digital signature to the user that must be provided on all subsequent eSpeed API requests. A list of the trading system entities that they are entitled to interact with is also returned.

The interface may be used again to create additional logins with the session manager.

2.3.1 System Preferences

If no preferences are set (i.e. `cmdPreferences` is zero) the default action is for eSpeed API to cancel all markets and orders when the user disconnects or logs out. This behavior can be modified using the following.

`CFETI_RETAIN_MKTS_ON_CLOSE`	Markets will be retained on logout / disconnect.
`CFETI_RETAIN_ORDERS_ON_CLOSE`	Orders will be retained on logout / disconnect.

2.3.2 Return Codes

A successful return code, `CFETI_SUCCESS`, indicates that the authentication request has been successfully processed. (Not that the login has been successful – the success or failure of the login is delivered to the application through the supplied system callback). Failure indicates that the eSpeed API request has not been processed. In this case the reason is indicated in the return code. Return codes that may be delivered are listed below.

`CFETI_SUCCESS`	Successful return code indicates that the eSpeed API routine has been successfully processed. Command details will be returned in the `systemCallback`.
`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETILogin` (e.g. `userName` is null or is an empty string).
`CFETI_NO_CALLBACK`	No callback routine was supplied
`CFETI_LOGIN_EXISTS`	A login request for this user has previously been issued.

2.3.3 Command Response

Responses to an authentication request are as follows:

Authentication Status	Authentication Details
Authentication Accepted	Session Identification
Authentication Rejected	Possible Reasons: • Invalid authentication credentials. • User already logged into system. ▪ Authentication error

All system-level command responses relating to this session and login are returned in the callback routine provided in the initial `CFETILogin` call. The eSpeed API invokes the callback routine as follows:

```
void systemCallback(
                        CFETI_CMD cmd,
                        CFETI_CMD_STATUS cmdStatus,
                        CFETI_CMDDATA cmdData,
                        CFETI_UD userData );
```

`Cmd`	Command value indicating system status (e.g. login accepted or rejected).
`cmdStatus`	Additional status associated with the given command (e.g. reason for rejected login).
`cmdData`	Command specific data structure that may contain information useful to the application.
`userData`	User data that was specified when the `CFETILogin` call was made is delivered to the application callback function.

Commands that may be issued by the eSpeed API to the application as a result of the login request are as follows:

Command	Command Status	Command Data Type
CFETI_LOGIN_ACCEPTED	CFETI_SUCCESS	CFETI_LOGIN_INFO*
CFETI_LOGIN_REJECTED	CFETI_INVALID_CREDENTIALS CFETI_USERNAME_CURRENTLY_CONNECTED CFETI_CANNOT_AUTHENTICATE	char* szUserName

Login information is delivered to the application if authentication is successful. This information comprises a session identifier that is required to be used by the application for subsequent eSpeed API function calls and a null-terminated list of trading systems that the successfully authenticated user is permitted to attempt to connect to. If the login is rejected the user name is returned in the command data.

The login information structure and the associated data types are defined as follows:

```
typedef char* CFETI_SESSION_ID;
typedef unsigned int CFETI_TRADING_SYSTEM;
typedef struct CFETI_LOGIN_INFO CFETI_LOGIN_INFO;
typedef struct CFETI_TRADING_SYS_DESC CFETI_TRADING_SYS_DESC;
typedef CFETI_TRADING_SYS_DESC* CFETI_TRADING_SYS;

struct CFETI_TRADING_SYS_DESC {
    CFETI_TRADING_SYSTEM tsId;
    char *tsDescription;
};

struct CFETI_LOGIN_INFO {
    CFETI_SESSION_ID   sessionId;
    CFETI_TRADING_SYS* ts;
    const char*         szConnectionMode;
    const char*         szUserId; /**< eSpeed user identity
*/
};
```

As with any data delivered by the eSpeed API to an application through its callbacks, information that is to be preserved by the application (e.g. the session id since this will be required in subsequent calls to the eSpeed API) should be copied.

The system callback may be subsequently called as status related to the connection is affected. In such cases the `cmd` and `cmdStatus` will indicate the event that occurred and action that eSpeed API applications should take (if any). Throughout the eSpeed API the command status is a pointer to a structure containing an integer status code and, if it is not null, a string describing the nature of the error.

Command	Command Status	Command Data Type
CFETI_LOGIN_TERMINATED	CFETI_FORCED_LOGOFF	CFETI_SESSION_ID

CFETI_LOGIN_TERMINATED	CFETI_TRADE_SYS_OFFLINE	CFETI_SESSION_ID
CFETI_STATUS	CFETI_SESSION_LOST	CFETI_SESSION_ID
CFETI_STATUS	CFETI_SESSION_RESTORED	CFETI_SESSION_ID
CFETI_STATUS	CFETI_TRADE_SYS_OFFLINE	CFETI_LOGIN_INFO*
CFETI_STATUS	CFETI_TRADE_SYS_ONLINE	CFETI_LOGIN_INFO*

The CFETI_LOGIN_INFO* data passed to the system callback when the command status is CFETI_TRADE_SYS_OFFLINE or CFETI_TRADE_SYS_ONLINE, is a null terminated list of affected trading systems. When the command status is CFETI_SESSION_LOST this indicates that the connection to the eSpeed Session Manager has been lost. No further action is required by the eSpeed API application – the application will be notified through the same callback with command status CFETI_SESSION_RESTORED when the connection to the eSpeed Session Manager has been restored.

The field szConnectionMode indicates the connection mode that was used by the eSpeed API to establish the connection to the Session Manager.

2.4 Logout

eSpeed API applications may logout from a session manager to which they have previously logged in using CFETILogout. When logging out, applications must supply a valid session identifier, which was the session identifier received as a result of a previously successful login. System preferences may be provided to override those submitted during the login process.

```
CFETI_RC CFETILogout(
                        const CFETI_SESSION_ID sessId,
                        CFETI_PREF systemPreferences );
```

SessId	Valid session identifier from previously successful login.
systemPreferences	User preferred settings to override the preferences specified during login. No value (0) indicates that the preferences are not changed. (See 2.3.1 - System Preferences)

2.4.1 Return Codes

A successful return code, CFETI_SUCCESS, indicates that the logout request has been successfully processed, otherwise a non-zero return code is returned to indicating the reason for the failure. Return codes that may be delivered to the application include:

CFETI_SUCCESS	Successful return code indicates that the eSpeed API routine has been successfully processed. Command details (i.e. whether or not the logout was accepted or rejected) are returned in the systemCallback specified when the original login request was made.
CFETI_INVALID_ARG	Invalid argument supplied to CFETILogout (e.g. session Id is null).
CFETI_NO_SUCH_LOGIN	A login identified by the supplied session identifier cannot be found.

2.4.2 Command Response

Responses to a logout request are as follows:

Authentication Status	Authentication Details
Logout Accepted	N/A
Logout Rejected	Possible Reasons: • User not currently logged into system.

All command responses are returned in the callback routine provided in the initial call to the eSpeed API authentication function. The callback routine is invoked as follows:

```
void systemCallback(
                        CFETI_CMD cmd,
                        CFETI_CMD_STATUS cmdStatus,
                        CFETI_CMDDATA cmdData,
                        CFETI_UD userData);
```

`cmd` — Command value indicating system status (i.e. logout accepted or rejected).

`cmdStatus` — Additional status associated with the given command (e.g. reason for rejected logout).

`cmdData` — Command specific data structure that may contain information useful to the application.

`userData` — User data that was specified when the `CFETILogin` call was made is delivered to the application callback function.

Commands that may be issued by the eSpeed API to the application as a result of the logout request are as follows:

Command	Command Status	Command Data Type
CFETI_LOGOUT_ACCEPTED	CFETI_SUCCESS	CFETI_SESSION_ID
CFETI_LOGOUT_REJECTED	CFETI_USERNAME_CURRENTLY_NOT_CONNECTED	CFETI_SESSION_ID

2.5 Trading System Connect

After a successful login to the system has been established, eSpeed API applications must connect to the trading systems with which they wish to interact. All subsequent markets and orders that are posted through the eSpeed API will be routed to the trading system specified. When connecting to a specific trading system, applications may specify a trading system specific password and connection preferences that will override those submitted during the login process if desired (see 2.3.1 - System Preferences). (If a password is not supplied, the `userPassword` provided in the authentication call `CFETILogin` will be sent to the trading system along with the session identifier and `userName` and these will be used to authenticate the user's connection to the trading system).

```
CFETI_RC CFETIConnect(
                        const CFETI_SESSION_ID sessId,
                        const char *userPassword,
                        CFETI_TRADING_SYSTEM tradingSystem,
                        CFETI_PREF tradingSysPreferences,
                        CFETI_CONNECT_CB tradingSysCallback,
                        CFETI_UD userData,
                        CFETI_TRADE_SETTINGS_DESC* pSettings );
```

`sessId` — Valid session identifier from previously successful login.

`userPassword`	Optional user password – should be supplied null if the system-level password is to be used.
`tradingSystem`	Trading System identifier of the trading system to which the user wishes to connect.
`tradingSysPreferences`	Trading System user preferred settings that can be specified to control the behavior of the eSpeed API. These preferences will override those specified during the login. (I.e. Cancel all markets on disconnect, Cancel all orders on disconnect).
`tradingSysCallback`	Callback routine that will be invoked for all messages for this trading system connection.
`userData`	Data specified by the user, which will be returned to the `tradingSysCallback` on each invocation for this connection.
`pSettings`	Trade settings to be applied to markets and orders for this trading session. This mechanism is added for future use. Applications should pass in 0 for this parameter.

2.5.1 Return Codes

A successful return code, `CFETI_SUCCESS`, indicates that the connection request has been successfully processed (not that it has been successful – the success or failure of the connection is delivered to the application through the supplied trading system callback). Failure indicates that the eSpeed API request has not been processed. In this case the reason is indicated in the return code. Return codes that may be delivered to the application include:

`CFETI_SUCCESS`	Successful return code indicates that the eSpeed API routine has been successfully processed. Command details (i.e. whether or not the connection was accepted or rejected) are returned in the `tradingSysCallback`.
`CFETI_INVALID_CREDENTIALS`	The user does not have permission to connect to the trading system.
`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETIConnect` (e.g. session Id is null).
`CFETI_NO_CALLBACK`	Callback function was not specified in call to `CFETIConnect`.
`CFETI_NO_SUCH_LOGIN`	The login identified by the supplied session identifier cannot be found.
`CFETI_NO_SESSION`	The connection request has been made without having previously established a session with the eSpeed Session Manager.

2.5.2 Command Response

Responses to a connection request are as follows:

Connect Status	Connect Details
Connection Accepted	Trading System Session Id, Trading System Name/Value
Connection Rejected	Possible Reasons: • Invalid authentication credentials. • User already logged into system. ▪ Authentication error ▪ Trading System is Offline

All eSpeed API command responses relating to this trading system connection are returned in the callback routine provided in the initial `CFETIConnect` call. The eSpeed API invokes the callback routine as follows:

```
void tradeSysCallback(
                        CFETI_CMD cmd,
                        CFETI_CMD_STATUS cmdStatus,
                        CFETI_CMDDATA cmdData,
                        CFETI_UD userData );
```

`cmd`	Command value indicating connection status (i.e. connection accepted or rejected).
`cmdStatus`	Additional status associated with the given command (e.g. reason for rejected connection).
`cmdData`	Command specific data structure that may contain information useful to the application.
`userData`	User data that was specified when the connection call was made is delivered to the callback function.

Commands that may be issued by the eSpeed API to the application as a result of the connection request are as follows:

Command	Command Status	Command Data Type
`CFETI_CONNECTION_ACCEPTED`	`CFETI_SUCCESS`	`CFETI_CONNECT_INFO*`

`CFETI_CONNECTION_REJECTED`	`CFETI_INVALID_CREDENTIALS`
	`CFETI_USERNAME_CURRENTLY_CONNECTED`
	`CFETI_CANNOT_AUTHENTICATE`
	`CFETI_TRADE_SYS_OFFLINE`
		`CFETI_SESSION_ID`

Connection information is delivered to the application if connection is successful. This information comprises a trading system session id, a description of the trading system to which connection has been made, and trade control flags indicating which trading actions are permitted on the connection. If trade control flags for the connection are subsequently modified, they will be delivered to the connection callback.

If the connection is rejected the session id of the requester is returned in the command data to the connection callback. The connection information structure and associated data types are defined as follows:

```
typedef unsigned int CFETI_TRADE_SESS_ID;
typedef unsigned int CFETI_TRADING_SYSTEM;
typedef struct CFETI_CONNECT_INFO CFETI_CONNECT_INFO;
typedef struct CFETI_TRADING_SYS_DESC CFETI_TRADING_SYS_DESC;
typedef unsigned int CFETI_TRADE_CONTROL_FLAGS;
```

```
struct CFETI_TRADING_SYS_DESC {
    CFETI_TRADING_SYSTEM       tsId;
    char                      *tsDescription;
};

struct CFETI_CONNECT_INFO {
    CFETI_TRADE_SESS_ID          sessionId;
    CFETI_TRADING_SYS_DESC       tradingSystem;
    CFETI_TRADE_CONTROL_FLAGS    tradeFlags;
    CFETI_TRADE_CONTROL_FLAGS    tradeFlags2;
    CFETI_TRADE_SETTINGS_DESC    tradeSettings;
};
```

The trade control flags field tradeFlags is a bit-mask and can be set to a combination of values indicated below.

CFETI_BID_DISABLED	Bid is disabled.
CFETI_ASK_DISABLED	Ask is disabled.
CFETI_BUY_DISABLED	Buy is disabled.
CFETI_SELL_DISABLED	Sell is disabled.
CFETI_GIVEUP_ENABLED	Name-giveup enabled.
CFETI_TC_INSTRUMENT_DATA_ENABLED	The trading system will deliver instrument attributes data in trade confirmation messages.
CFETI_LINKED_MARKETS_ENABLED	The trading system interface supports a facility to submit a basket of markets to which user specified actions can be attributed.
CFETI_CHECKOUT_CLIENT_USER	The trading system provides support for electronic checkout of trades where there was broker action taken on behalf of the user during its execution.

The trade control flags field tradeFlags2 is a bit-mask and can be set to a combination of values indicated below.

CFETI_ORDER_ALLOCATION_ENABLED	The trading system provides support for inclusion of user-specified allocation text in markets/orders.
CFETI_CHECKOUT_ALLOCATION_ENABLED	The trading system provides support for inclusion of user-specified allocation text when accepting trades where there was broker action taken on behalf of the user during its execution.

The trade settings field describes other trading session settings. This field is added for future use.

For giveup-enabled business units, CFETI_GIVEUP_ENABLED will be set. As with any other data delivered by the eSpeed API to the application through its callbacks, information that is to be preserved by the application should be copied.

The trading system connection callback may be subsequently called as status related to the connection is affected. In such cases the `cmd` and `cmdStatus` will indicate the event that occurred and action, which eSpeed API applications should take (if any).

Command	Command Status	Command Data Type
CFETI_CONNECTION_TERMINATED	CFETI_FORCED_DISCONNECT	CFETI_CONNECT_INFO*
CFETI_CONNECTION_TERM	CFETI_TRADE_SYS_OFFLINE	CFETI_CONNECT_INFO*

INATED		
CFETI_STATUS	CFETI_SESSION_LOST	N/A
CFETI_STATUS	CFETI_SESSION_RESTORED	N/A
CFETI_STATUS	CFETI_TRADE_SYS_OFFLINE	CFETI_TRADING_SYSTEM
CFETI_STATUS	CFETI_TRADE_SYS_ONLINE	CFETI_TRADING_SYSTEM
CFETI_STATUS	CFETI_TRADE_SUBSYSTEM_OFFLINE	CFETI_TRADING_SUBSYSTEM
CFETI_STATUS	CFETI_TRADE_SUBSYSTEM_ONLINE	CFETI_TRADING_SUBSYSTEM
CFETI_CONNECTION_MODIFIED	CFETI_TRADE_CONTROL_FLAGS_MODIFIED	CFETI_TRADE_CONTROL_FLAGS

Subsequent to a successful trading system connection, a refresh of the markets previously submitted and orders previously queued by that user that are remaining on the trading system at the time of connection is delivered to the connection callback. There will only be markets and/or orders on the system if a user preference to retain them was set for the previous session for that user. (Full details of the refresh mechanism are given in section 2.11). The command status values CFETI_TRADE_SUBSYSTEM_OFFLINE and CFETI_TRADE_SUBSYSTEM_ONLINE indicate that a sub-component of the trading system is affected. Markets and orders posted to that trading system will include a non-zero trading subsystem value in subsequent updates (e.g. when a market is accepted). These status values should be handled in a similar way to the trading system off-line and on-line states for the affected markets and orders. For example, if the connection preference was not to retain markets on disconnection, these markets are removed when the trading sub-system enters off-line state.

If the trading system connection is for a name-giveup business unit (such as Interest Rate Derivatives), details of the user's credit-worthiness with other counterparties is stored within the system in what is known as a Giveup Matrix.

After connection to a name-giveup business, the application connection callback will be invoked with the command `CFETI_GIVEUP_MATRIX_RECEIVED` and passed the id of an eSpeed API field, the value of which must always be passed to `CFETIDecodeDataField` (see 2.18.3).

If the Giveup Matrix is subsequently modified or deleted, the application trading callback will be invoked with the commands `CFETI_GIVEUP_MATRIX_RECEIVED` and `CFETI_GIVEUP_MATRIX_DELETED` respectively. No command data will be sent in either case.

Command	Command Status	Command Data Type
CFETI_GIVEUP_MATRIX_RECEIVED	CFETI_SUCCESS	unsigned int
CFETI_GIVEUP_MATRIX_CHANGED	CFETI_SUCCESS	N/A
CFETI_GIVEUP_MATRIX_DELETED	CFETI_SUCCESS	N/A

2.6 Trading System Disconnect

eSpeed API applications may disconnect from trading systems to which they have connected using `CFETIDisconnect`. When disconnecting, applications must supply the session identifier and the trading session identifier to disconnect from. Trading system preferences (e.g. retain current markets or orders) may be provided to override those submitted during the connection process.

```
CFETI_RC CFETIDisconnect(
                        const CFETI_SESSION_ID sessId,
                        CFETI_TRADE_SESS_ID trdSysSessId,
                        CFETI_PREF tradingSysPreferences );
```

`sessId`	Valid session identifier from previously successful login.
`trdSysSessId`	Id of trading system connection that is to be closed.
`tradingSysPreferences`	User preferred settings to override the preferences specified during connection. No value (0) indicates that the preferences are not changed. (See 2.3.1 - System Preferences)

2.6.1 Return Codes

A successful return code, `CFETI_SUCCESS`, indicates that the disconnection request has been successfully processed. Otherwise, the reason for the failure is indicated in the return code. Return codes that may be delivered to the application include:

`CFETI_SUCCESS`	The request was processed successfully. Command details (i.e. whether or not the disconnection request was accepted or rejected) are returned in the `tradingSysCallback` specified when the original connection request was made.
`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETIDisconnect` (e.g. session Id is null).
`CFETI_NO_SESSION`	The connection request has been made without having previously established a session with the eSpeed Session Manager.
`CFETI_NO_SUCH_LOGIN`	The login identified by the supplied session identifier cannot be found.
`CFETI_NO_SUCH_CONNECTION`	The connection specified by the supplied trading system session identifier cannot be found.

2.6.2 Command Response

Responses to a disconnection request are as follows:

Connect Status	Connect Details
Disconnection Accepted	Trading System Session Id, Trading System Name/Value
Disconnection Rejected	Possible Reasons:

	• Invalid authentication credentials.

All command responses will be returned in the callback routine provided in the initial `CFETIConnect` call. The callback routine is invoked as follows:

```
void tradeSysCallback(
                        CFETI_CMD cmd,
                        CFETI_CMD_STATUS cmdStatus,
                        CFETI_CMDDATA cmdData,
                        CFETI_UD userData );
```

`cmd`	Command value indicating disconnection status (i.e. accepted or rejected).
`cmdStatus`	Additional status associated with the given command (e.g. reason for rejected disconnection).
`CmdData`	Command specific data structure that may contain information useful to the application.
`userData`	User data that was specified when the original connection call was made is delivered to the callback function.

Commands that may be issued by the eSpeed API to the application as a result of the disconnection request are as follows:

Command	Command Status	Command Data Type
`CFETI_DISCONNECT_ACCEPTED`	`CFETI_SUCCESS`	`CFETI_CONNECT_INFO*`
`CFETI_DISCONNECT_REJECTED`	`CFETI_INVALID_CREDENTIALS` `CFETI_USERNAME_CURRENTLY_NOT_CONNECTED`	`CFETI_CONNECT_INFO*`

After a successful disconnection, the trading system session id `trdSysSessId` is no longer valid. Application wishing to communicate further with that trading system must reestablish a connection using `CFETIConnect`.

2.7 Bid/Offer

Bids and Offers submitted must contain a complete description of the market being posted including the trade instrument, price and size as well as system and trading system authentication identifiers.

Markets submitted through the eSpeed API will be acknowledged upon receipt as accepted or rejected. If accepted, the acknowledgment will indicate the market that was accepted (price, trade instrument, size and side). If rejected, the acknowledgment will indicate the reason why the market was not processed by the system. (E.g. invalid syntax, better market…).

```
CFETI_RC CFETIPostMessage(
                        const CFETI_SESSION_ID sessId,
                        CFETI_TRADE_SESS_ID trdSysSessId,
                        CFETI_CMD cmd,
                        CFETI_CMDDATA cmdData,
                        CFETI_CMDPREF cmdPreferences );
```

`sessId`	Valid session identifier from previously successful login.
`trdSysSessId`	Trading system session identifier returned on successful

	connection.
`Cmd`	Command (request) being submitted. To post a market this shall be `CFETC_POST_MARKET`. (The CFETI commands that can be posted using `CFETPPostMessage` are defined in `cfeti_consts.h`. Each has the prefix `CFETC`).
`cmdData`	Command data that contains the details of the request. The `CFETIPostMessage` interface is used for a number of different eSpeed API operations and the command data varies accordingly. In the case of bids and offers a `CFETI_MARKET` is expected as the command data.
`cmdPreferences`	User preferred settings that can be specified to control the behavior of the market being posted. (E.g. interpret size as incremental or total).

2.7.1 Market Definition

The `CFETI_MARKET` supplied to `CFETIPostMessage` for bids and offers is defined as follows:

```
typedef unsigned char* CFETI_INSTRUMENT;
typedef double CFETI_PRICE;
typedef double CFETI_SIZE;
typedef unsigned char CFETI_MARKET_SIDE;
typedef unsigned int CFETI_ID;
typedef unsigned int CFETI_TRADING_SUBSYSTEM;
typedef unsigned char* CFETI_TRADE_ID;
typedef struct CFETI_MARKET_DESC CFETI_MARKET_DESC;

struct CFETI_MARKET_DESC {
    CFETI_INSTRUMENT            tradeInstrument;
    CFETI_PRICE                 price;
    CFETI_SIZE                  size;
    CFETI_MARKET_SIDE           side;
    CFETI_PREF                  preferences;
    CFETI_TRADE_ID              tradeId;
    CFETI_SIZE                  tradeSize;
    time_t                      tradeTime;
    CFETI_ID                    id;
    CFETI_TRADING_SUBSYSTEM     subsystem;
    void*                       userData;
    unsigned short              userDataSize;
    void*                       appUserData;
    unsigned short              appUserDataSize;
    const char*                 shortCode;
    CFETI_ID                    requestId;
    CFETI_TRADING_SYSTEM        tsId;
    unsigned int                priceImprovement;
    unsigned int                orderInfoType;
    void*                       orderInfo;
    CFETI_PRICE                 altPrice1;
    CFETI_PRICE                 altPrice2;
    unsigned int                basketId;
    unsigned int                basketActions;
    const char*                 userName;
    const char*                 requestorId;
    const char*                 originatorId;
    time_t                      creationTime;
    const char*                 allocationInfo;
    CFETI_PRICE                 executionPrice;
    const CFETI_TRADE_SETTINGS_DESC* tradeSettings;
    unsigned int                timeOffset;
    CFETI_SIZE                  reserveMinSize;
```

```
    CFETI_SIZE                  reserveMaxSize;
    CFETI_SIZE                  reserveInitialSize;
    const char*                 tradeReference;
    CFETI_MARKET_SIDE           tradeSide;
    Const char*                 counterpartyName;
};

typedef CFETI_MARKET_DESC* CFETI_MARKET;
```

The market side element of this structure should have the value `CFETI_MARKET_BID` for a bid or `CFETI_MARKET_ASK` for an offer. The `tradeId, tradeSize and tradeTime` elements of the structure are only valid in the trading system connection callback when the command is `CFETI_MKT_EXECUTED`. Their value at any other time is undefined. The market size should be specified as a multiple of the size multiplier field, which can be retrieved from the instrument data (Section 2.12). For example, if the value of the size multiplier is 1 million (1000000) then a bid for 5 million should be submitted as 5000000. The trade size should be interpreted in the same way. The preferences element of this structure is not required when posting markets. In subsequent command responses it may contain the preferences passed to the CFETIPostMessage call, which should then be combined with any additional required preferences if canceling the market.

The subsystem component of the market structure is not required to be filled in when the market is submitted. If this field is non-zero in subsequent responses, it indicates the sub-system of the trading system to which the market was posted that is handling the market. The status of the trading sub-system is reported to the application's trading system connection callback (see section 2.5.2).

The user data component of the market structure is provided to allow the application to attach its own data to the entered market. The appUserData component of the market structure is provided to allow an application programmer to provide a facility to allow the application user to attach data to the market being posted. When subsequent messages are received for the market which include the market details, the user data and application user data shall be returned also. The maximum size of the user data is `CFETI_MAX_USER_DATA_SIZE` bytes. The size of the user data and application user data shall be indicated by the application in the corresponding size field of the market structure.

The short code component of the market structure enables an application to specify the name of a third party on whose behalf a trade is made. When subsequent messages are received for the market that includes the market details, the short code shall be returned also. The maximum size of the short code is CFETI_MAX_SHORTCODE_LENGTH bytes. Some trading systems may not support this facility.

When the API delivers any individual notification for the market (see 0), the `tsId` field in the market structure is set to be equal to the id for that business unit.

The field `reserved3` is reserved for future use.

The field `creationTime` is used to deliver the time at which the market was created. If the value in the field is 0 then this information is not available for the corresponding trading system.

The field `allocationInfo` can be used by client applications to send allocation information with the market. Availability of the facility to include allocation information is indicated at the time of connection to the trading system in the trade control flags.

`basketId` is an identifier that is used by the client application to link this market to others that it has created and/or submitted.

`basketActions` is a bit-mask specifying actions that the user attaches to markets linked with this identifier. The list of possible actions is defined in the table below.

For businesses that support it `requestorId` will contain the userId that submitted the request in market notifications. It should not be set by the application in submitted markets.

For businesses that support it `originatorId` will contain the userId that first submitted the request in market notifications. It should not be set by the application in submitted markets.

The field `executionPrice` will deliver the price executed in market-executed notifications.

If the market is specified as good until time, the number of minutes for which it shall be valid is specified in the `timeOffset` field.

The trade settings field is used to deliver the trade settings for the posted market. This is added for future use – applications should pass 0 for this field.

The fields `reserveMinSize`, `reserveMaxSize` and `reserveInitialSize` are used when submitting a market using the eSpeed Max Display feature. These values specify the upper and lower bounds of the max display order and optionally the initial size to display. The `preferences2` bit-mask in the market structure must include the value CFETI_USE_RESERVE_SIZE.

The field `tradeReference`, if populated in execution notifications, will contain the trade reference common to all execution notifications for a market. It is not guaranteed that this same reference will be delivered in subsequent trade confirmation notifications.

It is strongly recommended that the application should initialize the entire market structure to zero before filling in any of the fields. For example:

```
CFETI_MARKET_DESC mkt;
memset((char*)&mkt, 0, sizeof(mkt));
```

2.7.1.1 OrderInfo and OrderInfoType

The fields orderInfo and orderInfoType in the market are used to describe trade server specific information regarding a market. The orderInfoType field is used to determine what type of information is stored in the order info. These fields can be valid in any message where the market structure is used but will depend upon the trading system.

The orderInfo field should be treated as invalid if the orderInfoType field is zero.

2.7.1.1.1 Treasury Swaps & Interest Rate Swaps

These fields will be valid only when the `orderInfoType` is `CFETI_ORDERINFO_TSWAP` and the `orderInfo` field shall be populated with the CFETI_TSWAP_DESC structure. The CFETI_TSWAP_DESC structure is also used as order info in the order structure. Please see section 2.9.1.1.5 for details. A synonym is also provided for the order info type as CFETI_ORDERINFO_IRS_VS_FUTURE and a synonym for the CFETI_TSWAP_DESC data structure is provided as CFETI_IRS_VS_FUTURE_DESC.

- When the lock price of the benchmark to which a US Treasury Swaps instrument or Interest Rate Swap vs Future instrument is linked is modified, the necessary changes applied by the trading system will notify the user application with the market structure and the market moved command.

Market Response	**Market Details**
Market Moved	Trade instrument, price, size, side

2.7.2 Command Preferences

The following preferences can be set to control the behavior of the market being posted. If no preferences are set (i.e. `cmdPreferences` is zero), the default action is for eSpeed API to post markets with total size. If the stated preference cannot be provided by the trading system or if the specified combination of preferences is invalid, the command response will be a rejection of the posted market and the reason will be delivered in the command status. Not all of the preferences listed are supported by all trading systems.

CFETI_MARKET_GOOD_TILL_CANCEL	The market price is a firm price (i.e. the price cannot be bested or cut by the eSpeed system. The price exists until it is traded or explicitly cancelled).
CFETI_MARKET_SIZE_IS_TOTAL	The market size is the total size (use to override any existing size for an already posted market).
CFETI_MARKET_SIZE_IS_INCREMENTAL	The market size is to be added to any existing size for an already posted market.
CFETI_MARKET_ALL_OR_NONE	The market should be completed in its entirety, or not at all.
CFETI_MARKET_LIMIT_PRICE	The market price specified is the limit (i.e. the maximum price to be paid).
CFETI_TRADE_OPTION_1	Generic trade option that is given meaning according to the instrument being traded.
CFETI_TRADE_OPTION_2	As CFETI_TRADE_OPTION_1
CFETI_TRADE_OPTION_3	As CFETI_TRADE_OPTION_1
CFETI_MARKET_DISABLE_DERIVATION	The created market should not be used by the eSpeed system to derive further markets. If derivation can not be disabled for this instrument the command status code CFETI_DERIVATION_MANDATORY is delivered in the market rejection response.
CFETI_MARKET_INDICATIVE	The market price is an indicative price. That is, the price cannot be aggressed by a buy or sell order. The existence of an indicative bid or offer is delivered in the market data stream. This option can only be used if your account is enabled for indicative market contribution for the business in question. For more information please contact your eSpeed Customer Integration representative.
GOOD_UNTIL_TIME	Market is valid for the number of minutes specified in the timeOffset field in the market structure. The time starts from receipt of the market by the eSpeed system.
CFETI_ONLY_AT_BEST	Only at best markets are submitted without a price. Instead the price shall be determined by the eSpeed system upon receipt of the market. The assigned price will then not subsequently change for the life of that market.

The following preferences can be set to control the behavior of the market being posted in the `preferences2` field of the market structure

CFETI_USE_RESERVE_SIZE	Market or order is specified with a minimum and maximum reserve size and a total size for reserve.

2.7.2.1 European Repos

For European Repos the trade options are used to indicate the mechanisms that can be used to clear the trade if the market is aggressed. The options can be combined to indicate that clearing can be through more than one mechanism.

CFETI_TRADE_OPTION_1	CFETI_TRADE_CLEARING_LCH
CFETI_TRADE_OPTION_2	CFETI_TRADE_CLEARING_CLEARNET

```
CFETI_TRADE_OPTION_3        CFETI_TRADE_CLEARING_INTERBANK
```

2.7.2.2 Interest Rate Derivatives

For Interest Rate Derivatives the first and third trade options are used to indicate the mechanisms that can be used to clear the trade if the market is aggressed. These options can be combined to indicate that clearing can be through more than one mechanism. The second option is used to indicate that the price is a derived spread price.

```
CFETI_TRADE_OPTION_1        CFETI_TRADE_CLEARING_LCH
CFETI_TRADE_OPTION_2        CFETI_TRADE_DERIVED_SPREAD
CFETI_TRADE_OPTION_3        CFETI_TRADE_CLEARING_INTERBANK
```

2.7.3 Linked Markets

Markets in FX Options and other term structure products frequently require a user to enter a “run” of prices. That is, the user will enter a price in an option for multiple maturities with the intention of filling just one of the markets. When one of the markets is filled, the remaining markets are no longer executable.

A mechanism is available by which eSpeed applications can create an association between markets submitted in FX options. Each market submitted is given a basket identifier by the client application. The submitted markets are also given a basket action that specifies to the trading interface the actions that the client application attaches to markets submitted with the same basket identifier. The basket identifier attached to a market is specified in the `basketId` field of the CFETI_MARKET_DESC data structure. The actions that the client application attaches to markets in the basket are specified as a bit-mask in the `basketActions` field of the same data structure. The same fields are also provided in CFETI_ORDER_DESC data structure but are not currently used.

All markets submitted with the same basket identifier for a trading session are treated by the trading interface as being in the same group of markets. If a market is submitted in a basket where another market with the same basket identifier is already executing, the new market shall be rejected by the trading system with the command status code set to CFETI_BASKET_MARKET_TRADE_IN_PROGRESS. When a market in the basket is executed the trading system shall respond according to the actions specified on that market. The list of actions available to client applications is listed in the table below. If an action is specified that is not recognized by the trading system the market will be rejected with the command status code set to CFETI_BASKET_ACTION_NOT_RECOGNISED.

Action	Description
`CFETI_BASKET_ACTION_CANCEL_ON_EXECUTE`	Markets with the same basket identified shall be cancelled if one of those markets is executed. When this event occurs notification of the cancel of each of the markets in the basket shall be delivered with the command CFETI_MKT_CANCELLED. The command data shall be a pointer to a CFETI_MARKET_DESC data structure and the command status shall be CFETI_BASKET_MARKET_EXECUTED.
`CFETI_BASKET_ACTION_CANCEL_ALL`	This action shall be specified by client applications only when submitting a cancel request for a market in a basket. The action instructs the trading interface to identify and cancel all other markets for the same user that have the same basket identifier and issue cancel notifications for these markets. These cancellations shall be delivered to the client application trading system connection callback with the command set to CFETI_MKT_CANCELLED the command status code set to CFETI_BASKET_CANCELLED.

2.7.4 Spot/Out Protection

Markets in FX Options frequently require a user to enter a floor/ceiling in the spot rate that, if the boundary is crossed, shall cause the FX option markets to be cancelled. A mechanism is available by which eSpeed applications can specify these limits for FX Option markets. When an FX option bid/offer is executed the FX spot rate ceiling and floor are recorded by the system. If the mid-point of the current FX spot market bid/offer exceeds the user defined ceiling or floor then the FX Option markets are cancelled. Furthermore, if the bid or offer in the FX option is executed, the ceiling/floor are validated against the mid-point of the prevailing spot rates. The trade is not executed if this is outside the ceiling or floor prices and the market is cancelled.

The fields `altPrice1` and `altPrice2` in the CFETI_MARKET_DESC data structure are used to communicate the lower and upper spot rates for the market tolerance respectively. The client application is required to set the trading preference CFETI_TOLERANCE_SET in the preferences bit-mask when submitting the markets with tolerances specified.

Markets are cancelled by the trading system when the spot rate is outside of the tolerances specified. These cancellations shall be delivered to the client application trading system connection callback with the command set to CFETI_MKT_CANCELLED the command status code set to CFETI_TOLERANCE_EXCEEDED.

2.7.5 Return Codes

A successful return code, `CFETI_SUCCESS`, indicates that the disconnection request has been successfully processed. Otherwise, the reason for the failure is indicated in the return code. Return codes that may be delivered to the application include:

`CFETI_SUCCESS`	The request was processed successfully. Whether or not the posted market was accepted or rejected by the trading system is returned in the `tradingSysCallback` specified when the original connection request was made.
`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETIPostMessage`.
`CFETI_NO_SESSION`	An attempt is made to post an eSpeed API request without having previously established a session with the eSpeed Session Manager.
`CFETI_NO_SUCH_LOGIN`	The login identified by the supplied session identifier cannot be found.
`CFETI_NO_SUCH_CONNECTION`	The connection specified by the supplied trading system session identifier cannot be found.

2.7.6 Command Response

Responses to a post market request are as follows:

Command Response	Command Details
Market Accepted	Market Posted (trade instrument, price, size, and side)
Market Rejected	Market not posted. (trade instrument, price, size and side) Possible Reasons: • Invalid authentication signature • Unknown trade instrument. • Insufficient price and/or size.

	• Market already executed. • Transaction not approved. • Instrument not currently trade-able • Action prohibited (e.g. bid disabled) • Price entered would cause the market to backwardate • Price entered is not the best price • Quantity entered is the same as that of a current market • Price entered differs from the current market by too great an amount • Quantity entered is below the minimum • Price is outside the boundary of the number of price levels the trading system can hold • Disabling of derivation is not available for this instrument • Market was submitted using max display feature but one or more of the initial, minimum or maximum reserve size specified in the market was invalid.
Market Not Executed	Market posted but could not be executed (trade instrument, price, size and side). Possible Reasons: • An attempt was made to aggress a price that was not available. This response is only applicable to name-giveup business units.

All responses related to the posted market are returned in the callback routine provided in the initial `CFETIConnect` call. The callback routine is invoked as follows:

```
void tradeSysCallback(
                                CFETI_CMD cmd,
                                CFETI_CMD_STATUS cmdStatus,
                                CFETI_CMDDATA cmdData,
                                CFETI_UD userData );
```

Commands that may be issued by the eSpeed API to the application as a result of the posted market request are as follows:

Command	Command Status	Command Data Type
CFETI_MKT_ACCEPTED	CFETI_SUCCESS	CFETI_MARKET
CFETI_MKT_REJECTED	CFETI_UNKNOWN_TRD_INST CFETI_INVALID_AUTH CFETI_INVALID_MARKET CFETI_TRADE_IN_PROGRESS CFETI_TRANSACTION_NOT_APPROVED CFETI_INSTRUMENT_NOT_TRADEABLE CFETI_ACTION_PROHIBITED CFETI_MARKET_WOULD_BACKWARDATE CFETI_NOT_BEST_PRICE CFETI_SAME_QUANTITY_SPECIFIED CFETI_PRICE_CHANGE_TOO_GREAT CFETI_QUANTITY_TOO_SMALL CFETI_INVALID_SWAP_NUMBER CFETI_INVALID_LOCK_PRICE CFETI_INVALID_CURRENT_LOCK_PRICE CFETI_MARKET_NOT_BEST_PRICE CFETI_INVALID_CURRENT_SWAP_NUMBER CFETI_PRICE_LEVELS_EXCEEDED CFETI_DERIVATION_MANDATORY	CFETI_MARKET

	CFETI_INVALID_RESERVE_SIZE	
CFETI_MKT_NOT_EXECUTED	CFETI_NO_CREDIT	CFETI_MARKET

2.7.7 Individual Notification

Following successful submission of a market, the submitter will receive subsequent messages when events occur which affect that market.

- Notification will be received when the either the bid is bested or the offer is cut, either of these notifications indicates that the market has been removed and is no longer active. (A market-cancelled response will also be delivered when the market is cleared for some other reason). If a market cancelled notification is delivered with a market-to-follow command status code then this indicates to the client application that a subsequent market created notification will be delivered.

Market Response	Market Details
Market Cancelled	Market Cancelled (trade instrument, price, size and side) Possible Reasons: • Bid Bested • Market Cleared • Offer Cut • Cancelled by system (size is amount cancelled) • Transaction now disapproved by eSpeed • Market is not best price

- Notification will be received when the posted market is hit or taken. The notification will indicate the size of the market executed and will be assigned a unique trade identifier. A trade confirmation with the given unique trade identifier will be received which summaries the trade. The trade identifier may be used in subsequent queries.

Market Response	Market Details
Market Executed	trade instrument, price, size, side and trade identifier, trade size and trade time.

- If any part of the market remains after trading is completed (i.e. size exceeds size done in trade confirmation), and if the trading system provides the facility to do so, a new market shall be introduced by the system. This will be at the traded price and for either the portion of the market that was not filled, or for a base unit size. (The actual size selected will depend upon agreement between the customer and eSpeed, LLC., prior to commencement of electronic trading).

Market Response	Market Details
Market Created	trade instrument, price, size, side

- When the lock price of the benchmark to which an instrument is linked is modified, the necessary changes applied by the trading system will notify the user application with the market structure and the market moved command.

  Alternatively, if the API user has the Market As Order preference set and the trading system has determined that the market posted can be elevated to a trade state at a better price, the market structure and the market moved command will be issued to notify the user application of the new price.

Market Response	Market Details

Market Moved	trade instrument, price, size, side

- If an API user has the MarketAsOrder preference set and a market is posted which can be elevated to a trade state while a trade is ongoing, but there is not currently any size available to execute against the market, then the trading system will notify the user application with the market structure and the market executing command. This indicates that the market will be executed if more size becomes available during the trade, in which case the user application will be notified with the market structure and the market executed command..

Market Response	Market Details
Market Executing	trade instrument, price, size, side

Commands that may be issued by the eSpeed API to the application as a result of the subsequent cancellation or execution of the market are therefore as follows.

Command	Command Status	Command Data Type
CFETI_MKT_EXECUTED	CFETI_SUCCESS CFETI_CHECK_CREDIT	CFETI_MARKET[1]
CFETI_MKT_CANCELLED	CFETI_MKT_BESTED CFETI_MKT_CLEARED CFETI_MKT_CUT CFETI_MKT_CANCELLED_BY_SYSTEM CFETI_TRANSACTION_DISAPPROVED CFETI_MARKET_NOT_BEST_PRICE CFETI_MARKET_TO_FOLLOW CFETI_ORDER_EXPIRED	CFETI_MARKET
CFETI_TRADE_PENDING	CFETI_SUCCESS	CFETI_ORDER[2]
CFETI_TRADE_CONFIRM	CFETI_SUCCESS	CFETI_ORDER[2]
CFETI_MKT_CREATED	CFETI_MKT_CLEARED	CFETI_MARKET
CFETI_MKT_MOVED	CFETI_SUCCESS	CFETI_MARKET
CFETI_MKT_EXECUTING	CFETI_SUCCESS	CFETI_MARKET

The command status value CFETI_CHECK_CREDIT may be set when the market is executed in a giveup-enabled business. This indicates that a manual credit check will be necessary before the trade can be confirmed.

The command status value CFETI_ORDER_EXPIRED is delivered for markets specified with the good-until-time preference when the market has expired.

If when trading is complete a manual process must be applied to the trade before it is confirmed (e.g. a credit check for a giveup-enabled business) the command CFETI_TRADE_PENDING shall be delivered to the application. When the trade is subsequently confirmed a trade confirmation shall be delivered. For some businesses a CFETI_TRADE_REJECTED command may be delivered to cancel the trade.

2.7.8 Global Notification

Bids and Offers that are accepted by the trading system will be distributed to all users that are subscribed to the instrument affected as a market data update (See 2.12 - Subscribe/Unsubscribe). The update will include the necessary components to indicate the new market (trade instrument, price, size and side) as well as updates to the relevant bid or offer participant list.

Market Response	Market Details
Market Data Update	trade instrument, price, size and side. (i.e. Bid 98.12+ Bid

[1] In this case the `tradeId` component of the market description will be a unique id for this trade generated by the trading system.
[2] See 2.9.1 - Order Definition

	Size 10)

2.8 Cancel Bid/Offer

Bids and Offers may be cancelled only after the minimum market time has elapsed. [3] Cancellations submitted must contain the full description of the market that is being removed. Market cancellations submitted to the eSpeed API will be acknowledged upon receipt as accepted or rejected. If accepted, the acknowledgment will indicate the market that has been removed from the system (price, trade instrument, size and side). In the event that a portion of the market has been executed prior to the receipt of the cancellation, the acknowledgment will indicate the portion of the market cancelled. If rejected, the acknowledgment will indicate the reason why the market was not removed by the system.

```
CFETI_RC CFETIPostMessage(
                        CFETI_SESSION_ID sessId,
                        CFETI_TRADE_SESS_ID trdSysSessId,
                        CFETI_CMD cmd,
                        CFETI_CMDDATA cmdData,
                        CFETI_CMDPREF cmdPreferences );
```

`sessId`	Valid session identifier from previously successful login.
`trdSysSessId`	Trading system session identifier returned on successful connection.
`Cmd`	Command (request) being submitted. To cancel a market this shall be `CFETC_CANCEL_MARKET`.
`cmdData`	Command data that contains the details of the request. In the case of market cancellation a `CFETI_MARKET` is expected as the command data.
`cmdPreferences`	User preferred settings that can be specified to control the behavior of the market cancellation being posted. (E.g. cancel all markets for the given trade instrument).

2.8.1 Command Preferences

The following preferences can be set to control the behavior of the market cancellation. If no preferences are set (i.e. `cmdPreferences` is zero), the default action is for eSpeed API to cancel the specified market only. If the stated preference cannot be provided by the trading system or if the specified combination of preferences is invalid, the command response will be a rejection of the market cancellation request and the reason will be delivered in the command status. These preferences should be combined with any returned in the market structure when the market was accepted by the trading system.

`CFETI_MARKET_CANCEL_ALL_FOR_ISSUE`	Cancel all of this users markets for the specified instrument.

2.8.2 Return Codes

[3] The minimum market timeout will be dictated by the trading system and may vary between business units.

A successful return code, `CFETI_SUCCESS`, indicates that the market cancel request has been successfully processed. Otherwise, the reason for the failure is indicated in the return code. Return codes that may be delivered to the application include:

`CFETI_SUCCESS`	The request was processed successfully. Whether or not the market was cancelled by the trading system is returned in the `tradingSysCallback` specified when the original connection request was made.
`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETIPostMessage`.
`CFETI_NO_SESSION`	An attempt is made to cancel a market without having previously established a session with the eSpeed Session Manager.
`CFETI_NO_SUCH_LOGIN`	The login identified by the supplied session identifier cannot be found.
`CFETI_NO_SUCH_CONNECTION`	The connection specified by the supplied trading system session identifier cannot be found.

2.8.3 Command Response

Responses to a cancel market request are as follows:

Command Response	Command Details
Cancel Accepted	Market cancel command successfully received by system for processing(trade instrument, price, size and side)
Cancel Rejected	Market cancellation was unsuccessful (trade instrument, price, size and side) Possible Reasons: • Invalid authentication signature • Market not posted. (I.e. Insufficient price and/or size.) • Minimum Market time not exceeded. • Market already executed. • Invalid id
Market Cancelled	Market cancelled successfully(trade instrument, price and side)
Market Cancel Queued	A request to cancel is received that cannot be acted upon immediately. An acknowledgement of the request to cancel is delivered. The request may subsequently either be accepted or rejected.

The command response related to the cancellation market is returned in the callback routine provided in the initial `CFETIConnect` call. The callback routine is invoked as follows:

```
void tradeSysCallback(
                                CFETI_CMD cmd,
                                CFETI_CMD_STATUS cmdStatus,
                                CFETI_CMDDATA cmdData,
                                CFETI_UD userData );
```

Commands that may be issued by the eSpeed API to the application as a result of the requested cancellation of the market are as listed below.

Command	Command Status	Command Data Type
`CFETI_CANCEL_MARKET_ACCEPTED`	`CFETI_SUCCESS`	`CFETI_MARKET`
`CFETI_CANCEL_MARKET_REJ`	`CFETI_UNKNOWN_TRD_INST`	`CFETI_MARKET`

ECTED	CFETI_INVALID_AUTH CFETI_INVALID_MARKET CFETI_INSUFFICIENT_TIME CFETI_INVALID_ID	
CFETI_MARKET_CANCELLED	CFETI_SUCCESS CFETI_SUBSTITUTED	CFETI_MARKET
CFETI_MKT_CANCEL_QUEUED	CFETI_ATTEMPTING_SUBSTIT UTION	CFETI_MARKET

A market cancellation request will be acknowledged by the system with the delivery of a Cancel Market Accepted/Rejected message. In the case of a Cancel Market Accepted message this is an acknowledgement that the cancel command has been received and is being processed by the system, not an indication that the market has been cancelled. Only upon receipt of a Market Cancelled message has the market been removed from the system.

A market cancellation will be considered successful as long as a portion of the market may be removed. It follows that if a prior portion of the market was executed, but cancellation of the remainder was possible, the cancel market command is considered successful. The market returned in the command data will indicate the portion of the market that was removed from the system.

2.9 Buy/Sell

Buy and Sell orders can be submitted for any tradable instrument and will be queued for execution at the price and size indicated in the request. A buy or sell request will not be considered valid until it is processed and then queued by the eSpeed trading system concerned. Therefore, users and systems should mark orders as suspect until a successful acknowledgement is returned.

```
CFETI_RC CFETIPostMessage(
                        CFETI_SESSION_ID sessId,
                        CFETI_TRADE_SESS_ID trdSysSessId,
                        CFETI_CMD cmd,
                        CFETI_CMDDATA cmdData,
                        CFETI_CMDPREF cmdPreferences );
```

`sessId`	Valid session identifier from previously successful login.
`trdSysSessId`	Trading system session identifier returned on successful connection.
`cmd`	Command (request) being submitted. To submit an order this shall be `CFETC_SUBMIT_ORDER`.
`cmdData`	Command data that contains the details of the request. In the case of a buy or sell a `CFETI_ORDER` shall be expected as the command data.
`cmdPreferences`	User preferred settings that can be specified to control the behavior of the order being submitted. (E.g. complete fill only).

2.9.1 Order Definition

The `CFETI_ORDER` supplied to `CFETIPostMessage` for buys and sells is defined as follows:

```
typedef unsigned char* CFETI_INSTRUMENT;
typedef double CFETI_PRICE;
typedef double CFETI_SIZE;
```

```
        typedef unsigned char CFETI_ORDER_INDICATOR;
        typedef unsigned char* CFETI_TRADE_ID;
        typedef unsigned int CFETI_TRADING_SUBSYSTEM;
        typedef struct CFETI_ORDER_DESC CFETI_ORDER_DESC;
        typedef unsigned char CFETI_TRADE_SIDE;

    struct CFETI_ORDER_DESC {
     CFETI_INSTRUMENT        tradeInstrument;
     CFETI_PRICE             price;
     CFETI_SIZE              size;
     CFETI_ORDER_INDICATOR   indicator;
     CFETI_PREF              preferences;
     CFETI_TRADE_ID          tradeId;
     CFETI_SIZE              tradeSize;
     time_t                  tradeTime;
     CFETI_TRADE_SIDE        tradeSide;
     CFETI_PRICE             tradePrice;
     time_t                  tradeSettlement;
     const char*             tradeReference;
     unsigned short          tradeConfirmOperation;
     unsigned int             recordVersion;
     unsigned int            legId;
     unsigned int            legCount;
     CFETI_ID                id;
     CFETI_TRADING_SUBSYSTEM subsystem;
     void*                   userData;
     unsigned short          userDataSize;
     void*                   appUserData;
     unsigned short          appUserDataSize;
     const char*             userName;
     char*                   shortCode
     CFETI_PRICE             toPrice;
     CFETI_PRICE             altPrice1;
     CFETI_PRICE             altPrice2;
     CFETI_ID                requestId;
     time_t                  endDate;
     time_t                  tradeRepoEndDate  /*Deprecated */
     unsigned int            instrumentIdType;
     CFETI_INSTRUMENT        instrumentId;
     char*                   tradeComments;
     unsigned int            tradeInfoType;
     char*                   tradeInfo;
     unsigned int            settlementType;
     unsigned int            orderInfoType;
     void*                   orderInfo;
     CFETI_COUNTERPARTY_NAME counterpartyName;
     unsigned int            counterpartyID;
     char*                   contactName;
     char*                   contactTelephoneNumber;
     unsigned int            rejectionId;
     CFETI_TRADING_SYSTEM    tsId;
     char*                   settlementMethod;
     double                  brokerage;
     time_t                  paymentDate;
     unsigned int            instProperties;
     unsigned int            tradeProperties;
     unsigned int            priceImprovement;
     unsigned int            checkoutPermissions;
     char*                   reserved3;
     unsigned int            basketId;
     unsigned int            basketActions;
     const char*             requestorId;
     const char*             originatorId;
     const CFETI_INSTRUMENT_DATA_DESC* instrumentData;
     const char*             clearerTradeId;
     const CFETI_PI_BENEFIT_DESC*   pPIBenefit;
     time_t                  creationTime;
     const char*             allocationInfo;
     unsigned int            dealStructure;
```

```
    unsigned int            tradeType;
    unsigned int            pricingMethod;
    CFETI_PRICE             executionPrice;
    const CFETI_TRADE_SETTINGS_DESC* tradeSettings;
    unsigned int            timeOffset;
    double                  assetSwapLevel;
    CFETI_SIZE              reserveMinSize;
    CFETI_SIZE              reserveMaxSize;
    CFETI_SIZE              reserveInitialSize;
    double                  yield;
    const char*             bicCode;
};
typedef CFETI_ORDER_DESC* CFETI_ORDER;
```

The order side element of this structure should have the value `CFETI_ORDER_SELL` for a sell or `CFETI_ORDER_BUY` for a buy. The `tradeId, tradeSize and tradeTime` elements of the structure are only valid in the trading system connection callback when the command is `CFETI_ORDER_EXECUTED` or `CFETI_TRADE_CONFIRM`. Their value at any other time is undefined. The order size should be specified as a multiple of the size multiplier field that can be retrieved from the instrument data (Section 2.12). For example, if the value of the size multiplier is 1 million (1000000) then an order for 5 million should be submitted as 5000000. The trade size should be interpreted in the same way. The preferences element of this structure is not required when submitting orders. In subsequent command responses it may contain the preferences passed to the CFETIPostMessage call, which should then be combined with any additional required preferences if canceling the order.

The fields `tradeSide`, `tradeSettlement` and `tradePrice` in the order structure will also only be valid in the trading system connection callback when the command is `CFETI_TRADE_CONFIRM`. Exactly which are populated is dependent upon the trading system that delivered the trade confirmation. The `tradeSide` field indicates whether the trade was initiated passively (`CFETI_TRADE_PASSIVE`) or actively (`CFETI_TRADE_ACTIVE`). The `tradeSettlement` field indicates the settlement date for the trade and the `tradePrice` field where it is set indicates the price including commission.

The subsystem component of the order structure is not required to be filled in when the order is submitted. If this field is non-zero in subsequent responses, it indicates the sub-system of the trading system to which the order was submitted that is handling the order. The status of the trading sub-system is reported to the application's trading system connection callback (see section 2.5.2).

The user data component of the order structure is provided to allow the application to attach its own data to the submitted order. The appUserData component of the order structure is provided to allow an application programmer to provide a facility to allow the application user to attach data to the order being posted. When subsequent messages are received for the order which include the order details, the user data and application user data shall be returned also. The maximum size of the user data is `CFETI_MAX_USER_DATA_SIZE` bytes. The size of the user data and application user data shall be indicated by the application in the corresponding size field of the order structure.

The short code component of the order structure enables an application to specify the name of a third party on whose behalf a trade is made. When subsequent messages are received for the order that include the order details, the short code shall be returned also. The maximum size of the short code is CFETI_MAX_SHORTCODE_LENGTH bytes. Some trading systems may not support this facility.

The field `endDate` in the order structure will also only be valid in the trading system connection callback when the command is `CFETI_TRADE_CONFIRM`. Use of the field is business specific. If the instrument traded is a REPO instrument it is the end date of the REPO. If the instrument traded is an Interest Rate Derivative it is the maturity date. If the instrument traded is the futures leg of a basis trade (e.g. International Bonds) then it is the expiry date of the contract. The field `tradeRepoEndDate` is deprecated.

The fields `instrumentId` and `instrumentIdType` in the order structure will also only be valid in the trading system connection callback when the command is `CFETI_TRADE_CONFIRM`. If the instrument id type field is zero then the instrument id field is not defined. The instrument id type shall

contain one of the constants `CFETI_INSTRUMENT_ID_CUSIP` or `CFETI_INSTRUMENT_ID_ISIN` in which case the instrument id is the CUSIP or ISIN of the traded instrument (according to market convention). If neither CUSIP or ISIN is available then one of CFETI_INSTRUMENT_ID_CFID (an eSpeed generated instrument identifier) or CFETI_INSTRUMENT_ID_SYMBOL (an instrument identifier derived from the instrument name) shall be used instead.

The field `tradeComments` in the order structure will also only be valid in the trading system connection callback when the command is `CFETI_TRADE_CONFIRM`. This field may contain textual information regarding the trade. If the field is not set then the content of the statusText field of the status structure passed to the connection callback should be examined instead.

The fields `tradeInfo` and `tradeInfoType` in the order structure are used to describe trade server specific information regarding a trade. The will also only be valid in the trading system connection callback when the command is `CFETI_TRADE_CONFIRM`. Currently only the US REPO trading system provides additional information in this way (in which case the tradeInfoType shall be `CFETI_TRADEINFO_USREPO`. The `tradeInfo` field is a null terminated string and should be treated as invalid if the `tradeInfoType` field is zero.

The field `settlementType` in the order structure may be specified when the command is `CFETI_TRADE_CONFIRM`. It may also be specified in a submitted order, but will be ignored unless the trade server to which the order will be delivered is expecting it. If the settlement type field is zero then the settlement type is not specified. The values that the field can take are listed in cfeti_consts.h with a `CFETI_SETTLEMENT_` prefix.

The field `counterpartyName` and counterpartyID in the order structure shall be specified for name-giveup business units when the command is `CFETI_TRADE_CONFIRM`. Otherwise the field shall remain unpopulated.

The field `contactName` and `contactTelephoneNumber` are the name and telephone number of the contact at the counterparty in a trade.

The field `rejectionId` in the order structure shall be specified for name-giveup business units when the command is `CFETI_ORDER_REJECTED` (See 2.9.7).

When the API delivers any individual notification for the order (see 2.9.5), the `tsId` field in the order structure is set to be equal to the id for that business unit.

The field `settlementMethod` in the order structure shall be specified if the trade was settled at a clearing house, otherwise the field shall remain unpopulated.

The field `brokerage` in the order structure shall be specified for trades settled as name-giveup or via a clearing house.

The field `paymentDate` is the date that counterparties must settle netted cash flows incurred during the life of a traded derivative contract.

The field `instProperties` is a bit-mask of instrument properties, which, if present, should match those available in the market data stream for the traded instrument.

The field `tradeProperties` is a bit-mask value used to describe the properties of a trade. The possible values that may be included in the bit-mask are listed below.

CFETI_TRADE_DIRECT_DEAL	Trade was a result of a direct deal.
CFETI_TRADE_NON_ELECTRONIC	Trade was not executed on the eSpeed Platform.
CFETI_TRADE_NETTED	Trade confirmation is netted.
CFETI_TRADE_ERROR	Trade is delivered in error. Trade feeds should ignore this trade.
CFETI_TRADE_MANAGED	A broker managed the trade.
CFETI_TRADE_IMPROVED_EXECUTION	An improved execution was made.

CFETI_TRADE_NO_COMMISSION	No commission will be applied to this trade.

The field `reserved3` is reserved for future use.

The field `creationTime` is used to deliver the time at which the order was created. If the value in the field is 0 then this information is not available for the corresponding trading system.

The field `allocationInfo` can be used by client applications to send allocation information with the order. Availability of the facility to include allocation information is indicated at the time of connection to the trading system in the trade control flags.

The trade settings field is used to deliver the trade settings for the submitted order. This is added for future use – applications should pass 0 for this field.

For trade confirmations delivered for the Middle Markets business (MMTS) the `brokerage` field in the order structure shall be used to report the commission charged for the trade in dollars. Furthermore, the meaning of the fields `altPrice1` and `altPrice2` in MMTS trade confirmations shall be as follows.:

`altPrice1`	Accrued Interest - This is the amount of interest that is to go to the seller as fraction of a coupon payment (days since last coupon)/ 160 * (quantity).
`altPrice2`	All-In-Price - price that includes the commission and the accrued interest. For SKIP or CORP settlement types this shall also contain any REPO price adjustment.

The field `tradeReference`, if populated in execution notifications, will contain the trade reference common to all execution notifications for an order. If populated in n trade confirmation notifications, the trade reference shall be common to all legs of the trade.

`tradeConfirmOperation`	Enumerated trade confirm operation, when non-zero:
CFETI_TRADE_CONFIRM_OPERATION_NEW_TRANSACTION	New Transaction
CFETI_TRADE_CONFIRM_OPERATION_AMEND_TRANSACTION	Amend Transaction
CFETI_TRADE_CONFIRM_OPERATION_CANCEL_TRANSACTION	Cancel Transaction

`recordVersion` is the version number of the trade.

`legId` is the trade leg number for trade confirms consisting of multiple legs. The leg is is 1, 2, 3 up to `legCount`.

`legCount` is the number of legs in the trade confirmation.

`basketId` is an identifier that is used by the client application to link this market to others that it has created and/or submitted.

`basketActions` is a bit-mask specifying actions that the user attaches to markets linked with this identifier. The list of possible actions is defined in the table below.

For businesses that support it `requestorId` will contain the username of the user that submitted the request in order notifications. It should not be set by the application in submitted orders.

For businesses that support it `originatorId` will contain the username of the user that first submitted the request in order notifications. It should not be set by the application in submitted orders.

The field `executionPrice` will deliver the price executed in order executed notifications.

If the order is specified as good until time, the number of minutes for which it shall be valid is specified in the `timeOffset` field.

For businesses that support it, the asset swap level at the time of the trade is delivered in the field `assetSwapLevel`.

The fields `dealStructure, tradeType` and `pricingMethod` deliver the enumerated values for deal structure, trade type and pricing method, The available enumerations are defined in cfeti_consts.h with prefixes CFETI_DEAL_STRUCTURE_, CFETI_TRADE_TYPE_ and CFETI_PRICING_METHOD_ respectively.

For businesses that support it `instrumentData` will provide a pointer to a data structure that will contain instrument attributes relevant to creating a display price or size from the data provided in the trade confirm data structure. This enables trade-feed applications to avoid subscribing to instruments for such businesses. The same data structure that is delivered in market data updates is used and should be accessed in the same way.

The field `clearerTradeId` will be populated where eSpeed is reporting the trade to a third-party clearer and that clearer requires the same Id for the buy and sell sides of the same trade for matching purposes. This field will also be present in results of trade queries.

The fields `reserveMinSize`, `reserveMaxSize` and `reserveInitialSize` are used when submitting an order using the eSpeed Max Display feature. These values specify the upper and lower bounds of the max display order and optionally the initial size to display. The `preferences2` bit-mask in the order structure must include the value CFETI_USE_RESERVE_`SIZE`.

The field `yield` is a calculated yield price corresponding to the traded price (valid only when the pricing method for the traded instrument is as a price or yield).

The field `bicCode` carries the BIC code in trade confirmations for European Repos.

It is strongly recommended that the application should initialize the entire order structure to zero before filling in any of the fields. For example:

```
CFETI_ORDER_DESC order;
memset((char*)&order, 0, sizeof(order));
```

2.9.1.1 OrderInfo and OrderInfoType

The fields `orderInfo` and `orderInfoType` in the order are used to describe trade server specific information regarding an order. The `orderInfoType` field is used to determine what type of information is stored in the order info. These fields can be valid in any message where the order structure is used but will depend upon the trading system.

The `orderInfo` field should be treated as invalid if the `orderInfoType` field is zero.

2.9.1.1.1 North American Energy

These fields will only be valid in the trading system connection callback when the command is `CFETI_TRADE_CONFIRM` (in which case the `orderInfoType` shall be `CFETI_ORDERINFO_ENERGY_TRADE and the orderInfo` field shall be populated with the CFETI_ENERGY_TRADE_DESC structure shown below).

The definition for the CFETI_ENERGY_TRADE_DESC structure is:

```
/*
 * CFETI_ENERGY_TRADE_DESC: Energy trade details
 */
typedef struct CFETI_ENERGY_TRADE_DESC CFETI_ENERGY_TRADE_DESC;
struct CFETI_ENERGY_TRADE_DESC {
  char*            location;
  char*            commodity;
  unsigned char    settlementType;
  unsigned int     indexTypeID;
```

```
    char*            indexType;
    double           durationDays;
    double           durationHours;
    unsigned char    flowBeginHour;
    unsigned char    flowEndHour;
    char*            counterpartyName;
    char*            contactName;
    unsigned int     contactNameId;
    char*            contactTelephoneNumber;
    CFETI_DATE       startCalendarDate;
    CFETI_DATE       endCalendarDate;
    unsigned int     counterpartyID;
    unsigned char    contractType;
    unsigned int     traderID;
    char*            traderName;
    unsigned int     counterpartyTraderID;
    char*            counterpartyTraderName;
    char*            pointName;
    unsigned int     pointNumber;
    double           totalQuantity;
    unsigned char    quantityUnit;
    char*            currency;
    unsigned char    priceType;
    char*            indexName;
    char*            loadProfile;
    unsigned char    holidayFlag;
    unsigned char    productPeriod;
    char*            shortLocation;
    char*            shortIndexName;
    char*            timeZone;
    char*            priceMechanism;
    unsigned int     indexTypeID2;
    char*            indexType2;
    char*            indexName2;
    char*            shortIndexName2;
};
typedef CFETI_ENERGY_TRADE_DESC* CFETI_ENERGY_TRADE;
```

Field Name	Description
Location	Physical location
commodity	Commodity Code
settlementType	Product type. 'F' = 70 for financial and 'P' = 80 for physical. Also represented by the macros: CFETI_ENERGY_SETTLEMENT_TYPE_UNSPECIFIED CFETI_ENERGY_SETTLEMENT_PHYSICAL CFETI_ENERGY_SETLLEMENT_FINANCIAL
indexTypeID	Enumerated index type. Possible values are provided in `cfeti_consts.h`
IndexType	Value "NYM" for some NTGAS products
durationDays	The total number of days of flow
durationHours	The total number of hours of flow
flowBeginHour	Hour beginning of flow (0-23)
flowEndHour	Hour ending of flow (0-23)
counterpartyName	Legal Entity Name of Counterparty
contactName	Name of counterpart's trader
contactNameId	Counterpart's traders page number. For cross reference
contactTelephoneNumber	Counterpart's traders phone number
startCalendarDate	Date of first day of flow. Represented in CFETI_DATE format CCYYMMDD.
endCalendarDate	Date of last day of flow. Represented in CFETI_DATE format CCYYMMDD.
counterpartyID	Numeric identifier of counterparty
contractType	Contract type CFETI_ENERGY_CONTRACT_NOT_SPECIFIED CFETI_ENERGY_CONTRACT_FIRM CFETI_ENERGY_CONTRACT_INTERRUPTIBLE.
TraderID	ID of trader for cross-reference
TraderName	Name of user doing trade for entity
counterpartyTraderID	ID of counterparty trader for cross-reference
PointName	Pipeline meter name
pointNumber	Pipeline meter number
totalQuantity	Total quantity of trade

quantityUnit	Enumerated value to identify quantity unit CFETI_ENERGY_QUANTITY_UNIT_DTH (Decatherm) CFETI_ENERGY_QUANTITY_UNIT_GJ (Gigajoule) CFETI_ENERGY_QUANTITY_UNIT_MW (Megawatts) CFETI_ENERGY_QUANTITY_UNIT_MMBTU (British Thermal Units) CFETI_ENERGY_QUANTITY_UNIT_BBL (Barrel) CFETI_ENERGY_QUANTITY_UNIT_TONS (Tons)
currency	ISO standard 3 character mnemonic for currency
priceType	Price type CFETI_ENERGY_PRICE_TYPE_UNSPECIFIED CFETI_ENERGY_PRICE_TYPE_FIXED CFETI_ENERGY_PRICE_TYPE_INDEXED CFETI_ENERGY_PRICE_TYPE_SWINGSWAP CFETI_ENERGY_PRICE_TYPE_BASISSWAP CFETI_ENERGY_PRICE_TYPE_FIXEDSWAP CFETI_ENERGY_PRICE_TYPE_OUTRIGHT_FORWARD
indexName	Name of index (GD = Gas Daily, IF = InsideFerc etc)
loadProfile	Load profile e.g. 5X16, 6X16,2X24 etc
holidayFlag	Boolean value used to denote whether to include Holiday (non-zero) or not (zero).
productPeriod	Enumerated value used to denote product period CFETI_ENERGY_PRODUCT_PERIOD_UNSPECIFIED CFETI_ENERGY_PRODUCT_PEAK CFETI_ENERGY_PRODUCT_OFFPEAK
shortLocation	Short description of full location name
shortIndexName	Short description of full index name
timeZone	3 character mnemonic for prevailing time zone
priceMechanism	Price mechanism (e.g. Fixed, SWAP, NGI Chicago).
indexTypeID2	Enumerated index type for swap instrument (swap trades only). Possible values are as for indexTypeID.
IndexType2	Index type name for swap instrument (swap trades only)
indexName2	Name of index for swap instrument (swap trades only)
shortIndexName2	Short description of full index name of swap instrument (swap trades only)

2.9.1.1.2 FX

When the command is `CFETI_TRADE_CONFIRM` and the `orderInfoType` has the value `CFETI_ORDERINFO_FX_TRADE` the `orderInfo` field shall be populated with a pointer to the CFETI_FX_TRADE_DESC structure shown below).

```
/*
 * CFETI_FX_TRADE_DESC: FX trade details
 */
typedef struct CFETI_FX_TRADE_DESC CFETI_FX_TRADE_DESC;
struct CFETI_FX_TRADE_DESC {
   unsigned int productType;  /**< FX Product Type (enumerated) */
   unsigned int dealType;     /**< FX Deal Type (enumerated) */
   unsigned int contractDate; /**< Trading Date CCYYMMDD */
   unsigned int deliveryDate; /**< Settlement Date CCYYMMDD */
   unsigned short legType;    /**< FX Leg Type (enumerated) */
   unsigned int cutoffTime;   /**< Cutoff time HHMMSSCC (FX Options only) */
   const char*  cutoffTimeRegion;        /**< Cutoff region (FX Options only) */
   unsigned int expiryDate;   /**< Expiry date CCYYMMDD (FX Options only) */
   double       referencePrice;          /**< Reference price */
   const char*  receivingAgent;          /**< Receiving Agent */
   const char*  beneficiary;  /**< Beneficiary */
   const char*  beneficiaryAccount;      /**< Benficiary Account */
   const char*  buyCurrency;  /**< Mnemonic of buy currency */
   double       buyAmount;    /**< Buy currency amount */
   const char*  sellCurrency; /**< Mnemonic of sell currency */
   double       sellAmount;   /**< Sell currency amount */
   const char*  currency1Buyer;          /**< Buyer of currency 1 */
   const char*  currency1Seller;         /**< Seller of currency 1 */
```

```
   double       forwardRate;  /**< Formard Rate for FX Swap, FX Forward and non-
deliverable forwards */
   const char*  currencySettlement;      /**< Mnemonic of settlement currency for non-
deliverable forwards */
   unsigned int fixingDate;   /**< Date to be used for fixing for non-deliverable
forwards */
   const char*  fixingSource; /**< Fixing source for non-deliverable forwards */
   const char*   hedgeId;     /**< Hedge id */
   unsigned char hedgeType;   /**< Enumerated hedge type */
   const char*   brokerageCurrency; /**< Brokerage currency */
 };
```

The following enumerations are used in the FX Option trade confirmation. See cfeti_consts.h for details on each value.

- Enumerated FX Product Type
- Enumerated FX Deal Type
- Enumerated FX Leg Type
- Enumerated FX Hedge Type

2.9.1.1.3 FX Options

When the command is `CFETI_TRADE_CONFIRM` and the `orderInfoType` has the value `CFETI_ORDERINFO_FX_OPTION_TRADE` the `orderInfo` field shall be populated with a pointer to the CFETI_FX_OPTION_TRADE_DESC structure shown below). The FX Option trade definition consists of the fields in an FX trade as described in 2.9.1.1.2 plus the fields specific to an FX Option trade.

```
struct CFETI_FX_OPTION_TRADE_DESC {
   CFETI_FX_TRADE_DESC  fxTrade;         /**< Core FX trade structure */
   Unsigned char  exerciseStyle;         /**< Enumerated exercise style */
   unsigned short optionClass;           /**< Enumerated option class */
   unsigned short optionStyle;           /**< Enumerated FX Option style */
   unsigned short optionStrategy;        /**< Enumerated option strategy */
   const char*    deliveryTerms;         /**< Delivery terms */
   unsigned char  hedgeType;             /**< Enumerated hedge type */
   double         lowTrigger;
   double         highTrigger;
   unsigned short lowTriggerBasis;  /**< Enumerated Amount Method */
   unsigned short highTriggerBasis; /**< Enumerated Amount Method */
   double         lowTriggerRebate;
   double         highTriggerRebate;
   const char*    triggerAgent;
   unsigned char  putCallIndicator;      /**< Enumerated put/call */
   double         strikePrice;           /**< Exercise price for option leg */
   unsigned short strikeBasis;           /**< Enumerated Amount Method */
   const char*    premiumCurrency;       /**< currency for payment of premium */
   double         premiumQuote;          /**< Premium ratio */
   unsigned short premiumQuoteBasis; /**< Terms of premium quote. Enumerated Amount
Method */
   double         premiumAmount;         /**< Amount of premium for leg 1 */
   double         volatility;
   double         spotRate;
   double         swapPoints;
   double         depositRateCurrency1;
   double         depositRateCurrency2;
};
```

The following enumerations are used in the FX Option trade confirmation. See cfeti_consts.h for details on each value.

- Enumerated Settlement Type (populated in CFETI_ORDER_DESC::settlementType)
- Enumerated FX Option Strategy
- Enumerated FX Option Class
- Enumerated FX Option Style
- Enumerated FX Option Exercise style
- Enumerated FX Put Call Indicator

- Enumerated FX Delivery Terms
- Enumerated Amount Method

2.9.1.1.4 Interest Rate Derivatives

When the command is `CFETI_TRADE_CONFIRM` and the `orderInfoType` has the value `CFETI_ORDERINFO_IRD_TRADE` the `orderInfo` field shall be populated with a pointer to the CFETI_IRD_TRADE_DESC structure shown below). Enumerations and bit-masks are identified in the descriptions of the data structures.

```
/*
 * CFETI_IRD_TRADE: IRD trade description
 * (N.B. When Used, it populates the orderInfo buffer of a CFETI_ORDER_DESC structure)
 */
typedef struct CFETI_IRD_TRADE_DESC CFETI_IRD_TRADE_DESC;
struct CFETI_IRD_TRADE_DESC {
   unsigned short productType; /**< Product type (enum:CFETI_IRD_PRODUCT_TYPE_) */
   unsigned int tradeAttributes; /**< Trade attributes (bit-mask: CFETI_IRD_ATTR_) */
   double brokerageAmount; /**< Amount of brokerage charged */
   const char* brokerageCurrency; /**< Brokerage currency */
   unsigned short contractState; /**< ContractState (enum:CFETI_CONTRACT_STATE_) */
   const char* counterparty; /**< Same as CFETI_ORDER_DESC::counterparty */
   const char* fixedCurrency;  /**< Currency for fixed interest */
   double fixedNotional; /**< Notional amount for fixed interest */
   const char* fixingDatesHolidayCenters; /**< Fixing date holiday calendar */
   int fixingDatesOffset; /**< Fixing date offset */
   unsigned short floatingBasis; /**< Fixed interest payment calc basis
(enum:CFETI_BASIS_) */
       unsigned short floatingRateIndex; /**< Floating rate index
(enum:CFETI_IRD_FLOATING_RATE_INDEX_) */
   unsigned short masterAgreement; /**< Master agreement
(enum:CFETI_MASTER_AGGREEMENT_) */
   const char* myEntity; /**< Entity under which user trades */
   const char* paymentDatesHolidayCenters; /**< Payment date holiday calendar */
   unsigned int paymentDatesOffset; /**< Offset from end of calculation period */
   CFETI_IRD_TRADE_LEG_DESC tradeLeg;/**< IRD specific trade info (index by
productType) */
};

/*
 * CFETI_IRD_TRADE_DESC: IRD Trade Leg description
 */
typedef union CFETI_IRD_TRADE_LEG_DESC CFETI_IRD_TRADE_LEG_DESC;
union CFETI_IRD_TRADE_LEG_DESC
{
   CFETI_IRD_IRS_TRADE_DESC              irs;
   CFETI_IRD_OIS_TRADE_DESC              ois;
   CFETI_IRD_FRA_TRADE_DESC              fra;
   CFETI_IRD_SWAPTION_TRADE_DESC         swaption;
   CFETI_IRD_CAPSFLOORS_TRADE_DESC       capsfloors;
};

/*
 * CFETI_IRD_TRADE: Interest Rate Swaps
 */
typedef struct CFETI_IRD_IRS_TRADE_DESC CFETI_IRD_IRS_TRADE_DESC;
struct CFETI_IRD_IRS_TRADE_DESC {
   const char* bond1; /**< First bond for USD IRS traded as spread */
   double bond1Price; /**< Price of first bond */
   unsigned int bond1PriceType; /**< Price Type of first bond */
   double bond1Qty; /**< Quantity of first bond */
   const char* bond2; /**< Second bond for USD IRS traded as spread */
   double bond2Price; /**< Price of second bond */
   unsigned int bond2PriceType; /**< Price Type of second bond */
   double bond2Qty; /**< Quantity of second bond */
   unsigned char breakAtUnit; /**< When break occurs (CFETI_INTERVAL_UNIT_) */
   unsigned short breakAtQty; /**< Number of above units */
   unsigned char breakEveryUnit; /**< Defines break occurrence (if optional,
CFETI_INTERVAL_UNIT_) */
   unsigned short breakEveryQty; /**< Number of above units */
   unsigned short compoundingMethod; /**< Compounding method
(enum:CFETI_COMPOUNDING_METHOD_) */
```

```
   unsigned int endDate; /**< End date (CCYYMMDD) */
   double firstFixingRate; /**< First floating rate fixing (%) */
   unsigned short fixedBasis;  /**< Fixed interest payment calc basis
(enum:CFETI_BASIS_) */
   unsigned short fixedConvention; /**< Convention for payment date
(enum:CFETI_PAYMENT_CONVENTION_) */
   unsigned char  fixedPaymentFrequencyUnit; /**< Fixed interest payment interval,
(enum:CFETI_INTERVAL_UNIT_) */
   unsigned short fixedPaymentFrequencyQty; /**< Number of above units */
   double fixedRate; /**< Fixed interest rate (%) */
   unsigned short floatingConvention; /**< Convention for payment date
(enum:CFETI_PAYMENT_CONVENTION_) */
   const char* floatingCurrency; /**< Currency for floating interest payments */
   double floatingNotional; /**< Notional for calc of floating payments */
   unsigned char floatingPaymentFrequencyUnit; /**< Floating interest payment interval
(enum:CFETI_INTERVAL_UNIT_) */
   unsigned short floatingPaymentFrequencyQty; /**< Number of above units */
   unsigned char floatingRollFrequencyUnit; /**< Floating rate index fixing interval
(enum:CFETI_INTERVAL_UNIT_)*/
   unsigned short floatingRollFrequencyQty; /**< Number of above units */
   const char* rollDatesHolidayCenters; /**< Roll dates holiday calendar */
   unsigned int rolls; /**< Day of month that floating rate is fixed */
   double spreadOverFloating; /**< +ve/-ve spread in basis points */
   unsigned int startDate; /**< Start date (CCYYMMDD) */
   double swapSpread; /**< Swap spread (basis points) */
};

/*
 * CFETI_IRD_TRADE: Overnight Index Swaps
 */
typedef struct CFETI_IRD_OIS_TRADE_DESC CFETI_IRD_OIS_TRADE_DESC;
struct CFETI_IRD_OIS_TRADE_DESC {
   unsigned int endDate; /**< End date (CCYYMMDD) */
   unsigned short fixedBasis; /**< Fixed interest payment calc basis
(enum:CFETI_BASIS_) */
   unsigned short fixedConvention; /**< Convention for payment date
(enum:CFETI_PAYMENT_CONVENTION_) */
   unsigned char fixedPaymentFrequencyUnit; /**< Fixed interest payment interval
(enum:CFETI_INTERVAL_UNIT_) */
   unsigned short fixedPaymentFrequencyQty; /**< Number of above units */
   double fixedRate; /**< Fixed interest rate (%) */
   unsigned short floatingConvention; /**< Convention for payment date
(enum:CFETI_PAYMENT_CONVENTION_) */
   const char* floatingCurrency; /**< Currency for floating interest payments */
   double floatingNotional; /**< Notional for calc of floating payments */
   unsigned char floatingPaymentFrequencyUnit; /**< Payment interval
(enum:CFETI_INTERVAL_UNIT_)*/
   unsigned short floatingPaymentFrequencyQty; /**< Number of above units */
   unsigned char floatingRollFrequencyUnit; /**< Floating rate index fixing interval
(enum:CFETI_INTERVAL_UNIT_) */
   unsigned short floatingRollFrequencyQty; /**< Number of above units */
   const char* rollDatesHolidayCenters; /**< Roll dates holiday calendar */
   unsigned int rolls; /**< Day of month that floating rate is fixed */
   unsigned int startDate; /**< Start date (CCYYMMDD) */
};

/*
 * CFETI_IRD_TRADE: Fixed Rate Agreements
 */
typedef struct CFETI_IRD_FRA_TRADE_DESC CFETI_IRD_FRA_TRADE_DESC;
struct CFETI_IRD_FRA_TRADE_DESC {
   unsigned int calcPeriodDays; /**< Num days from value date to maturity date */
   double fixedRate; /**< Fixed interest rate (%) */
   unsigned int fixingDate; /**< Fixing date (CCYYMMDD) */
   unsigned char indexTenor1Unit; /**< 1st index tenor (enum:CFETI_INTERVAL_UNIT_) */
   unsigned short indexTenor1Qty; /**< Number of above units */
   unsigned char  indexTenor2Unit; /**< 2nd index tenor (enum:CFETI_INTERVAL_UNIT_) */
   unsigned short indexTenor2Qty; /**< Number of above units */
   unsigned int matDate; /**< End date of FRA (CCYYMMDD) */
   unsigned int matDateTenor; /**< Months to end of FRA */
   unsigned char paymentDateConvention; /**< Convention for payment date
(enum:CFETI_PAYMENT_CONVENTION_) */
   unsigned int valueDate; /**< Value date (CCYYMMDD) */
   unsigned int valueDateTenor; /**< Months to start of FRA */
};

/*
```

```
 * CFETI_IRD_TRADE: Swaptions
 */
typedef struct CFETI_IRD_SWAPTION_TRADE_DESC CFETI_IRD_SWAPTION_TRADE_DESC;
struct CFETI_IRD_SWAPTION_TRADE_DESC {
   unsigned char breakAtUnit; /**< Specifies when break occurs
(enum:CFETI_INTERVAL_UNIT_)*/
   unsigned short breakAtQty; /**< Number of above units */
   unsigned char  breakEveryUnit; /**< Defines break occurrence (if optional,
enum:CFETI_INTERVAL_UNIT_) */
   unsigned short breakEveryQty; /**< Number of above units */
   unsigned short calcAgent; /**< Calculation agent (enum:CFETI_CALCULATION_AGENT_) */
   unsigned short compoundingMethod; /**< Compounding method
(enum:CFETI_COMPOUNDING_METHOD_) */
   unsigned int earliestTime; /**< Earliest time to exercise swaption on expiry date
*/
   unsigned int endDate; /**< End date (CCYYMMDD) */
   const char* exerciseHolidayCenters; /**< Holiday calendar name */
   const char* exerciseLocation; /**< Physical location for expiry of SWAPTION */
   unsigned int expiryDate; /**< Date on which SWAPTION expires (CCYYMMDD) */
   unsigned int expiryTime; /**< Time at which SWAPTION expires (HHMMSSCC) */
   unsigned short fixedBasis; /**< Fixed interest payment calc basis
(enum:CFETI_BASIS_) */
   unsigned short fixedConvention; /**< Convention for payment date
(enum:CFETI_PAYMENT_CONVENTION) */
   unsigned char fixedPaymentFrequencyUnit; /**< Fixed interest payment interval
(enum:CFETI_INTERVAL_UNIT_) */
   unsigned short fixedPaymentFrequencyQty; /**< Number of above units */
   double fixedRate; /**< Fixed interest rate (%) */
   unsigned short floatingConvention; /**< Convention for payment date
(enum:CFETI_PAYMENT_CONVENTION) */
   const char* floatingCurrency; /**< Currency for floating interest payments */
   double floatingNotional; /**< Notional for calc of floating payments */
   unsigned char floatingPaymentFrequencyUnit; /**< Floating interest payment interval
(enum:CFETI_INTERVAL_UNIT_)*/
   unsigned short floatingPaymentFrequencyQty; /**< Number of above units */
   unsigned char floatingRollFrequencyUnit; /**< Floating rate index fixing interval
(enum:CFETI_INTERVAL_UNIT_)*/
   unsigned short floatingRollFrequencyQty; /**< Number of above units */
   double notional; /**< Notional amount on the SWAPTION */
   unsigned char optionStyle; /**< Exercise style (enum) */
   double premiumAmount; /**< Amount to be paid from buyer to seller */
   double premiumBasis; /**< Option premium in basis points */
   const char* premiumCpty; /**< Seller of the option */
   const char* premiumPayer; /**< Buyer of the option */
   unsigned int premiumPaymentDate; /**< Date for premium payment (CCYYMMDD) */
   const char* rollDatesHolidayCenters; /**< Roll dates holiday calendar */
   unsigned int rolls; /**< Day of month that floating rate is fixed */
   unsigned short subType; /**< Sub-type (enum:CFETI_IRD_SUB_TYPE_) */
   struct
   {
                  unsigned short deliveryTerms; /**< Cash or Physical
(enum:CFETI_DELIVERY_TERMS_) */
                  const char* currency; /**< Settlement cash currency */
                  unsigned short cashMethod; /**< Payment method for cash */
                  unsigned char paymentOffset; /**< Payment offset */
                  unsigned short quotationRate; /**< Quotation rate
(enum:CFETI_QUOTATION_RATE_) */
                  const char* rateSource; /**< Source of settlement rate */
                  const char* banks; /**< Reference banks */
                  unsigned char valuationOffset; /**< Valuation offset */
                  unsigned int valuationTime; /**< (HHSSMMCC) */
   } settlement;
   double spreadOverFloating; /**< +ve/-ve spread in basis points */
   unsigned int startDate; /**< Start date (CCYYMMDD) */
   double strikePrice; /**< Strike price of the option */
};

/*
 * CFETI_IRD_TRADE: Caps/Floors
 */
typedef struct CFETI_IRD_CAPSFLOORS_TRADE_DESC CFETI_IRD_CAPSFLOORS_TRADE_DESC;
struct CFETI_IRD_CAPSFLOORS_TRADE_DESC {
   unsigned short basis; /**< Net payment calculation basis (enum:CFETI_BASIS_) */
   unsigned int endDate; /**< End date (CCYYMMDD) */
   unsigned short floatingConvention; /**< Convention for payment date
(enum:CFETI_PAYMENT_CONVENTION_) */
```

```
   unsigned char floatingPaymentFrequencyUnit; /**< Floating interest payment interval
(prefix CFETI_INTERVAL_UNIT_)*/
   unsigned short floatingPaymentFrequencyQty; /**< Number of above units */
   unsigned char floatingRateIndexTenorUnit; /**< Floating rate index tenor (prefix
CFETI_INTERVAL_UNIT_) */
   unsigned short floatingRateIndexTenorQty; /**< Number of above units */
   unsigned char floatingRollFrequencyUnit; /**< Floating rate index fixing interval
(prefix CFETI_INTERVAL_UNIT_)*/
   unsigned short floatingRollFrequencyQty; /**< Number of above units */
   double premiumAmount; /**< Amount to be paid from buyer to seller */
   double premiumBasis; /**< Option premium in basis points */
   const char* premiumCpty; /**< Seller of the option */
   const char* premiumPayer;  /**< Buyer of the option */
   unsigned int premiumPaymentDate; /**< Date for premium payment (CCYYMMDD) */
   const char* rollDatesHolidayCenters; /**< Roll dates holiday calendar */
   unsigned int rolls; /**< Day of month that floating rate is fixed */
   double spreadOverFloating; /**< +ve/-ve spread in basis points */
   unsigned int startDate; /**< Start date (CCYYMMDD) */
   double strikePrice; /**< Strike price of the option */
   unsigned short subType; /**< Sub-type (enum:CFETI_IRD_SUB_TYPE_) */
};
```

2.9.1.1.5 Treasury Swaps and IRS vs. Futures

```
/** OrderInfo structure for Treasury Swaps or IRS vs Futures*/
typedef struct CFETI_TSWAP_DESC CFETI_TSWAP_DESC;
typedef struct CFETI_TSWAP_DESC CFETI_PV01_LOCK_DESC;
struct CFETI_TSWAP_DESC {
     double    dTSwapRatio;    /**< User's selected TSWAP ratio or IRS PV01 value
*/
     double    dLockPrice;     /**< User's selected lock price */
     double    dCurrentTSwapRatio; /**< Not currently used */
     double    dCurrentLockPrice;  /**< Not currently used */
};
```

This structure is used as order info in both CFETI_ORDER_DESC and CFETI_MARKET_DESC.

2.9.2 Command Preferences

The following preferences can be set to control the behavior of the order being submitted. If no preferences are set (i.e. `cmdPreferences` is zero), the default action is for eSpeed API to submit the order as it is specified with the size being interpreted as total. If the stated preference cannot be provided by the trading system or if the specified combination of preferences is invalid, the command response will be a rejection of the submitted order and the reason will be delivered in the command status. Not all of the preferences listed are supported by all trading systems.

`CFETI_ORDER_COMPLETE_FILL_ONLY`	Request that order be filled in its entirety or not at all.
`CFETI_TRADE_AT_MARKET_PRICE`	Request that order be filled at prevailing market price (i.e. the price field in the supplied order is ignored).
`CFETI_TRADE_AT_MARKET_SIZE`	Request that order be filled up to users single order limit or the total available to be traded, whichever is the lower. (i.e. the size field in the supplied order is ignored).
`CFETI_ORDER_SIZE_IS_TOTAL`	The order size is the total size (use to override any existing size for an already queued order).
`CFETI_ORDER_SIZE_IS_INCREMENTAL`	The order size is to be added to any existing size for an already posted order.
`CFETI_TRADE_OPTION_1`	Generic trade option that is given meaning according to the instrument being traded.
`CFETI_TRADE_OPTION_2`	As `CFETI_TRADE_OPTION_1`

`CFETI_TRADE_OPTION_3`	As `CFETI_TRADE_OPTION_1`
`GOOD_UNTIL_TIME`	Order is valid for the number of minutes specified in the timeOffset field in the order structure. The time starts from receipt of the order by the eSpeed system.
`CFETI_ONLY_AT_BEST`	Only at best orders are submitted without a price. Instead the price shall be determined by the eSpeed system upon receipt of the order. The assigned price will then not subsequently change for the life of that order.

The following preferences can be set to control the behavior of the market being posted in the `preferences2` field of the market structure

`CFETI_USE_RESERVE_SIZE`	Market or order is specified with a minimum and maximum reserve size and a total size for reserve.

2.9.2.1 European Repos

For European Repos the trade options are used to indicate the mechanisms that can be used to clear the trade if the order is executed. The options can be combined to indicate that clearing can be through more than one mechanism.

`CFETI_TRADE_OPTION_1`	`CFETI_TRADE_CLEARING_LCH`
`CFETI_TRADE_OPTION_2`	`CFETI_TRADE_CLEARING_CLEARNET`
`CFETI_TRADE_OPTION_3`	`CFETI_TRADE_CLEARING_INTERBANK`

2.9.2.2 Interest Rate Derivatives

For Interest Rate Derivatives the first and third trade options are used to indicate the mechanisms that can be used to clear the trade if the order is executed. These options can be combined to indicate that clearing can be through more than one mechanism.

`CFETI_TRADE_OPTION_1`	`CFETI_TRADE_CLEARING_LCH`
`CFETI_TRADE_OPTION_3`	`CFETI_TRADE_CLEARING_INTERBANK`

2.9.3 Return Codes

A successful return code, `CFETI_SUCCESS`, indicates that order request has been successfully processed. Otherwise, the reason for the failure is indicated in the return code. Return codes that may be delivered to the application include:

`CFETI_SUCCESS`	The request was processed successfully. Whether or not the order was successfully queued by the trading system is returned in the `tradingSysCallback` specified when the original connection request was made.
`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETIPostMessage`.
`CFETI_NO_SESSION`	An attempt is made to submit an order without having previously established a session with the eSpeed Session Manager.
`CFETI_NO_SUCH_LOGIN`	The login identified by the supplied session identifier cannot be found.
`CFETI_NO_SUCH_CONNECTION`	The connection specified by the supplied trading system

session identifier cannot be found.

2.9.4 Command Response

All buy and sell requests will be acknowledged upon receipt as queued or rejected. Valid buy or sell requests will be queued for execution. If queued, the acknowledgement will indicate the order details submitted. If rejected the acknowledgment will indicate the reason why the order was not processed by the system. Each discrete order request is processed independently.

Command Response	Command Details
Order Queued	trade instrument, price, size and side
Order Rejected	Possible Reasons: • Invalid authentication credentials. • No market • Invalid order • Trading suspended (New trades cannot be initiated – existing markets and orders can be modified or cancelled). • Transaction not approved • Instrument not currently trade-able • Action prohibited (e.g. bid disabled) • No credit available with counterparty whose price the order was submitted against (name-giveup business units only) • State error – new buy/sell order cannot be accepted • Order was submitted using max display feature but one or more of the initial, minimum or maximum reserve size specified in the order was invalid.
Order Moved	Only applies to API users with the OrderAsMarket or MarketAsOrder preference set. This command indicates that: • The order will be executed at a better price than that entered, or • The order cannot be matched and is being converted to a market Depending on which status above applies, this response will be followed by an Order Queued, Order Rejected, Market Created or Market Rejected message.

All responses related to the submitted order are returned in the callback routine provided in the initial `CFETIConnect` call. The callback routine is invoked as follows:

```
void tradeSysCallback(
                                CFETI_CMD cmd,
                                CFETI_CMD_STATUS cmdStatus,
                                CFETI_CMDDATA cmdData,
                                CFETI_UD userData );
```

Commands that may be issued by the eSpeed API to the application as a result of the submitted order request are as follows:

Command	Command Status	Command Data Type
`CFETI_ORDER_QUEUED`	`CFETI_SUCCESS`	`CFETI_ORDER`

CFETI_ORDER_REJECTED	CFETI_UNKNOWN_TRD_INST CFETI_INVALID_AUTH CFETI_NO_MARKET CFETI_INVALID_ORDER CFETI_INSUFFICIENT_TIME CFETI_TRANSACTION_NOT_APPROVED CFETI_TRADING_SUSPENDED CFETI_INSTRUMENT_NOT_TRADEABLE CFETI_ACTION_PROHIBITED CFETI_NO_CREDIT CFETI_ORDER_STATE_ERROR CFETI_INVALID_RESERVE_SIZE	CFETI_ORDER
CFETI_ORDER_MOVED	CFETI_PRICE_MOVED CFETI_ORDER_CHANGED_TO_MARKET	CFETI_ORDER

Note that the `tradeId` element of the `CFETI_ORDER` passed to the trading system callback for these commands is undefined.

2.9.5 Individual Notification

Queued orders will receive subsequent messages as the order is filled. Each Order execution contains incremental details on the state of the order. Upon completion of the transaction a Trade Confirmation will be received, which summarizes the details of the order, including a unique trade identifier that can be used to track orders and in subsequent queries, as well as the time of the trade and the amount actually traded. If any part of the order remains after trading is completed (i.e. size exceeds size done in trade confirmation), and if the trading system provides the facility to do so, a new market shall be introduced by the system. This will be at the traded price and for either the portion of the order that was not filled, or for a base unit size. (The actual size selected will depend upon agreement between the customer and eSpeed, LLC., prior to commencement of electronic trading).

If an order will execute but the execution price is not yet known notification will be received of the executing order. The command status code shall be one of CFETI_MATCHED (Customer us guaranteed to trade from an identified match not yet executed), CFETI_MATCHED_BETTER_FILL (Customer is guaranteed to trade from an identified match not yet executed by trading through the stack) or CFETI_SEEKING_BETTER_FILL (Customer is trying to trade through the stack behind a trade through the stack already in front of them).

Order Responses	Order Details
Order Cancelled	trade instrument, price, size cancelled, order indicator Possible Reasons: • Cancelled by system • Trade cleared • Transaction now disapproved by eSpeed
Order Executing	The posted order will be executed at the specified price or at a better price. The actual price that the user will get filled at is not known at the time this message is created and delivered.
Order Executed	The posted order is executed at the specified price or at a better price.
Trade Confirmation	trade identifier, trade instrument, price, total size, order indicator
Market Created	trade instrument, price, size, side

Individual notifications are delivered in the callback routine provided in the initial `CFETIConnect` call. The commands that may be issued by the eSpeed API to the application are as follows:

Command	Command Status	Command Data Type
CFETI_ORDER_EXECUTED	CFETI_SUCCESS CFETI_CHECK_CREDIT	CFETI_ORDER
CFETI_ORDER_EXECUTING	CFETI_MATCHED CFETI_MATCHED_BETTER_FILL CFETI_SEEKING_BETTER_FILL	CFETI_ORDER
CFETI_TRADE_PENDING	CFETI_SUCCESS	CFETI_ORDER
CFETI_TRADE_CONFIRM	CFETI_SUCCESS	CFETI_ORDER
CFETI_ORDER_CANCELLED	CFETI_MKT_CLEARED CFETI_ORDER_CANCELLED_BY_SYSTEM CFETI_TRANSACTION_DISAPPROVED CFETI_MARKET_TO_FOLLOW CFETI_ORDER_EXPIRED	CFETI_ORDER
CFETI_MKT_CREATED	CFETI_MKT_CLEARED	CFETI_MARKET

If an order cancelled notification is delivered with a market-to-follow command status code then this indicates to the client application that a subsequent market created notification will be delivered.

The command status value CFETI_ORDER_EXPIRED is delivered for orders specified with the good-until-time preference when the order has expired.

The command status value CFETI_CHECK_CREDIT may be set when the order is executed in a giveup-enabled business. This indicates that a manual credit check will be necessary before the trade can be confirmed.

If when trading is complete a manual process must be applied to the trade before it is confirmed (e.g. a credit check for a giveup-enabled business) the command CFETI_TRADE_PENDING shall be delivered to the application. When the trade is subsequently confirmed a trade confirmation shall be delivered. For some businesses a CFETI_TRADE_REJECTED command may be delivered to cancel the trade.

2.9.6 Global Notification

As orders are processed, global notifications will be distributed to all users that are subscribed to the instrument affected to indicate that trading activity using the market data update mechanism (See 2.12 - Subscribe/Unsubscribe). The update will include the necessary components to indicate the new trade - trade instrument, price and trade indicator (BUY/SELL) - as well as updates to the relevant bid or offer participant lists.

Market Response	Market Details
Market Data Update	trade instrument, price and trade indicator.

2.9.7 Credit View Modification

For name-giveup businesses, an order may be rejected if there is no credit available with the counterparty whose price the order was submitted against. In this case, the individual notification delivered to the application will have a command `CFETI_ORDER_REJECTED` and a command status `CFETI_NO_CREDIT`. The `rejectionId` field in the order (`CFETI_ORDER`) will be populated (See 2.9.1).

A facility within the API enables a user's view of the Giveup Matrix (for that business unit) to be modified such that prices from rejected counterparties can be identified as not available for trading. In order to modify the credit view, the `CFETIPostMessage` interface must be invoked.

```
CFETI_RC CFETIPostMessage(
                        CFETI_SESSION_ID sessId,
                        CFETI_TRADE_SESS_ID trdSysSessId,
```

```
                                CFETI_CMD cmd,
                                CFETI_CMDDATA cmdData,
                                CFETI_CMDPREF cmdPreferences );
```

`SessId`	Valid session identifier from previously successful login.
`TrdSysSessId`	Trading system session identifier returned on successful connection.
`Cmd`	Command (request) being submitted. To modify the credit view this shall be `CFETC_MODIFY_CREDIT_VIEW`.
`CmdData`	Command data that contains the details of the request. In the case of credit view modification the `rejectionId` in the order is expected as the command data. If the `rejectionId` supplied is 0 (zero), all previous modifications are cleared.
`CmdPreferences`	This argument is not required.

After modifying the credit view, the `CFETIDecodeDataField` interface must be called for encoded eSpeed API fields delivered to the application to re-evaluate the prices that are available for trading.

2.10 Cancel Buy/Sell

User may submit requests to cancel previously submitted orders. Cancels submitted must contain the order details being cancelled. Cancellations will only be processed if the order has not yet been executed. If accepted the acknowledgment will contain a summary of what was cancelled and executed. Users should note that Order Executions and Trade Confirmation would still be received for portions of the order, which were executed prior to the cancellation request.

```
        CFETI_RC CFETIPostMessage(
                                CFETI_SESSION_ID sessId,
                                CFETI_TRADE_SESS_ID trdSysSessId,
                                CFETI_CMD cmd,
                                CFETI_CMDDATA cmdData,
                                CFETI_CMDPREF cmdPreferences );
```

`sessId`	Valid session identifier from previously successful login.
`trdSysSessId`	Trading system session identifier returned on successful connection.
`cmd`	Command (request) being submitted. To cancel an order this shall be `CFETC_CANCEL_ORDER`.
`cmdData`	Command data that contains the details of the request. In the case of a cancel buy or cancel sell a `CFETI_ORDER` shall be expected as the command data.
`cmdPreferences`	User preferred settings that can be specified to control the behavior of the order cancellation. (E.g. cancel all orders for the given trade instrument).

2.10.1 Command Preferences

The following preferences can be set to control the behavior of the order cancellation. If no preferences are set (i.e. `cmdPreferences` is zero) the default action is for eSpeed API to cancel the specified order only. If the stated preference cannot be provided by the trading system or if the specified combination of preferences is invalid, the command response will be a rejection of the order cancellation request and the reason will be delivered in the command status. These preferences should be combined with any returned in the order structure when the order was accepted by the trading system.

`CFETI_ORDER_CANCEL_ALL_FOR_ISSUE`	Cancel all of this users orders for the specified instrument.

2.10.2 Return Codes

A successful return code, `CFETI_SUCCESS`, indicates that the order cancellation request has been successfully processed. Otherwise, the reason for the failure is indicated in the return code. Return codes that may be delivered to the application include:

`CFETI_SUCCESS`	The request was processed successfully. Whether or not the order cancellation was successfully queued by the trading system is returned in the `tradingSysCallback` specified when the original connection request was made.
`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETIPostMessage`.
`CFETI_NO_SESSION`	An attempt is made to cancel an order without having previously established a session with the eSpeed Session Manager.
`CFETI_NO_SUCH_LOGIN`	The login identified by the supplied session identifier cannot be found.
`CFETI_NO_SUCH_CONNECTION`	The connection specified by the supplied trading system session identifier cannot be found.

2.10.3 Command Responses

Responses to an order cancellation request are as follows:

Command Response	Command Details
Order Cancel Accepted	Order cancel command successfully received by system for processing(trade instrument, price, side and total size)
Order Cancel Rejected	Order Cancel Unsuccessful (trade instrument, price, size and side) Possible Reasons: • Invalid authentication signature • Order not queued. (I.e. Insufficient price and/or size.) • Minimum Order time not exceeded. • Invalid id
Order Cancelled	Order cancelled successfully(trade instrument, price, side and total size)
Order Cancel Queued	If a request to cancel is received that cannot be acted upon immediately an acknowledgement of the request to cancel is delivered. The request may subsequently either be accepted or rejected.

All responses related to the order cancellation request are returned in the callback routine provided in the initial call to `CFETIConnect`. The callback routine is invoked as follows:

```
void tradeSysCallback(
                                CFETI_CMD cmd,
                                CFETI_CMD_STATUS cmdStatus,
                                CFETI_CMDDATA cmdData,
                                CFETI_UD userData );
```

Commands that may be issued by the eSpeed API to the application as a result of the submitted order cancellation request are as follows:

Command	Command Status	Command Data Type
CFETI_CANCEL_ORDER_ACCEPTED	CFETI_SUCCESS	CFETI_ORDER
CFETI_CANCEL_ORDER_REJECTED	CFETI_UNKNOWN_TRD_INST CFETI_INVALID_AUTH CFETI_INVALID_ORDER CFETI_INSUFFICIENT_TIME CFETI_INVALID_ID	CFETI_ORDER CFETI_ORDER CFETI_ORDER CFETI_ORDER
CFETI_ORDER_CANCELLED	CFETI_SUCCESS CFETI_SUBSTITUTED	CFETI_ORDER
CFETI_ORDER_CANCEL_QUEUED	CFETI_ATTEMPTING_SUBSTITUTION	CFETI_ORDER

An order cancellation request will be acknowledged by the system with the delivery of a Cancel Order Accepted/Rejected message. In the case of a Cancel Order Accepted message this is an acknowledgement that the cancel command has been received and is being processed by the system, not an indication that the order has been cancelled. Only upon receipt of an Order Cancelled message has the market been removed from the system.

As with market cancellations, an order cancellation will be considered successful if only a portion of the remaining order is removed from the trading system. The order returned will indicate the portion of the order that was cancelled. Where possible the system will indicate the portion of the order that was executed as well as cancelled.

2.11 Refresh

When a user successfully connects to a trading system a refresh of the markets submitted and orders queued by that user that remain on the trading system at the time of connection is delivered to the connection callback. A facility is provided to enable the user to request a further refresh at other times.

```
CFETI_RC CFETIPostMessage(
                        CFETI_SESSION_ID sessId,
                        CFETI_TRADE_SESS_ID trdSysSessId,
                        CFETI_CMD cmd,
                        CFETI_CMDDATA cmdData,
                        CFETI_CMDPREF cmdPreferences );
```

`sessId`	Valid session identifier from previously successful login.
`trdSysSessId`	Trading system session identifier returned on successful connection.
`cmd`	Command (request) being submitted. To refresh markets and orders the command shall be `CFETC_REFRESH`.
`cmdData`	N/A

`cmdPreferences`	N/A

2.11.1 Return Codes

A successful return code, `CFETI_SUCCESS`, indicates that the refresh request has been successfully processed. Otherwise, the reason for the failure is indicated in the return code. Return codes that may be delivered to the application include:

`CFETI_SUCCESS`	The request was processed successfully.
`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETIPostMessage`.
`CFETI_NO_SESSION`	An attempt is made to refresh without having previously established a session with the CFETI Session Manager.
`CFETI_NO_SUCH_LOGIN`	The login identified by the supplied session identifier cannot be found.
`CFETI_NO_SUCH_CONNECTION`	The connection specified by the supplied trading system session identifier cannot be found.

2.11.2 Command Response

A refresh command response shall be delivered to the application for each market and order that currently exists on the trading system for the user indicated by the request. A notification is always sent when the refresh is complete.

Command Response	Command Details
Market Refresh	Market details (trade instrument, price, size and side)
Order Refresh	Order details (trade instrument, price, size and indicator)
Refresh Complete	Status to indicate success / failure of completed refresh

The responses related to the refresh request are returned in the callback routine provided in the initial `CFETIConnect` call. The callback routine is invoked as follows:

```
void tradeSysCallback(
                                CFETI_CMD cmd,
                                CFETI_CMD_STATUS cmdStatus,
                                CFETI_CMDDATA cmdData,
                                CFETI_UD userData );
```

Commands that may be issued by the eSpeed API to the application as a result of the requested refresh are as listed below.

Command	Command Status	Command Data Type
`CFETI_MKT_REFRESH`	`CFETI_SUCCESS`	`CFETI_MARKET`
`CFETI_ORDER_REFRESH`	`CFETI_SUCCESS`	`CFETI_ORDER`
`CFETI_REFRESH_COMPLETE`	`CFETI_SUCCESS` `CFETI_REFRESH_FAILED`	`N/A`

2.12 Subscribe/Unsubscribe

eSpeed API applications must subscribe for each of the instruments they wish to receive instrument details and market updates. The eSpeed API does not automatically subscribe to any instrument that the application posts a market or order request. It is therefore incumbent upon the application to subscribe to market data for instruments as required. Subscriptions can be submitted using the `CFETIPostMessage` routine. Upon successful subscription, the eSpeed API will return to the application the current state of the instrument to the connection callback. Subsequent market data updates that affect this instrument will also be delivered to the connection callback.

Applications that no longer require interest in an instrument must unsubscribe using the routine `CFETIPostMessage`.

```
CFETI_RC CFETIPostMessage(
                          CFETI_SESSION_ID sessId,
                          CFETI_TRADE_SESS_ID trdSysSessId,
                          CFETI_CMD cmd,
                          CFETI_CMDDATA cmdData,
                          CFETI_CMDPREF cmdPreferences );
```

`sessId`	Valid session identifier from previously successful login.
`trdSysSessId`	Trading system session identifier returned on successful connection.
`cmd`	Command (request) being submitted. To subscribe to an instrument this shall be `CFETC_SUBSCRIBE`. To cancel a subscription this shall be `CFETC_UNSUBSCRIBE`.
`cmdData`	Command data that contains the details of the request. In the case of an instrument subscription or cancellation of an instrument subscription a `CFETI_INSTRUMENT` shall be expected as the command data.
`cmdPreferences`	N/A

2.12.1 Return Codes

A successful return code, `CFETI_SUCCESS`, indicates that the subscription request has been successfully processed. Otherwise, the reason for the failure is indicated in the return code. Return codes that may be delivered to the application include:

`CFETI_SUCCESS`	The request was processed successfully. Whether or not the subscription or cancellation of a subscription was successful is returned in the `tradingSysCallback` specified when the original connection request was made.
`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETIPostMessage`.
`CFETI_NO_SESSION`	An attempt is made to subscribe or cancel a subscription to market data without having previously established a session with the eSpeed Session Manager.
`CFETI_NO_SUCH_LOGIN`	The login identified by the supplied session identifier cannot be found.
`CFETI_NO_SUCH_CONNECTION`	The connection specified by the supplied trading system session identifier cannot be found.

2.12.2 Command Response

Responses to a subscription or cancel subscription request are as follows:

Command Response	Command Details
Subscribe Accepted	Instrument record name and data.
Subscribe Rejected	Possible Reasons: • Invalid authentication credentials. • Unauthorized request • Instrument Unknown
Unsubscribe Accepted	Instrument record name
Unsubscribe Rejected	Possible Reasons: • Invalid authentication credentials. • Unauthorized request • Instrument not subscribed.
Market Data Update	Instrument record name and data
Subscription status	Status message associated with the market data flow. Possible reasons: • Instrument lost • Instrument restored • Market data feed is down • Market data feed is restored • Trading sub-system off-line • Trading sub-system on-line

All responses related to the subscription (or cancel subscription) request as well as any subsequent market data updates are returned in the callback routine provided in the initial call to `CFETIConnect`. The callback routine is invoked as follows:

```
void tradeSysCallback(
                                CFETI_CMD cmd,
                                CFETI_CMD_STATUS cmdStatus,
                                CFETI_CMDDATA cmdData,
                                CFETI_UD userData );
```

Commands that may be issued by the eSpeed API to the application as a result of the subscription of subscription cancellation request are:

Command	Command Status	Command Data Type
CFETI_SUBSCRIBE_ACCEPTED	CFETI_SUCCESS	CFETI_INSTRUMENT_DATA
CFETI_SUBSCRIBE_REJECTED	CFETI_UNKNOWN_TRD_INST CFETI_INVALID_AUTH	CFETI_INSTRUMENT
CFETI_UNSUBSCRIBE_ACCEPTED	CFETI_SUCCESS	CFETI_INSTRUMENT
CFETI_UNSUBSCRIBE_REJECTED	CFETI_UNKNOWN_TRD_INST CFETI_TRD_INST_NOTSUBSCRIBED CFETI_INVALID_AUTH	CFETI_INSTRUMENT
CFETI_UPDATE	CFETI_SUCCESS	CFETI_INSTRUMENT_DATA
CFETI_SUBSCRIBE_STATUS	CFETI_INSTRUMENT_LOST CFETI_INSTRUMENT_RESTORED	CFETI_INSTRUMENT
CFETI_SUBSCRIBE_STATUS	CFETI_MARKET_FEED_DOWN CFETI_MARKET_FEED_RESTORED	CFETI_TRADING_SYSTEM
CFETI_SUBSCRIBE_STATUS	CFETI_TRADE_SUBSYSTEM_OFFLINE CFETI_TRADE_SUBSYSTEM_ONLINE	CFETI_TRADING_SUBSYSTEM

2.12.3 Instrument Data Definition

The instrument data structure and associated data types are defined as follows:

```
typedef char* CFETI_INSTRUMENT;
typedef unsigned int CFETI_TRADING_SYSTEM;
typedef struct CFETI_FIELD_DESC CFETI_FIELD_DESC;
typedef           struct           CFETI_INSTRUMENT_DATA_DESC
                      CFETI_INSTRUMENT_DATA_DESC;
typedef union CFETI_FIELD_VALUE CFETI_FIELD_VALUE;

union CFETI_FIELD_VALUE {
    char int8;
    unsigned char byte;
    short int16;
    unsigned short uint16;
    int int32;
    unsigned int uint32;
    double decimal;
    time_t dateTime;
    struct {
        unsigned short len;
        char *bp;
    } buffer;
    struct {
        unsigned short len;
        unsigned short fields;
        char *bp;
    } bytestream;
    char *string;
};

struct CFETI_FIELD_DESC {
    unsigned short    fieldId;
    unsigned char     fieldType;
    CFETI_FIELD_VALUE fieldValue;
};

struct CFETI_INSTRUMENT_DATA_DESC {
    CFETI_INSTRUMENT         instName;
    CFETI_TRADING_SYSTEM     tsId;
    unsigned short           numFields;
    CFETI_FIELD_DESC        *fieldTable;
};

typedef CFETI_INSTRUMENT_DATA_DESC* CFETI_INSTRUMENT_DATA;
```

Instrument data fields are therefore delivered to the trading system callback as an array of field descriptions, where each field is comprised of the field identifier, field data type (a constant) and the field value itself. The field value is delivered as a union of the relevant data types. An application receiving the market data should extract the appropriate element from the union according to the specified field data type constant. As with any other data delivered by the eSpeed API to the application through its callbacks, information that is to be preserved by the application should be copied. The relationship between field data types and fields in the `CFETI_FIELD_VALUE` union is given in the table below.

Field data type	Field	Description
`CFETI_FIELDTYPE_INT8`	`Int8`	8-bit integer
`CFETI_FIELDTYPE_BYTE`	`Byte`	Unsigned 8-bit integer
`CFETI_FIELDTYPE_INT16`	`Int16`	16-bit integer
`CFETI_FIELDTYPE_UINT16`	`Uint16`	Unsigned 16-bit integer
`CFETI_FIELDTYPE_INT32`	`Int32`	32-bit integer
`CFETI_FIELDTYPE_UINT32`	`UInt32`	Unsigned 32-bit integer
`CFETI_FIELDTYPE_DATETIME`	`DateTime`	Time in seconds since 00:00 01/01/70
`CFETI_FIELDTYPE_STRING`	`String`	Null-terminated string

CFETI_FIELDTYPE_BUFFER	Buffer	Fixed length string
CFETI_FIELDTYPE_DECIMAL	Decimal	Floating point value
CFETI_FIELDTYPE_BYTESTREAM	ByteStream	Encoded fixed length buffer

Dates and times in the eSpeed API market data stream may be carried in fields with either field data type `DateTime` or field data type `Uint32`. Applications should ensure that the correct data type is used according to the field description. If the field data type is `CFETI_FIELD_DATETIME` the `DateTime` element of the `CFETI_FIELD_DESC` data structure should be used. The representation is then the number of seconds since 00:00 01/01/1970 UTC. If the field data type is `Uint32` and the field represents a date the representation shall be CCYYMMDD where CC = century, YY = year (00-99), MM = month (01-12) and DD = day (01 to 31). If the field data type is `Uint32` and the field represents a time the representation shall be HHMMSSCC where HH = hour (00-23), MM = minute (00-59), SS = seconds (00-59), CC = $^{1}/_{100}$ seconds (00-99). Further information on the decoding of the date and time values is given in section 2.18.4.

2.13 Subscribe To List

eSpeed API applications may subscribe to a list of instruments from a given trading system using the subscribe-to-list facility. Subscribe-to-list requests can be submitted using the `CFETIPostMessage` interface. Acceptance of a subscribe-to-list request is an acknowledgement of the validity of the request itself but not of the instruments specified in the request. If a request was issued successfully, the eSpeed API will return an acceptance of the request to the connection callback followed by accepted or rejected subscriptions for each of the instruments in the request. Subsequent market data updates that affect the instrument shall also be delivered to the connection callback. A facility to unsubscribe to a list of instruments is not provided.

```
CFETI_RC CFETIPostMessage(
                        CFETI_SESSION_ID sessId,
                        CFETI_TRADE_SESS_ID trdSysSessId,
                        CFETI_CMD cmd,
                        CFETI_CMDDATA cmdData,
                        CFETI_CMDPREF cmdPreferences );
```

`sessId`	Valid session identifier from previously successful login.
`trdSysSessId`	Trading system session identifier returned on successful connection.
`cmd`	Command (request) being submitted. To subscribe to a list of instruments this shall be `CFETC_SUBSCRIBE_LIST`.
`cmdData`	Command data that contains the details of the request. In the case of an instrument list subscription a pointer to an instrument list data structure `CFETI_INSTRUMENT_LIST_DESC` shall be expected as the command data. An upper limit on the number of instruments that can be included in a list subscription request is imposed by the eSpeed API. This limit is defined by `CFETI_MAX_SUBSCRIBE_LIST_INSTRUMENTS`
`cmdPreferences`	N/A

2.13.1 Return Codes

A successful return code, `CFETI_SUCCESS`, indicates that the subscription request has been successfully processed. Otherwise, the reason for the failure is indicated in the return code. Return codes that may be delivered to the application include:

`CFETI_SUCCESS`	The request was processed successfully. Whether or not the subscription or cancellation of a subscription was successful is returned in the `tradingSysCallback` specified when the original connection request was made.
`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETIPostMessage`.
`CFETI_NO_SESSION`	An attempt is made to subscribe or cancel a subscription to market data without having previously established a session with the eSpeed Session Manager.
`CFETI_NO_SUCH_LOGIN`	The login identified by the supplied session identifier cannot be found.
`CFETI_NO_SUCH_CONNECTION`	The connection specified by the supplied trading system session identifier cannot be found.
`CFETI_MAX_INSTRUMENTS_EXCEEDED`	Maximum number of instruments in the subscribe-to-list request was exceeded .

2.13.2 Instrument Data Definition

The instrument list data structure is defined as follows:

```
typedef struct CFETI_INSTRUMENT_LIST_DESC
                      CFETI_INSTRUMENT_LIST_DESC;
struct CFETI_INSTRUMENT_LIST_DESC {
  CFETI_TRADING_SYSTEM     tsId;
  unsigned int             numInstruments;
  CFETI_INSTRUMENT_DESC   *instList;
};
typedef CFETI_INSTRUMENT_LIST_DESC* CFETI_INSTRUMENT_LIST;
```

2.13.3 Command Response

Responses to a list subscription request are as follows:

Command Response	Command Details
Subscribe List Accepted	Request to subscribe to list was accepted. Individual subscriptions may still be rejected.
Subscribe List Rejected	Possible Reasons: • Invalid authentication credentials. • Unauthorized request

All responses related to the request as well as any subsequent accepted subscriptions and market data updates are returned in the callback routine provided in the initial call to `CFETIConnect`. The callback routine is invoked as follows:

```
void tradeSysCallback(
                              CFETI_CMD cmd,
                              CFETI_CMD_STATUS cmdStatus,
                              CFETI_CMDDATA cmdData,
                              CFETI_UD userData );
```

Commands that may be issued by the eSpeed API to the application as a result of the list subscription request are:

Command	Command Status	Command Data Type
CFETI_SUBSCRIBE_LIST_ACCEPTED	CFETI_SUCCESS	CFETI_INSTRUMENT_LIST
CFETI_SUBSCRIBE_LIST_REJECTED	CFETI_INVALID_AUTH	CFETI_INSTRUMENT_LIST

2.14 Queries

The eSpeed API provides a series of queries that provide summary information for intra-day orders based on valid trade identifiers. The eSpeed API also provides a query mechanism to perform trade instrument lookups based on search criteria. Search results will be returned in the connection callback routine provided during as part of the login process. Queries are submitted using the `CFETIPostMessage` routine. Instrument and trade lookups are directed towards a trading system and therefore require that the user has established a trading system connection.

```
CFETI_RC CFETIPostMessage(
                        CFETI_SESSION_ID sessId,
                        CFETI_TRADE_SESS_ID trdSysSessId,
                        CFETI_CMD cmd,
                        CFETI_CMDDATA cmdData,
                        CFETI_CMDPREF cmdPreferences );
```

`sessId`	Valid session identifier from previously successful login.
`trdSysSessId`	Trading system session identifier returned on successful connection.
`cmd`	Command (request) being submitted. To submit a query this shall be `CFETC_QUERY`.
`cmdData`	Command data that contains the details of the request. In the case of a query `CFETI_QUERY` shall be expected as the command data.
`cmdPreferences`	N/A

2.14.1 Query Types

Supported query types are as follows:

CFETQ_INSTRUMENT — Instrument queries are used to retrieve details of instruments available within the eSpeed system. A set of indicative fields can be supplied to limit the search. The following fields are supported in an instrument query:

CFETF_NAME	Instrument name (accepts * wildcards)
CFETF_DESCRIPTION	Instrument Description (accepts * wildcards)
CFETF_COUPON	Coupon
CFETF_MAT_DATE	Maturity date
CFETF_ISSUE_DATE	Issue date
CFETF_CUSIP	Cusip or ISIN number (accepts * wildcards)

CFETQ_TRADE — Trade queries are used to retrieve details of confirmed trades. Queries are performed either using start/end times or trade Id. If the start-time is zero the search is assumed to be by trade Id.

CFETQ_USERLIST	Retreives the list of users in the same legal entity as the requestor. Use of the query is restricted to certain users determined by eSpeed when the users are created.
CFETQ_TRADE_LE	The local entity trade query is provides a mechanism for retrieval of all trades for the legal entity of the requesting user within the constraints specified. Use of the query is restricted to certain users determined by eSpeed when the users are created. The list of users for which trades are required is specified in the query structure. A list of users can be retrieved using the CFETQ_USERLIST query.
CFETQ_COMMISSION_TABLE	Retrieves commission information. This query is only valid when connected to the MMTS US Treasury OddLots system.

2.14.2 Query Definition

The query structures and associated data types are defined as follows:

```
typedef unsigned char CFETI_QUERY_TYPE;
typedef unsigned short CFETI_INSTRUMENT_TYPE;
typedef char* CFETI_TRADE_ID;
typedef           struct           CFETI_INSTRUMENT_QUERY_DESC
                      CFETI_INSTRUMENT_QUERY_DESC;

struct CFETI_INSTRUMENT_QUERY_DESC {
    CFETI_INSTRUMENT_TYPE    trdType;
    unsigned int             numIndicativeFields;
    CFETI_FIELD              indicativeFields;
};

typedef struct CFETI_TRADE_QUERY_DESC CFETI_TRADE_QUERY_DESC;
struct CFETI_TRADE_QUERY_DESC {
    time_t                   startTime;
    time_t                   endTime;
    CFETI_TRADE_ID           tradeId;
};

typedef            struct            CFETI_TRADE_LE_QUERY_DESC
                       CFETI_TRADE_LE_QUERY_DESC;
struct CFETI_TRADE_LE_QUERY_DESC {
  time_t                   startTime;
  time_t                   endTime;
  CFETI_TRADE_ID           tradeId;
  unsigned int             numUsers;
  char**                   userList;
};

typedef struct CFETI_QUERY_DESC CFETI_QUERY_DESC;
struct CFETI_QUERY_DESC {
    CFETI_QUERY_TYPE         queryType;
    union {
        CFETI_TRADE_QUERY_DESC      trade;
        CFETI_INSTRUMENT_QUERY_DESC instrument;
        CFETI_TRADE_LE_QUERY_DESC   tradeLE;
    } query;
    void* extend; // Reserved for future use
};

typedef CFETI_QUERY_DESC* CFETI_QUERY;
```

The appropriate component of the query shall be completed according to query type.

2.14.3 Return Codes

A successful return code, `CFETI_SUCCESS`, indicates that the query has been successfully processed. Otherwise, the reason for the failure is indicated in the return code. Return codes that may be delivered to the application include:

`CFETI_SUCCESS`	The request was process successfully. Whether or not the query was successful and the results of the query are returned in the `tradingSysCallback` specified when the original connection request was made.
`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETIPostMessage`.
`CFETI_NO_SESSION`	An attempt is made to generate a query without having previously established a session with the eSpeed Session Manager.
`CFETI_NO_SUCH_LOGIN`	The login identified by the supplied session identifier cannot be found.
`CFETI_NO_SUCH_CONNECTION`	The connection specified by the supplied trading system session identifier cannot be found.

2.14.4 Command Responses

Responses to a query request are as follows:

Command Response	Command Details
Query Accepted	Query results
Query Rejected	Query unsuccessful Possible Reasons: • Invalid authentication signature • Invalid query syntax • Invalid trade identifier. • Permission denied • Database error
Query Invalidated	An available query result has been modified. Applications that have cached the query result should re-issue the query if they require an update.

The response to the query request is returned in the callback routine provided in the initial call to `CFETIConnect`. The callback routine is invoked as follows:

```
void tradeSysCallback(
                                CFETI_CMD cmd,
                                CFETI_CMD_STATUS cmdStatus,
                                CFETI_CMDDATA cmdData,
                                CFETI_UD userData );
```

Commands that may be issued by the eSpeed API to the application as a result of the query are:

Command	Command Status	Command Data Type
`CFETI_QUERY_ACCEPTED`	`CFETI_SUCCESS`	`CFETI_QUERY_RESULT`

`CFETI_QUERY_REJECTED`	`CFETI_INVALID_AUTH`
	`CFETI_INVALID_SYNTAX`
	`CFETI_PERMISSION_DENIED`
	`CFETI_DATABASE_ERROR`
		`CFETI_QUERY`

`CFETI_QUERY_INVALIDATED`	`CFETI_SUCCESS`	`CFETI_QUERY`

CFETI_INST_QUERY_COMPLETE	CFETI_SUCCESS	CFETI_TRADING_SYSTEM
CFETI_TRADE_QUERY_COMPLETE	CFETI_SUCCESS	CFETI_TRADING_SYSTEM

2.14.5 Query Results Definition

If the query is successful the results are returned in a CFETI_QUERY_RESULT structure; if it is unsuccessful the original query is returned. As with any other data delivered by the eSpeed API to the application through its callbacks, information that is to be preserved by the application should be copied.

An instrument or trade query result may be delivered in more than one call to the connection callback for large results. A separate call to the callback is made when all of the information has been delivered. In this case the command is CFETI_INST_QUERY_COMPLETE for an instrument query and CFETI_TRADE_QUERY_COMPLETE for a trade query. The command data is the trading system id for which the query that was issued in both cases.

The query result structures and associated data types are defined as follows:

```
typedef struct CFETI_INSTRUMENT_DESC CFETI_INSTRUMENT_DESC;
struct CFETI_INSTRUMENT_DESC {
  CFETI_INSTRUMENT   name;
  char              *description;
  char              *id;        /* e.g. CUSIP or ISIN */
  char              *classification;
  char              *componentId;
  unsigned int       properties;
};

typedef          struct          CFETI_INSTRUMENT_QUERY_RESULT
                    CFETI_INSTRUMENT_QUERY_RESULT;
struct CFETI_INSTRUMENT_QUERY_RESULT {
    unsigned int             numInstruments;
    CFETI_INSTRUMENT_DESC   *instrumentList;
};

typedef            struct            CFETI_TRADE_QUERY_RESULT
                      CFETI_TRADE_QUERY_RESULT;
typedef            struct            CFETI_TRADE_QUERY_RESULT
                      CFETI_TRADE_LE_QUERY_RESULT;
struct CFETI_TRADE_QUERY_RESULT {
    unsigned int             numOrders;
    CFETI_ORDER              orderList;
};

typedef           struct           CFETI_USERLIST_QUERY_RESULT
                     CFETI_USERLIST_QUERY_RESULT;
struct CFETI_USERLIST_QUERY_RESULT {
  unsigned int             numUsers;
  char**                   userList;
};

typedef          struct          CFETI_COMMISSION_TABLE_ENTRY
                     CFETI_COMMISSION_TABLE_ENTRY;
struct CFETI_COMMISSION_TABLE_ENTRY {
  unsigned int type;          /* When-issue or Bill      */
  double yearsToMaturity;     /* Years to maturity       */
  double size;                /* Size                    */
  double adjustment;          /* Adjustment per 100 face */
};

typedef       struct       CFETI_COMMISSION_TABLE_QUERY_RESULT
                    CFETI_COMMISSION_TABLE_QUERY_RESULT;
struct CFETI_COMMISSION_TABLE_QUERY_RESULT {
```

```
  unsigned int                    commissionType;
  double                          flatFee;
  unsigned int                    numFeeTableEntries;
  CFETI_COMMISSION_TABLE_ENTRY*   feeTable;
};

typedef             struct             CFETI_QUERY_RESULT_DESC
                      CFETI_QUERY_RESULT_DESC;
struct CFETI_QUERY_RESULT_DESC {
    CFETI_QUERY_DESC         query;
    union {
        CFETI_TRADE_QUERY_RESULT      trade;
        CFETI_INSTRUMENT_QUERY_RESULT instrument;
        CFETI_USERLIST_QUERY_RESULT   user;
        CFETI_TRADE_LE_QUERY_RESULT   tradeLE;
        CFETI_COMMISSION_TABLE_QUERY_RESULT commissionTable;
    } result;
    void* extend; // Reserved for future use
};

typedef CFETI_QUERY_RESULT_DESC* CFETI_QUERY_RESULT;
```

The instrument query result includes the instrument name, description, industry standard name strings, a classification string and a component identifier. The component identifier (componentId) is used to describe the underlying industry standard names in a switch or swap instrument (separated by a '/' character). Also present is a properties bit-mask describing attributes of the instrument. This can be a combination of zero or more of the values given in the table below.

`CFETI_DISPLAY_PROPERTIES_IS_RESTRICTED_INSTRUMENT`	Used by the eSpeed client application to prevent users from selecting the instrument to place it in the trading grid.
`CFETI_DISPLAY_PROPERTIES_IS_PORTFOLIO_EXCLUDED`	Used by the eSpeed application to prevent users from saving portfolio files that contain references to this instrument in the trading grid.
`CFETI_DISPLAY_PROPERTIES_HAS_CHAIN`	Instrument has a chain defined. When an application has subscribed to this instrument the chain will be defined in the field `CFETF_INSTRUMENT_CHAIN`. If this field is present it is decoded using `CFETIDecodeDataField`.
`CFETI_DISPLAY_PROPERTIES_CHAIN_UNRESTRICTED`	Instrument chain is "unrestricted". The eSpeed GUI does not enforce display of the instruments in the chain if a chain has this property set.
`CFETI_DISPLAY_PROPERTIES_IS_BENCHMARK`	Used by the eSpeed client application to distinguish between instruments designated as benchmarks and other instruments.

2.15 Client Checkout

A mechanism is available by which eSpeed applications can electronically accept or reject trades where there was broker action taken on behalf of the user during its execution.

This facility can only be used if your account is enabled for client checkout and then only for businesses where there is manual broker intervention in the trade in question. For more information please contact your eSpeed Customer Integration representative.

Trades that are required to either be accepted or rejected are delivered to the user with command CFETI_ORDER_HELD. Once the request to accept or reject the trade has been sent by the client application response messages are sent to either accept or reject the request. Subsequent to this a notification is delivered to indicate that the trade has been checked out (command

CFETI_TRADE_CHECKED_OUT) or that the held state has been removed from this order (command CFETI_ORDER_UNHELD). Once a trade has been checked out the trade confirmation message is also delivered in the usual manner.

For businesses where all sides of the trade must be accepted before the trade can be confirmed, the trade shall be re-sent with command CFETI_ORDER_HELD and the command status code CFETI_AWAITING_COUNTERPARTY once the user has accepted the trade if there are other parties to the trade that are yet to accept it. The notifications for the checked out trade and the trade confirmation will be delivered once all parties have accepted the trade.

Immediately prior to the first time a trade is delivered with the command CFETI_ORDER_HELD, an additional notification may be delivered to identify the source of trades delivered for this trading system. The command used shall be CFETI_SET_CHECKOUT_BUSINESS_INFO and the cmdData shall be a pointer to a CFETI_CHECKOUT_BUSINESS_INFO_DESC data structure (defined in cfeti_types.h).

If the client checkout allocation facility is available, then at the time that a trade is accepted the client application may include allocation information in the `allocationInfo` element of the order data structure in the request. (The availability of the feature is indicated at the time of connection to the trading system). The text in this field will be delivered in the subsequent trade confirmation sent to the user concerned and also to any trade feed for the customer.

```
CFETI_RC CFETIPostMessage(
                        CFETI_SESSION_ID sessId,
                        CFETI_TRADE_SESS_ID trdSysSessId,
                        CFETI_CMD cmd,
                        CFETI_CMDDATA cmdData,
                        CFETI_CMDPREF cmdPreferences );
```

`sessId`	Valid session identifier from previously successful login.
`trdSysSessId`	Trading system session identifier returned on successful connection.
`cmd`	Command (request) being submitted. To accept a trade this shall be `CFETC_TRADE_CHECKOUT_REQUEST`. To reject a trade this shall be `CFETC_TRADE_REFUSE_REQUEST`.
`cmdData`	Command data that contains the details of the request. In the case of a trade checkout or refusal a pointer to a `CFETI_ORDER_DESC` data structure shall be expected as the command data. The same data structures are delivered in the responses to the trading session callback.
`cmdPreferences`	N/A

2.15.1 Return Codes

A successful return code, `CFETI_SUCCESS`, indicates that the subscription request has been successfully processed. Otherwise, the reason for the failure is indicated in the return code. Return codes that may be delivered to the application include:

`CFETI_SUCCESS`	The request was processed successfully. Whether or not the trade was posted successfully is returned in the `tradingSysCallback` specified when the original connection request was made.

`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETIPostMessage`.
`CFETI_NO_SESSION`	An attempt is made to subscribe or cancel a subscription to market data without having previously established a session with the eSpeed Session Manager.
`CFETI_NO_SUCH_LOGIN`	The login identified by the supplied session identifier cannot be found.
`CFETI_NO_SUCH_CONNECTION`	The connection specified by the supplied trading system session identifier cannot be found.

2.15.2 Command Response

Responses to a request to accept or reject a trade are as follows:

Command Response	Command Details
Checkout Queued	Receipt of the request to checkout the trade is acknowledged.
Checkout Accepted	The request to accept the trade was accepted.
Checkout Rejected	Possible Reasons: • Unauthorized request • Instrument unknown • Invalid trade
Trade Refusal Queued	Receipt of the request to refuse the trade is acknowledged.
Trade Refusal Accepted	The request to refuse the trade was accepted.
Trade Refusal Rejected	Possible Reasons: • Unauthorized request • Instrument unknown • Invalid trade

Responses to the trade checkout or refusal requests subsequently delivered notifications and trade confirmations are returned in the callback routine provided in the initial call to `CFETIConnect`. The callback routine is invoked as follows:

```
void tradeSysCallback(
                                CFETI_CMD cmd,
                                CFETI_CMD_STATUS cmdStatus,
                                CFETI_CMDDATA cmdData,
                                CFETI_UD userData );
```

Commands that may be issued by the eSpeed API to the application as a result of the subscription of subscription cancellation request are as listed in the table below. In all cases the command data type is a pointer to a CFETI_ORDER_DESC data structure.

Command	Command Status	Description
`CFETI_ORDER_HELD`	`CFETI_SUCCESS`	The order is in a held state and is waiting for the client to either accept or reject the trade.
`CFETI_TRADE_CHECKOUT_QUEUED`	`CFETI_SUCCESS`	The request by the client to accept the trade has been received by the system.
`CFETI_TRADE_CHECKOUT_ACCEPTED`	`CFETI_SUCCESS`	The request by the client to accept the trade has been accepted by the system.
`CFETI_TRADE_CHECKOUT_REJECTED`	`CFETI_UNKNOWN TRD_INST`	The request by the client to accept the trade has been rejected by the system.

	CFETI_ INVALID AUTH	
CFETI_TRADE_ CHECKED OUT	CFETI_SUCCESS	The client has accepted the identified trade.
CFETI_TRADE_ REFUSE_QUEUED	CFETI_SUCCESS	The request by the client to refuse the trade has been received by the system.
CFETI_TRADE_ REFUSE_ACCEPTED	CFETI_SUCCESS	The request by the client to refuse the trade has been accepted by the system.
CFETI_TRADE_ REFUSE_REJECTED	CFETI_UNKNOWN TRD_INST CFETI_ INVALID AUTH	The request by the client to refuse the trade has been rejected by the system.
CFETI_ORDER_ UNHELD	CFETI_SUCCESS	The order is no longer in a held state awaiting client accept or reject. This will be delivered if a trade refusal is accepted. It may also be delivered if client acceptance or rejection of the trade is no longer required by the system.

2.16 Other Notifications

2.16.1 Session Boundaries

After a user has established a connection to a trading system a session notification is delivered to the trading system session callback with the details of the start/end of the current trading session. Subsequently at the close the session or start of a new session additional notifications bay be delivered. The application will receive the CFETI_TRADING_SESSION_STATUS_NOTIFICATION command in its tradeSysCallback with a description of the instrument notification.

```
void tradeSysCallback(
                        CFETI_CMD cmd,
                        CFETI_CMD_STATUS cmdStatus,
                        CFETI_CMDDATA cmdData,
                        CFETI_UD userData );
```

cmd	A command value indicating that a session status notification is issued
cmdStatus	Additional status associated with the given command.
cmdData	A command-specific data structure that may contain information useful to the application.
userData	User data specified when the CFETIConnect call was made delivered to the application callback function.

The notification shall be populated in the cmdData as a pointer to a CFETI_TRADING_SESSION_INFO_DESC data structure, details of which are given below

```
/*
 * CFETI_TRADING_SESSION_INFO_DESC: Notification of the state
                        of the current
 * trading session
 */
typedef struct CFETI_TRADING_SESSION_INFO_DESC
                        CFETI_TRADING_SESSION_INFO_DESC;
struct CFETI_TRADING_SESSION_INFO_DESC
{
  unsigned int uiTradingSystemId; /**< Trading system id */
  unsigned int uiBusinessDate; /**< Trading session business
                        date(YYYYMMDD) */
  unsigned int uiSessionStartDate; /**< Date of trading
                        session (YYYYMMDD) */
```

```
        unsigned int uiSessionStartTime; /**< Start of trading
                              session GMT (HHMMSSCC) */
        unsigned int uiSessionEndDate; /**< Date of end of trading
                              session (YYYYMMDD) */
        unsigned int uiSessionEndTime; /**< GMT time of end of
                              trading session (HHMMSSCC) */
        unsigned int uiSessionState; /**< The enumerated Trading
                              Session State */
    };
```

Enumerated values for the session state are listed in cfeti_consts.h as follows:

CFETI_TRADING_SESSION_STATE_OPEN	Trading Session is open
CFETI_TRADING_SESSION_STATE_CLOSED	Trading Session is closed

2.16.2 Instrument Notify

Instrument notify messages can be sent to notify users of events, or information regarding a specific instrument or selection of instruments. The application will receive the CFETI_INSTRUMENT_NOTIFY call in its tradeSysCallback with a description of the instrument notification.

```
    void tradeSysCallback(
                          CFETI_CMD cmd,
                          CFETI_CMD_STATUS cmdStatus,
                          CFETI_CMDDATA cmdData,
                          CFETI_UD userData );
```

`cmd`	A command value indicating that an instrument-notify message has been sent to users.
`cmdStatus`	Additional status associated with the given command.
`cmdData`	A command-specific data structure that may contain information useful to the application.
`userData`	User data specified when the `CFETIConnect` call was made delivered to the application callback function.

The notification shall be populated in the cmdData as a `CFETI_INSTRUMENTNOTIFY` object, the description of which is given below.

```
/*
 * CFETI_INSTRUMENTNOTIFY_DESC
 */
typedef     struct      CFETI_INSTRUMENTNOTIFY_DESC
CFETI_INSTRUMENTNOTIFY_DESC;
struct CFETI_INSTRUMENTNOTIFY_DESC {
      CFETI_TRADING_SYSTEM    tsId;
      CFETI_INSTRUMENT_DESC*  instList; // list of instruments
      int                     numInstruments; // instrument
count
      unsigned int            notifyType; // type of
notification
      time_t                  sendTime; // time message sent
      char**                  text; // text lines
      int                     textLines; // number of text
lines
```

```
        unsigned int            countdownTime; // time in seconds
};
typedef struct CFETI_INSTRUMENTNOTIFY_DESC*
CFETI_INSTRUMENTNOTIFY;
```

The type of notification message is set in `notifyType`. The possible values for this field are tabled below.

CFETI_INSTRUMENT_NEW	New instrument(s) available
CFETI_INSTRUMENT_DELETE	Instrument(s) no longer available
CFETI_INSTRUMENT_AUTO_ADD	New instrument(s) available (automatic insertion to your portfolio recommended)
CFETI_INSTRUMENT_AUTO_DELETE	Instrument(s) no longer available (automatic removal from your portfolio recommended)
CFETI_INSTRUMENT_ADVISORY	General information about instrument(s)
CFETI_INSTRUMENT_MKT_EXPIRE	Market about to expire on instrument(s)

2.16.2.1 North American Energy

When a market is about to expire, the CFETI_INSTRUMENT_MKT_EXPIRE instrument notify message shall be sent to notify users. This message will be sent from the trading system 5 minutes before the instrument expires, 1 minute before, and again right before it expires. The trading system will then send an Instrument Offline and the instrument will not be accessible. Once the instrument has rolled and is ready for trading again, the trading system will send an Instrument Online message to re-enable the instrument for trading.

2.17 Application Control

The delivery of command responses to eSpeed API applications through callback functions is asynchronous to the application. In order to receive these messages it is incumbent upon eSpeed API applications to use the message processing functions provided by the eSpeed API. Facilities are provided to process messages forever using a message loop, to wait for a single inbound message, as well as facilities to poll for incoming message queues.

2.17.1 CFETIMessageLoop

The `CFETIMessageLoop` interface causes eSpeed API applications to enter an infinite loop waiting for and then processing incoming commands.

```
void CFETIMessageLoop( void );
```

2.17.2 CFETIWaitMessage

The `CFETIWaitMessage` interface causes eSpeed API applications to wait either until an incoming message has been received and processed or until the specified time interval (given in milliseconds) has expired.

```
CFETI_RC CFETIWaitMessage( unsigned long timeOut );
```

`timeOut`	Interval (in milliseconds) to wait for an incoming message. If the interval is zero the routine will wait indefinitely for a

single incoming message.

A successful return code, `CFETI_SUCCESS`, indicates that a message has been processed successfully. If the request timed out a return code `CFETI_TIMED_OUT` is returned. Otherwise, the reason for the failure is indicated in the return code. A full list of return codes is given in Appendix A.

2.17.3 CFETIGetMessage

The `CFETIGetMessage` interface checks to see if there is an unprocessed incoming message and if so processes it. Control is then returned to the application.

```
CFETI_RC CFETIGetMessage( void );
```

A successful return code, `CFETI_SUCCESS`, indicates that a message has been processed successfully. Otherwise, the reason for the failure is indicated in the return code. A full list of return codes is given in Appendix A.

2.17.4 CFETIPeekMessage

The `CFETIPeekMessage` interface checks to see if there are any unprocessed incoming messages not yet processed. No processing of the message takes place.

```
CFETI_RC CFETIPeekMessage( void );
```

A successful return code, `CFETI_SUCCESS`, indicates that a message is available for processing. If there are no messages to be processed the return code `CFETI_NO_MESSAGE` is returned. Otherwise, the reason for the failure is indicated in the return code. A full list of return codes is given in Appendix A.

2.18 Miscellaneous Interfaces

The following interfaces are also provided in the CFETI API.

2.18.1 CFETIVersion

The `CFETIVersion` interface returns a string, which describes the version of the eSpeed API.

```
const char *CFETIVersion( void );
```

2.18.2 CFETIGetLastError

The `CFETIGetLastError` interface returns the error text associated with the last non-success error code returned by an eSpeed API interface. The result is not defined if no eSpeed API interface has returned an error.

```
const char *CFETIGetLastError( void );
```

2.18.3 CFETIDecodeDataField

The `CFETIDecodeDataField` interface may be used to unpack eSpeed API fields that are delivered to the application in an encoded form. The result is returned as a void* pointer since the actual data structure returned is field dependent.

```
CFETI_RC CFETIDecodeDataField(
        const CFETI_SESSION_ID sessId,
        CFETI_TS_SESS_ID trdSysSessId,
        const CFETI_DECODE_DATA_DESC* decodeData,
        void** result );
```

`sessId`	Valid session identifier from previously successful login.
`trdSysSessId`	Trading system session identifier returned on successful connection.
`decodeData`	Data structure describing the data to be decoded.
`Result`	Pointer to the decoded data.

The `CFETIDecodeDataField` data structure is defined as follows:

```
typedef struct CFETI_DECODE_DATA_DESC CFETI_DECODE_DATA_DESC;
struct CFETI_DECODE_DATA_DESC
{
  CFETI_FIELD_DESC* field; /**< Field to decode */
  const char* instrumentName; /**< Instrument name */
  void* instClass; /**< Instrument classification data */
};
```

The `field` and `instrumentName` elements of this data structures are pointers to the field to be decoded and the name of the instrument respectively. If connected to a name-giveup business, the `instClass` argument must be supplied with the value of the field whose field id was passed to the application connection callback (See 2.5.2). This argument contains arbitrary information required for some fields to be decoded.

As with any other data delivered by the eSpeed API to the application, information that is to be preserved by the application should be copied.

2.18.3.1 Return Codes

A successful return code, `CFETI_SUCCESS`, indicates that the field was successfully decoded. Otherwise, the reason for the failure is indicated in the return code. Return codes that may be delivered to the application include:

`CFETI_SUCCESS`	The request was processed successfully. A pointer to the decoded field data is returned in `result`.
`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETIDecodeDataField`.
`CFETI_FIELD_NOT_ENCODED`	The field supplied is not encoded.
`CFETI_FIELD_DECODE_FAILED`	An error occurred decoding the content of the field.

2.18.3.2 Supported Fields

The following encoded fields may be decoded using the `CFETIDecodeDataField` interface to yield a result of the data type indicated.

CFETF_BID_LIST_1 to CFETF_BID_LIST_10	CFETI_PARTICIPANT_LIST
CFETF_ASK_LIST_1 to CFETF_ASK_LIST_10	CFETI_PARTICIPANT_LIST
CFETF_BUY_LIST_1 to CFETF_BUY_LIST_10	CFETI_PARTICIPANT_LIST
CFETF_SELL_LIST_1 to CFETF_SELL_LIST_10	CFETI_PARTICIPANT_LIST
CFETF_TRADE_LIST_1 to CFETF_TRADE_LIST_10	CFETI_PARTICIPANT_LIST
CFETF_COMPOUND_INST_LIST	CFETI_COMPOUND_INST_LIST
CFETF_INSTRUMENT_CHAIN	
CFETF_CHAIN_PARENT_LIST	
CFETF_BENCHMARK_INST_SPEC	
CFETF_SESSION_BOUNDARY_24H	CFETI_PRICE_BOUNDARY_DESC
CFETF_SESSION_BOUNDARY_ASIA	
CFETF_SESSION_BOUNDARY_EUROPE	
CFETF_SESSION_BOUNDARY_US_CASH	
CFETF_SESSION_BOUNDARY_US_FUTURES	
CFETF_SESSION_BOUNDARY_NET_24H	CFETI_PRICE_NET_CHANGE_DESC
CFETF_SESSION_BOUNDARY_NET_ASIA	
CFETF_SESSION_BOUNDARY_NET_EUROPE	
CFETF_SESSION_BOUNDARY_NET_US_CASH	
CFETF_SESSION_BOUNDARY_NET_US_FUTURES	
CFETF_VWA_SESSION_BOUNDARY_24H	CFETI_VWA_BOUNDARY_DESC
CFETF_VWA_SESSION_BOUNDARY_ASIA	
CFETF_VWA_SESSION_BOUNDARY_EUROPE	
CFETF_VWA_SESSION_BOUNDARY_US_CASH	
CFETF_VWA_SESSION_BOUNDARY_US_FUTURES	
CFETF_MARKET_AVAILABITY_NOTIFICATION	CFETI MARKET AVAILABILITY DESC

A full list of all eSpeed API fields and their meaning is given in Appendix B.

2.18.3.3 CFETI_PARTICIPANT_LIST

Bid and Offer participant lists may be decoded using `CFETIDecodeDataField` yielding a result of data type `CFETI_PARTICIPANT_LIST`. This is defined as follows:

```
typedef unsigned int CFETI_PRICECODE;
typedef double CFETI_SIZE;
typedef unsigned int CFETI_TRADE_SESS_ID;
typedef unsigned int CFETI_COUNTERPARTY_STATE;

typedef struct CFETI_PARTICIPAN_LIST_DESC
                    CFETI_PARTICIPANT_LIST_DESC;
struct CFETI_PARTICIPANT_LIST_DESC {
    unsigned int numParticipants;
    CFETI_PARTICIPANT *participant;
};
typedef CFETI_PARTICIPANT_LIST_DESC* CFETI_PARTICIPANT_LIST;

typedef struct CFETI_PARTICIPANT_DESC CFETI_PARTICIPANT_DESC;
struct CFETI_PARTICIPANT_DESC {
    CFETI_TRADE_SESS_ID tsId;
    CFETI_SIZE size;
    CFETI_PRICECODE code;
    CFETI_COUNTERPARTY_STATE counterpartyState;
    unsigned int attributes; /**< Reserved for future use */
    void* extendedInfo; /**< Reserved for future use */
};
typedef CFETI_PARTICIPANT_DESC* CFETI_PARTICIPANT;
```

Each participant in a bid or offer participant list is comprised of:

- An unsigned integer trading system session id

- Size of the market for that participant
- Bit-mask indicating market preferences (e.g. `CFETI_PRICE_FIRM`, `CFETI_PRICE_ALL_OR_NONE`).
- For giveup-enabled businesses, the trading state of that participant (e.g. `CFETI_COUNTERPARTY_STATE_NO_CREDIT`, `CFETI_COUNTERPARTY_STATE_TRADABLE`, `CFETI_COUNTERPARTY_STATE_PRICE_NOT_AVAILABLE`, `CFETI_COUNTERPARTY_STATE_UNKNOWN`, `CFETI_COUNTERPARTY_STATE_NOT_APPLICABLE`[4]).
- The attributes and extendedInfo fields are reserved for future use.

If a trading system session id in the participant list is the same as the trading system session id of the caller then this bid or offer belongs to the caller. If it is not equal to the caller's trading system session id but is some other non-zero value then the bid or offer belongs to another user with the same customer.

A decoded buy or sell list is also represented by the CFETI_PARTCIPANT_LIST structure. The list will contain the outstanding size that may be traded for each participant[5]. A decoded trade list (not available for all eSpeed systems) is also represented by the CFETI_PARTCIPANT_LIST structure. The list will contain the already traded size for each participant. The element is comprised of:

- Trading system session id (this will either be zero or the id of the user if the caller is a participant)
- Sizes for that participant
- Market preferences for that participant
- Trading state for that participant

2.18.3.4 CFETI_COMPOUND_INST_LIST

The compound instrument list provides a mechanism to deliver details of eSpeed instruments that have a relationship to the instrument that contains the field. Typically this shall be used to deliver the instruments that are the two sides of a switch or basis instrument, or the two currencies in an FX Spot instrument. A compound instrument list is simply an array of instrument descriptions and a number that is the number of elements in the array and a bit-mask of list specific properties. Each instrument description provides the eSpeed instrument identifier and the eSpeed trading system id of the instrument.

```
typedef struct CFETI_INST_SPECIFICATION_DESC
CFETI_INST_SPECIFICATION_DESC;
struct CFETI_INST_SPECIFICATION_DESC
{
          unsigned int uiTradingSystemId;
          char* szInstName;
};
typedef struct CFETI_INST_SPECIFICATION_DESC*
CFETI_INST_SPECIFICATION;
typedef struct CFETI_COMPOUND_INST_LIST_DESC
CFETI_COMPOUND_INST_LIST_DESC;
struct CFETI_COMPOUND_INST_LIST_DESC
{
          unsigned int uiNumInstruments;
          unsigned int uiAttributes; /**< List specific properties */
          CFETI_INST_SPECIFICATION pInstSpec;
};
typedef struct CFETI_COMPOUND_INST_LIST_DESC*
CFETI_COMPOUND_INST_LIST;
```

[4] For non giveup-enabled businesses, the trading state shall be set to `CFETI_COUNTERPARTY_STATE_NOT_APPLICABLE`.

[5] Some trading systems may aggregate the participants and present just one entry in the buy or sell list.

The interpretation of the bit-mask shall depend upon the identify of the field that has this encoding. Those fields that define attributes are listed below, together with a description of those attributes. Fields not listed in this table have no attributes defined.

Field Id	Attributes bit-mask values	
CFETF_INSTRUMENT_CHAIN	The attributes for the instrument chain field are used by the eSpeed trading application to determine how to display the instrument chain.	
	CFETI_DISPLAY_CHAIN_BELOW	Display the chain in the order it is given below the parent.
	CFETI_DISPLAY_CHAIN_ABOVE	Display the chain in the order it is given above the parent.
	CFETI_DISPLAY_DEFAULT_GRID	Add the chain to the default trading grid
	CFETI_DISPLAY_ALL_GRIDS	Add the chain to all grids
	CFETI_DISPLAY_FOCUS_GRID	Add the chain to the grid that currently has focus

2.18.3.5 CFETI_PRICE_BOUNDARY_DESC

The Price Boundary information describes the session high, low, open and close prices for the different trading sessions. It is delivered with the following market data fields

- CFETF_SESSION_BOUNDARY_24H
- CFETF_ SESSION_BOUNDARY_ASIA
- CFETF_ SESSION_BOUNDARY_EUROPE
- CFETF_ SESSION_BOUNDARY_US_CASH
- CFETF_ SESSION_BOUNDARY_US_FUTURES

These fields should be decoded by client applications using the `CFETIDecodeDataField` interface. The resultant data structure in this case is a pointer to a structure of type CFETI_PRICE_BOUNDARY_DESC.

```
typedef struct CFETI_PRICE_BOUNDARY_DESC CFETI_PRICE_BOUNDARY_DESC;
struct CFETI_PRICE_BOUNDARY_DESC
{
          double dHigh; /**< Session high price */
          unsigned char highIndicator; /**< Event that generated high
price */
          double dHighYield; /**< Yield corresponding to high price
*/
          double dLow; /**< Session low price */
          unsigned char lowIndicator; /**< Event that generated low
price */
          double dLowYield; /**< Yield corresponding to low price */
          double dOpen; /**< Session opening price */
          double dOpenYield; /**< Session opening yield */
          double dClose; /**< Session closing price */
          double dCloseYield; /**< Session closing yield */
          unsigned int uiSessionState; /**< Session state */
};
```

The following enumerations are provided for the `highIndicator` and `lowIndicator` fields.

`CFETI MARKET BID` The high or low price was generated by a bid on the instrument.

`CFETI MARKET ASK`	The high or low price was generated by an offer on the instrument.
`CFETI_ORDER_HIT`	The high or low price was generated by a sell order on the instrument.
`CFETI_ORDER_TAK`	The high or low price was generated by a buy order on the instrument.

The following enumerations are provided for the `uiSessionState` field.

`CFETI_TRADING_SESSION_STATE_OPEN`	Trading session is open.
`CFETI_TRADING_SESSION_STATE_CLOSED`	Trading session is closed.

2.18.3.6 CFETI_PRICE_NET_CHANGE_DESC

The Price net change information describes the net change for the identified trading session. It is delivered in the following market data fields:

- CFETF_SESSION_BOUNDARY_NET_24H
- CFETF_ SESSION_BOUNDARY_NET_ASIA
- CFETF_ SESSION_BOUNDARY_NET_EUROPE
- CFETF_ SESSION_BOUNDARY_NET_US_CASH
- CFETF_ SESSION_BOUNDARY_NET_US_FUTURES

These fields should be decoded by client applications using the `CFETIDecodeDataField` interface. The resultant data structure in this case is a pointer to a structure of type CFETI_PRICE_NET_CHANGE_DESC.

```
typedef struct CFETI_PRICE_NET_CHANGE_DESC
CFETI_PRICE_NET_CHANGE_DESC;
struct CFETI_PRICE_NET_CHANGE_DESC
{
          double dNetChange; /**< Session net change */
          double dNetChangeYield; /**< Session net change in yield*/
};
```

2.18.3.7 CFETI_VWA_BOUNDARY_DESC

The VWA Boundary information describes the Volume Weighted Average Price and Yield for the different trading sessions. It is delivered with the following market data fields

- CFETF_VWA_SESSION_BOUNDARY_24H
- CFETF_VWA_SESSION_BOUNDARY_ASIA
- CFETF_VWA_SESSION_BOUNDARY_EUROPE
- CFETF_VWA_SESSION_BOUNDARY_US_CASH
- CFETF_VWA_SESSION_BOUNDARY_US_FUTURES

These fields should be decoded by client applications using the `CFETIDecodeDataField` interface. The resultant data structure in this case is a pointer to a structure of type CFETI_VWA_BOUNDARY_DESC.

```
typedef struct CFETI_VWA_BOUNDARY_DESC CFETI_VWA_BOUNDARY_DESC ;
struct CFETI_VWA_BOUNDARY_DESC
{
double dVWAP;          /* Volume weighted average price */
double dVWAY;          /* Volume weighted average yield */
double dNetChangeVWAP;/* Volume weighted average price net change */
double dNetChangeVWAY;/* Volume weighted average yield net change */
```

```
time_t timestamp;     /* Timestamp of last update */
unsigned int uiMilliseconds; /* milliseconds of the timestamp */
unsigned int uiSessionState; /* Enumerated Trading Session State */
};
```

The following enumerations are provided for the `uiSessionState` field.

`CFETI_TRADING_SESSION_STATE_OPEN`	Trading session is open.
`CFETI_TRADING_SESSION_STATE_CLOSED`	Trading session is closed.

2.18.3.8 CFETI_MARKET_AVAILABILITY_DESC

The eSpeed API will deliver notifications of priority for instruments that the application has subscribed to in the market data field CFETF_MARKET_AVAILABILITY_NOTIFICATION. For information regarding the availability of this field please contact your eSpeed Customer Integration representative.

If this field is delivered in an accepted market data subscription or a subsequent update it can be decoded by client applications using the `CFETIDecodeDataField` interface. The resultant data structure is a pointer to a structure of type CFETI_MARKET_AVAILABILITY_DESC.

If in the availability structure a tier is indicated as being available
when a buy/sell/bid/offer is clear for them to trade immediately, where *no* queued delay would result from a trade attempt

```
/**
 * CFETI_MARKET_AVAILABILITY_DESC
 * desc.tier[CFETI_INDEX_MARKET_BID][0] - Bid on tier 1
 * desc.tier[CFETI_INDEX_MARKET_ASK][1] - Ask on tier 2
 * desc.tier[CFETI_INDEX_ORDER_SELL][2] - Sell on tier 3
 * desc.tier[CFETI_INDEX_ORDER_BUY][3]  - Buy on tier 4
 */
typedef struct CFETI_TIER_AVAILABILITY_DESC
CFETI_TIER_AVAILABILITY_DESC;
struct CFETI_TIER_AVAILABILITY_DESC {
    unsigned int availability; /**< Enumerated availability flag */
    double size; /**< Size: valid only if availability is partial */
 };

typedef struct CFETI_MARKET_AVAILABILITY_DESC
CFETI_MARKET_AVAILABILITY_DESC;
struct CFETI_MARKET_AVAILABILITY_DESC {
    const char* instrumentName; /**< eSpeed instrument identifier */
    CFETI_TIER_AVAILABILITY_DESC tier[4][5]; /**< Availability for
Bid/Ask/Sell/Buy for five tiers */
};
```

The following enumerations are provided for the `availability` field:

CFETI_TRADER_AVAILABILITY_ALL	This section of the market is available for anyone to trade for the entire market size.
CFETI_TRADER_AVAILABILTY_ANOTHER	This section of the market is exclusively available to another user for the full market size. A queued delay would result from a trade attempt.
CFETI_TRADER_AVAILABILTY_ANOTHER_PARTIAL	This section of the market is exclusively available to another user for the size specified. A queued delay would result from a trade attempt in the indicated size. The remaining size is not available to anyone.
CFETI_TRADER_AVAILABILTY_YOURS	This section of the market is exclusively available to this user for the full market size. The market is available for the client to trade immediately and no

	queued delay would result from a trade attempt.
CFETI_TRADER_AVAILABILTY_YOURS_PARTIAL	This section of the market is exclusively available to this user for the size specified. The indicated size is available for the client to trade immediately and no queued delay would result from a trade attempt. The remaining size is not available to anyone.
CFETI_TRADER_AVAILABILITY_JOINT	This section of the market is available to the user, and is also available to one other user for the full market size. The market is available for the client and one other user only to trade immediately.
CFETI_TRADER_AVAILABILITY_JOINT_PARTIAL	This section of the market is available to the user, and is also available to one other user for the size specified. The indicated size is available for the client to trade immediately. The remaining size is not available to anyone.
CFETI_TRADER_AVAILABILITY_NONE	This section of the market is not available to anyone.

The following enumerations are provided to allow applications to index the `tier` matrix for bid, ask, buy and sell.

CFETI_INDEX_MARKET_BID	Key to market bid cells in the availability matrix.
CFETI_INDEX_MARKET_ASK	Key to market ask cells in the availability matrix.
CFETI_INDEX_ORDER_SELL	Key to market sell cells in the availability matrix.
CFETI_INDEX_ORDER_BUY	Key to market buy cells in the availability matrix.

2.18.4 Encoding and decoding of CFETI_DATE and CFETI_TIME

The types CFETI_DATE and CFETI_TIME data types define encoding for date and time respectively. A date beyond the year 2038 can be specified with CFETI_DATE. CFETI_DATE has the format CCYYMMDD where CC = century, YY = year (00-99), MM = month (01-12) and DD = day (01 to 31). CFETI_TIME has the format HHMMSSCC where HH = hour (00-23), MM = minute (00-59), SS = seconds (00-59), CC = $^{1}/_{100}$ seconds (00-99).

Constants used to encode and decode calculations for both date and time:

```
P = 1000000
Q = 10000
R = 100
```

A ***date*** field as an unsigned int in the format CCYYMMDD, is decoded by:

```
cc = date / P
yy = (date – (cc * P)) / Q
mm = (date – (cc * P) – (yy * Q)) / R
dd = date modulus R
```

The fields ***cc, yy, mm, dd*** are encoded into an unsigned int ***date*** by:

```
date = cc*P + yy*Q + mm*R + dd
```

A ***time*** field as an unsigned int in the format HHMMSSCC, is decoded by:

```
hh = time / P
mm = (time – (hh * P)) / Q
ss = (time – (hh * P) – (mm * Q)) / R
cc = time modulus R
```

The fields ***hh, mm, ss, cc*** are encoded into an unsigned int ***time*** by:
time = hh*P + mm*Q + ss*R + cc

2.18.5 CFETISetPassword

The `CFETISetPassword` interface provides a mechanism for eSpeed API applications to submit a request to change their eSpeed login password.

```
CFETI_RC CFETISetPassword (
                        const CFETI_SESSION_ID sessId,
                        const char* oldPassword,
                        const char* newPassword,
                        const char* confirmedPassword );
```

`sessId`	Valid session identifier from previously successful login.
`oldPassword`	The existing password that is to be replaced
`newPassword`	The new password
`confirmation`	The new password

A successful return code, `CFETI_SUCCESS`, indicates that the change password request has been successfully processed. (Not that the password itself has been changed – the success or failure of the request is delivered to the application through the system callback supplied when the login was established). Failure indicates that the eSpeed API request has not been processed. In this case the reason is indicated in the return code. Return codes that may be delivered are listed below.

`CFETI_SUCCESS`	Successful return code indicates that the eSpeed API routine has been successfully processed. Command details will be returned in the `systemCallback`.
`CFETI_INVALID_ARG`	Invalid argument supplied to `CFETILogin` (e.g. unknown session, password, new password or confirmation not specified, new password is too long or contains invalid characters).
`CFETI_NEW_PASSWORD_NOT_MATCHED`	No new password does not match the confirmed new password.
`CFETI_NEW_PASSWORD_REUSE_ERROR`	The new password resembles the old password too closely.

Commands that may be issued by the eSpeed API to the application system callback as a result of the set password request are as follows:

Command	Command Status	Command Data Type
`CFETI_SET_PASSWORD_ACCEPTED`	`CFETI_SUCCESS`	`CFETI_SESSION_ID`

`CFETI_SET_PASSWORD_REJECTED`	`CFETI_NEW_PASSWORD_NOT_MATCHED`
	`CFETI_NEW_PASSWORD_REUSE_ERROR`
	`CFETI_NEW_PASSWORD_INVALID_LENGTH`
	`CFETI_NEW_PASSWORD_INVALID_FORMAT`
	`CFETI_NEW_PASSWORD_NOT_SAVED`
	`CFETI_NEW_PASSWORD_INVALID`
		`CFETI_SESSION_ID`

3. Event and Message Flow

The following descriptions are provided to help clarify the application event flow. Specific examples have been included to further demonstrate the application events.

3.1 Establish connectivity

Applications must authenticate themselves to the eSpeed API using `CFETILogin`. If the user is successfully authenticated the eSpeed API will return a session identifier that the application must maintain for all subsequent requests as well as a list of available trading systems.

Applications may then attempt to connect to one of the trading systems required using `CFETIConnect`. For each connection the eSpeed API will return a trading system session identifier that the application must also maintain for further communication with that trading system.

Once successful connectivity has been established, applications may begin to interact with that trading system.

3.2 Posting Markets and Orders

Applications may post Markets (Bids and Offers) and Orders (Buys and Sells) with any of the trading systems to which they have connected.

Before submitting such requests, applications need to identify the instrument they wish to trade. Applications are expected to identify instruments using CF naming conventions. Applications can search the CF Instrument database using the query facility provided in the eSpeed API.

Applications are not required to subscribe to issues before posting a market or order. However, since eSpeed API does not automatically subscribe to the instruments specified it is strongly recommended that applications or application users should do so.

Markets and Orders may be submitted to any trading system using the `CFETIPostMessage` interface. Applications must provide the proper credentials for both the session and trading system for all such messages.

3.2.1 Market Events

Applications posting markets should expect to receive the following events.

Market Accepted: Applications will receive a market-accepted message for each market accepted by the trading system.

Market Data Updates: If the application has subscribed to the market data for the instrument, a market data update will be delivered each time the status of the trade instrument is modified.

Market Executions: When a market is hit or taken by a buyer or seller, the application that posted the market will receive a market execution. The event will indicate the market (trade instrument, price, size and side) which was executed and a trade reference identifier. As the trade progresses multiple market executions may be received. Each execution will contain an incremental size. In the event that a trade error occurs, a market execution with a negative size will be received to offset the previous execution event.

At the completion of the trade a trade confirmation will be received which contains the total market which was executed.

Market Cancellations: When the market is removed from the trade instrument a market cancellation will be received. The cancellation will contain the market that has been removed and the reason for the cancellation.

Trade Confirmations: As each trade completes all participants in the trade will be notified. The trade confirmation is the only event which applications should consider as a successful transaction.

A trade confirmation will contain a trade reference identifier and the summary details of the trade that has been executed. The confirmation will also indicate the portion of the market or order that remained at the time the transaction was completed. The remaining portion of the position will automatically be cleared from the system before the next trade begins. A market cancellation is delivered to indicate that the remaining size has been cleared.

3.2.2 Order Events

Applications submitting orders should expect to receive the following events:

Order Queued: When an ordered is received by the trading system the application that submitted the request will receive an order queued event. Applications should mark the order as pending and wait for further events.

Market Data Updates: If the application has subscribed to the market data for the instrument, a market data update will be delivered each time the status of the trade instrument is modified.

Order Executed: As your order is filled by the trading system, order executed events will be received. The order update will contain incremental details on the state of the order and a trade reference identifier. As the trade progresses additional order updates may be received.

At the completion of the trade a trade confirmation will be received which contains the total order which was executed.

Trade Confirmations: As each trade completes all participants in the trade will be notified. The trade confirmation is the only event which applications should consider as a successful transaction.

A trade confirmation will contain a trade reference identifier and the summary details of the trade that has been executed. The confirmation will also indicate the portion of the order that remained at the time the transaction was completed. The remaining portion of the position will automatically be cleared from the system before the next trade begins. An order cancellation is delivered to indicate that the remaining size has been cleared.

3.3 Canceling Markets and Orders

Applications may cancel markets (Bids and Offers) and Orders (Buys and Sells) with any of the trading systems to which they have posted requests provided that the minimum market/order timers have expired.

As with posting markets and orders, applications need to identify the instrument they wish to cancel and must provide the proper credentials. When a cancellation is received by the trading system, the request will be processed based on the current state of the trade instrument. A cancel accepted or rejected event will be received to indicate the action the trading system has taken. The event will contain the market or order that was removed from the system.

Applications should expect to receive the appropriate market and order events for the portions of the market or order that were processed prior to receipt of the cancellation request.

Appendix A - eSpeed API Error Codes

eSpeed API Error codes are returned by functions defined in the eSpeed API. These values are defined in the eSpeed API include file `cfeti_consts.h`.

Error Code	Description
`CFETI_SUCCESS`	Successful return code indicates that the eSpeed API routine has been successfully processed. Command details will be returned in the appropriate callback.
`CFETI_NOMEMORY`	An attempt to dynamically allocate memory failed.
`CFETI_INVALID_ARG`	An invalid argument was supplied to the interface (e.g. a null user name or session id).
`CFETI_SESSION_EXISTS`	Only one physical connection to a session manager may be established using `CFETIOpenSession`, though multiple logins are supported.
`CFETI_MAX_INSTANCE`	Internal error.
`CFETI_MAX_SESSION`	Internal error.
`CFETI_NO_SESSION`	An attempt has been made to perform an eSpeed API operation without having completed authentication.
`CFETI_NO_MESSAGE`	No pending messages are present in the inbound message queue.
`CFETI_NO_CALLBACK`	No user callback was supplied.
`CFETI_NO_SUCH_LOGIN`	A user login does not exist for the specified session id.
`CFETI_ENQUEUE`	Internal error.
`CFETI_UNKNOWN_COMMAND`	The posted command is not defined.
`CFETI_LOGIN_EXISTS`	A login already exists for the named user.
`CFETI_NO_SUCH_CONNECTION`	A connection does not exist for the specified trading system connection id.
`CFETI_UNEXPECTED_MESSAGE`	An unexpected message was received from the session manager.
`CFETI_SESSMGR_SPEC`	Invalid session manager name specification in call to `CFETIOpenSession`.
`CFETI_CONNECT_FAIL`	Connection to the session manager(s) specified in the primary and secondary session manager names was not successful.
`CFETI_CONNECTION_NOT_ON_LINE`	eSpeed API request cannot be accepted because the connection is not on-line.
`CFETI_TIMED_OUT`	A message was not received from the session manager within the specified time interval in a call to

	`CFETIWaitMessage`.
`CFETI_FIELD_NOT_ENCODED`	An attempt was made to decode a field using `CFETIDecodeDataField` that was not delivered encoded.
`CFETI_FIELD_DECODE_FAILED`	An error occurred when decoding a field using `CFETIDecodeDataField`.
`CFETI_SDK_EXPIRED`	The eSpeed API library has expired and is no longer valid.
`CFETI_USER_DATA_TOO_LARGE`	User data size specified in market or order exceeds the limit CFETI_MAX_USER_DATA_SIZE.
`CFETI_SHORTCODE_TOO_LONG`	User short code specified in market or order exceeds the limit CFETI_MAX_SHORTCODE_LENGTH.
`CFETI_DIRECTORY_READ_ERROR`	The directory optionally specified by environment variable `CFETI_LOG_PATH` for log-files does not exist.
`CFETI_FILE_WRITE_ERROR`	A log file could not be opened for writing.

Appendix B - eSpeed API Field Codes

eSpeed API applications that subscribe to instrument market data receive that data as an array of fields. Each element of this array is comprised of a field identifier, a field data type and a field value. The field identifiers are defined in the eSpeed API include file `cfeti_fields.h`. The field data types are defined in the eSpeed API include file `cfeti_consts.h`. This is a complete list of all eSpeed API fields; the actual fields available for an individual instrument will be a subset of those listed here.

Field Identifier	Data Type	Description
`CFETF_BID`		Synonym for CFETF_BID_1
`CFETF_BID_1`	Decimal	Current display bid price

`CFETF_BID_2`
`CFETF_BID_3`
`CFETF_BID_4`
`CFETF_BID_5`
`CFETF_BID_6`
`CFETF_BID_7`
`CFETF_BID_8`
`CFETF_BID_9`
`CFETF_BID_10`
	Decimal	Bid prices behind the best bid

`CFETF_BID_SIZE`		Synonym for CFETF_BID_SIZE_1
`CFETF_BID_SIZE_1`	Decimal	Current total bid volume

`CFETF_BID_SIZE_2`
`CFETF_BID_SIZE_3`
`CFETF_BID_SIZE_4`
`CFETF_BID_SIZE_5`
`CFETF_BID_SIZE_6`
`CFETF_BID_SIZE_7`
`CFETF_BID_SIZE_8`
`CFETF_BID_SIZE_9`
`CFETF_BID_SIZE_10`
	Decimal	Quantities for bids behind the best bid

`CFETF_BID_LIST`		Synonym for CFETF_BID_LIST_1
`CFETF_BID_LIST_1`	Buffer	Encoded bid list for current display bid price (Decode with `CFETIDecodeDataField).`

`CFETF_BID_LIST_2`
`CFETF_BID_LIST_3`
`CFETF_BID_LIST_4`
`CFETF_BID_LIST_5`
`CFETF_BID_LIST_6`
`CFETF_BID_LIST_7`
`CFETF_BID_LIST_8`
`CFETF_BID_LIST_9`
`CFETF_BID_LIST_10`
	Buffer	Encoded bid lists for bid prices behind the best bid
		(Decode with `CFETIDecodeDataField).`

`CFETF_ASK`		Synonym for CFETF_ASK_1
`CFETF_ASK_1`	Decimal	Current display offer price

`CFETF_ASK_2`
`CFETF_ASK_3`
`CFETF_ASK_4`
`CFETF_ASK_5`
`CFETF_ASK_6`
`CFETF_ASK_7`
	Decimal	Offer prices behind the best bid

CFETF_ASK_8 CFETF_ASK_9 CFETF_ASK_10		
CFETF_ASK_SIZE		Synonym for CFETF_ASK_SIZE_1
CFETF_ASK_SIZE_1	Decimal	Current total offer volume
CFETF_ASK_SIZE_2 CFETF_ASK_SIZE_3 CFETF_ASK_SIZE_4 CFETF_ASK_SIZE_5 CFETF_ASK_SIZE_6 CFETF_ASK_SIZE_7 CFETF_ASK_SIZE_8 CFETF_ASK_SIZE_9 CFETF_ASK_SIZE_10	Decimal	Quantities for offers behind the best offer
CFETF_ASK_LIST		Synonym for CFETF_ASK_LIST_1
CFETF_ASK_LIST_1	Buffer	Encoded offer list for current display offer price (Decode with `CFETIDecodeDataField`).
CFETF_ASK_LIST_2 CFETF_ASK_LIST_3 CFETF_ASK_LIST_4 CFETF_ASK_LIST_5 CFETF_ASK_LIST_6 CFETF_ASK_LIST_7 CFETF_ASK_LIST_8 CFETF_ASK_LIST_9 CFETF_ASK_LIST_10	Buffer	Encoded offer lists for offer prices behind the best offer (Decode with `CFETIDecodeDataField`).
CFETF_STATE	UInt16	Bit mask indicating current instrument state.
CFETF_STATE_1		Synonym for CFETF_STATE
CFETF_STATE_2 CFETF_STATE_3 CFETF_STATE_4 CFETF_STATE_5 CFETF_STATE_6 CFETF_STATE_7 CFETF_STATE_8 CFETF_STATE_9 CFETF_STATE_10	UInt16	Bit mask indicating current instrument state.
CFETF_TRADE_PRICE	Decimal	Price at which instrument is trading. (Will be zero when the instrument is not trading).
CFETF_TRADE_PRICE_1		Synonym for CFETF_TRADE_PRICE
CFETF_TRADE_PRICE_2 CFETF_TRADE_PRICE_3 CFETF_TRADE_PRICE_4 CFETF_TRADE_PRICE_5 CFETF_TRADE_PRICE_6 CFETF_TRADE_PRICE_7 CFETF_TRADE_PRICE_8 CFETF_TRADE_PRICE_9 CFETF_TRADE_PRICE_10	Decimal	Price at which instrument is trading. (Will be zero when the instrument is not trading).
CFETF_TRADE_SIZE	Decimal	Quantity that has been traded so far at the best price. (Will be zero when the instrument is not trading).

CFETF_TRADE_SIZE_1		Synonym for CFETF_TRADE_SIZE
CFETF_TRADE_SIZE_2 CFETF_TRADE_SIZE_3 CFETF_TRADE_SIZE_4 CFETF_TRADE_SIZE_5 CFETF_TRADE_SIZE_6 CFETF_TRADE_SIZE_7 CFETF_TRADE_SIZE_8 CFETF_TRADE_SIZE_9 CFETF_TRADE_SIZE_10	Decimal	Quantity that has been traded so far at the corresponding price tier. (Will be zero when the instrument is not trading).
CFETF_TOTAL_TRADE_SIZE	Decimal	Quantity that has been traded so far at all price tiers. (Will be zero when the instrument is not trading).
CFETF_BUY_SIZE	Decimal	Total quantity that the market is willing to buy
CFETF_BUY_SIZE_1		Synonym for CFETF_BUY_SIZE
CFETF_BUY_SIZE_2 CFETF_BUY_SIZE_3 CFETF_BUY_SIZE_4 CFETF_BUY_SIZE_5 CFETF_BUY_SIZE_6 CFETF_BUY_SIZE_7 CFETF_BUY_SIZE_8 CFETF_BUY_SIZE_9 CFETF_BUY_SIZE_10	Decimal	Total quantity that the market is willing to buy
CFETF_BUY_LIST	Buffer	Encoded list of buyers at the current trade price
CFETF_BUY_LIST_1		Synonym for CFETF_BUY_LIST
CFETF_BUY_LIST_2 CFETF_BUY_LIST_3 CFETF_BUY_LIST_4 CFETF_BUY_LIST_5 CFETF_BUY_LIST_6 CFETF_BUY_LIST_7 CFETF_BUY_LIST_8 CFETF_BUY_LIST_9 CFETF_BUY_LIST_10	Buffer	Encoded list of buyers at the current trade price
CFETF_SELL_SIZE	Decimal	Total quantity that the market is willing to sell
CFETF_SELL_SIZE_1		Synonym for CFETF_SELL_SIZE
CFETF_SELL_SIZE_2 CFETF_SELL_SIZE_3 CFETF_SELL_SIZE_4 CFETF_SELL_SIZE_5 CFETF_SELL_SIZE_6 CFETF_SELL_SIZE_7 CFETF_SELL_SIZE_8 CFETF_SELL_SIZE_9 CFETF_SELL_SIZE_10	Decimal	Total quantity that the market is willing to sell
CFETF_SELL_LIST	Buffer	Encoded list of sellers at the current trade price
CFETF_SELL_LIST_1		Synonym for CFETF_SELL_LIST
CFETF_SELL_LIST_2 CFETF_SELL_LIST_3 CFETF_SELL_LIST_4 CFETF_SELL_LIST_5	Buffer	Encoded list of sellers at the current trade price

CFETF_SELL_LIST_6 CFETF_SELL_LIST_7 CFETF_SELL_LIST_8 CFETF_SELL_LIST_9 CFETF_SELL_LIST_10		
CFETF_LAST		Synonym for CFETF_LAST_1
CFETF_LAST_1	Decimal	Last trade price
CFETF_LAST_2 CFETF_LAST_3	Decimal	Previous last trade prices
CFETF_LAST_SIZE		Synonym for CFETF_LAST_SIZE_1
CFETF_LAST_SIZE_1	Decimal	Last traded quantity
CFETF_LAST_SIZE_2 CFETF_LAST_SIZE_3	Decimal	Previous last traded quantities
CFETF_LAST_YIELD	Decimal	Yield at last traded price
CFETF_LAST_TIME	DateTime	Time of last trade
CFETF_VOLUME	Decimal	Traded volume during current market session
CFETF_OPEN	Decimal	Opening price
CFETF_HIGH	Decimal	Day high
CFETF_LOW	Decimal	Day low
CFETF_CLOSE	Decimal	Closing price
CFETF_TICK	Decimal	Price up/down tick (-1 = down, +1 = up)
CFETF_BID_YIELD	Decimal	Yield for current bid
CFETF_ASK_YIELD	Decimal	Yield for current offer
CFETF_COUPON	Decimal	Coupon rate
CFETF_MAT_DATE	DateTime or UInt32	Maturity date
CFETF_ISSUE_DATE	DateTime or UInt32	Bond issue date
CFETF_SETTLEMENT_DATE	DateTime or UInt32	Settlement date
CFETF_SETTLEMENT	Decimal	Settlement price
CFETF_FIRST_COUPON	DateTime or UInt32	First coupon date
CFETF_CANTORID	String	Instrument name
CFETF_CUSIP	String	Instrument CUSIP
CFETF_ISIN	String	Instrument ISIN number
CFETF_DESCRIPTION	String	Instrument description
CFETF_STATUS	String	Instrument status text

CFETF_PRICE_TICK	Decimal	Minimum price tick
CFETF_SIZE_MULTIPLIER	Decimal	Base market quantity (e.g. 1000000)
CFETF_SUBSYSTEM	UInt32	Trading sub-system for instrument.
CFETF_AUCTION_DATE	DateTime or UInt32	Auction date
CFETF_ANNOUNCEMENT_DATE	DateTime or UInt32	Announcement date
CFETF_ALT_INST_1	String	Name of an alternate instrument
CFETF_PRICETYPE	UInt32	Bit-mask to indicate price type
CFETF_DISPLAY_PRICETYPE	UInt32	Bit-mask to indicate price type where the display price format is different from the input price format.
CFETF_MINIMUM_SIZE	Decimal	Minimum market quantity
CFETF_SIZE_TICK	Decimal	Tick size
CFETF_LABEL	String	Instrument label
CFETF_PRICE_SEPARATOR	String	String used to separate prices in a spread
CFETF_COLLATERAL	String	Underlying instrument in a repo trade.
CFETF_COLLATERAL_ALL_IN_PRICE	Double	Market price adjusted for accrued interest
CFETF_LAST_BID	Decimal	Last bid price
CFETF_LAST_ASK	Decimal	Last offer price
CFETF_END_DATE	DateTime or UInt32	End date for a Repo or maturity date for an Intrerest Rate Derivative.
CFETF_REPO_END_DATE	DateTime or UInt32	Deprecated
CFETF_PRICE_TICK_RULES	UInt32	Enumeration for how to apply tick up/tick down for Repo instruments.
CFETF_COLLATERAL	String	Collateral for repo instrument.
CFETF_SETTLEMENT_PRICE_TYPE	UInt32	Bit-mask to indicate display format of settlement price.
CFETF_PRICE_DEC_PLACES	UInt32	Number of decimal places to display a price when the price type is CFETI_PRICETYPE_DECIMAL_DVAR
CFETF_PRICE_MIN_DEC_PLACES	UInt32	Minimum number of decimal places to display a price when the price type is CFETI_PRICETYPE_DECIMAL_DVAR
CFETF_SETTLEMENT_PRICE_DEC_PLACES	UInt32	Number of decimal places to display a settlement price when the price type is CFETI_PRICETYPE_DECIMAL_DVAR
CFETF_SETTLEMENT_PRICE_MIN_DEC_PLACES	UInt32	Minimum number of decimal places to display a settlement price when the price type is

		CFETI_PRICETYPE_DECIMAL_DVAR
CFETF_DISPLAY_PRICE_DEC_PLACES	UInt32	Number of decimal places to display a settlement price when the display price type is CFETI_PRICETYPE_DECIMAL_DVAR
CFETF_DISPLAY_PRICE_MIN_DEC_PLACES	UInt32	Minimum number of decimal places to display a settlement price when the display price type is CFETI_PRICETYPE_DECIMAL_DVAR
CFETF_HEDGE_RATIO	Decimal	Information for deducing the correct amount of instruments for the trade in an underlying instrument.
CFETF_CONVERSION_FACTOR	Decimal	Information for deducing the correct price for the trade in an underlying instrument.
CFETF_INSTRUMENT_SETTLEMENT_DATE	DateTime or UInt32	Settlement date for this instrument, when that settlement date is inconsistent with the general settlement date for instruments of that sector.
CFETF_PROPERTIES	UInt32	Instrument properties (bit-mask). Possible values for this field are defined in cfeti_consts.h
CFETF_INSTRUMENT_CLASSIFICATION	UInt16	Instrument clasification field which can be used by Giveup enabled trading systems.
CFETF_FUTURE_CROSS_PRICE	Decimal	Future Cross Price. Implemented for Gilt Basis trades. Is published when a basis security is trading.
CFETF_ENERGY_INDEX_TYPE	String	Index type
CFETF_CUTOFF_TIME	UInt32	Time and location for expiration cut-off time (Format is HHMMSSCC where HH (hour) MM (minutes) SS (seconds) CC (100ths))
CFETF_CUTOFF_REGION	String	Region in which time is specified
CFETF_CONTRACT_DATE	UInt32	Trading day for which contracts will be written. (Format is CCYYMMDD where CC (century – 1) YY (year) MM (month) DD (day))
CFETF_EXPIRY_DATE	UInt32	Expiration date of the option. (Format is CCYYMMDD where CC (century – 1) YY (year) MM (month) DD (day))
CFETF_DELIVERY_DATE	UInt32	Trading day for which contracts will be written. (Format is CCYYMMDD where CC (century – 1) YY (year) MM (month) DD (day))
CFETF_INST_CLASSIFICATION	String	The instrument classification string that is also shown in instrument query results
CFETF_PRICING_METHOD	UInt32	Enumerated field used to indicate how a product is priced.
CFETF_START_DATE	DateTime or UInt32	The start date of a REPO instrument.
CFETF_PAYMENT_DATE	DateTime or UInt32	The date that counterparties must settle netted cash flows.
CFETF_BREAK_CLAUSE	String	Date(s) during the life of the derivatives contract that it can be mutually agreed to close the contract at the then market rate.
CFETF_ENERGY_START_CALENDAR_DATE	UInt32	First day of first month of flow. CCYYMMDD
CFETF_ENERGY_END_CALENDAR_DATE	UInt32	Last day of last month of flow. CCYYMMDD
CFETF_ENERGY_CONTRACT_TYPE	Byte	Contract type. CFETI_ENERGY_CONTRACT_FIRM or CFETI_ENERGY_CONTRACT_INTERRUPTIBLE.

CFETF_ENERGY_POINT_NAME	String	Pipeline meter name
CFETF_ENERGY_POINT_NUMBER	UInt32	Pipeline meter number
CFETF_ENERGY_QUANTITY_UNIT	Byte	Enumerated value to identify quantity unit : CFETI_ENERGY_QUANTITY_UNIT_DTH, CFETI_ENERGY_QUANTITY_UNIT_GJ, CFETI_ENERGY_QUANTITY_UNIT_MW, CFETI_ENERGY_QUANTITY_UNIT_BBL, CFETI_ENERGY_QUNATITY_UNIT_TON, CFETI_ENERGY_QUNATITY_UNIT_MMBTU
CFETF_CURRENCY	String	ISO standard 3-character mnemonic for currency.
CFETF_ENERGY_PRICE_TYPE	Byte	Price type. One of CFETI_ENERGY_PRICE_TYPE_FIXED, CFETI_ENERGY_PRICE_TYPE_INDEXED, CFETI_ENERGY_PRICE_TYPE_SINGSWAP, CFETI_ENERGY_PRICE_TYPE_BASISSWAP, CFETI_ENERGY_PRICE_TYPE_FIXEDSWAP, CFETI_ENERGY_PRICE_TYPE_OUTRIGHT_FORWARD.
CFETF_ENERGY_INDEX_NAME	String	Name of index (GDD = Gas Daily Daily, IFE = Inside FERC, etc)
CFETF_ENERGY_LOAD_PROFILE	String	Load profile e.g. 5X16, 6X16,2X24 etc.
CFETF_ENERGY_PRODUCT_PERIOD	Byte	Enumerated value used to denote product period: CFETI_ENERGY_PRODUCT_PEAK or CFETI_ENERGY_PRODUCT_OFFPEAK
CFETF_ENERGY_PRODUCT_TYPE	Byte	Enumerated value used to denote product type: CFETI_ENERGY_PRODUCT_PHYSICAL or CFETI_ENERGY_PRODUCT_FINANCIAL
CFETF_ENERGY_SHORT_LOCATION	String	Short description of full location name
CFETF_ENERGY_SHORT_INDEX_NAME	String	Short description of full index name
CFETF_TIME_ZONE	String	3 character mnemonic for prevailing time zone
CFETF_COMMODITY_CODE	String	Commodity code
CFETF_ENERGY_FIRST_TRADE_DATE	Uint32	First date a product is listed
CFETF_ENERGY_LAST_TRADE_TIME	Uint32	Last time a product is listed
CFETF_ENERGY_PRICE_MECHANISM	String	Price mechanism (e.g. Fixed, SWAP, NGI Chicago).
CFETF_NAME	String	eSpeed instrument identifier.
CFETF_DAILY_LAST_TRADED_PRICE	Decimal	Intra-day last traded price. The field will be cleared at the end of the trading day and re-published only once trading has begun for the new trading day.
CFETF_TSWAP_RATIO	Decimal	The value is the ratio of an increment movement of the price of an instrument against an underlying instrument.
CFETF_PV01_RATIO	Decimal	Synonym for CFETF_TSWAP_RATIO.
CFETF_LOCK_PRICE	Decimal	Lock price of the underlying instrument. A. Zero value for this field is an invalid lock price.
CFETF_BID_YIELD_SPREAD	Decimal	Yield spread corresponding to the best bid price. Zero if there is no price.
CFETF_ASK_YIELD_SPREAD	Decimal	Yield spread corresponding to the best ask price. Zero if there is no price.
CFETF_YIELD_PRICE_TYPE	Uint32	Format to be used to display values in the yield and yield spread fields.
CFETF_LOCK_YIELD	Decimal	The yield value corresponding to the lock price.
CFETF_MARKET_DURATION	String	Text description of the date duration or effective start date of an energy security
CFETF_TRADE_THROUGH_STACK_PRICE_TICKS	Byte	Maximum number of price ticks to which trading through stack is permitted.
CFETF_ENERGY_INDEX_TYPE_ID	Uint32	Enumerated value for index type. Possible values: CFETI_ENERGY_INDEX_TYPE_FXD

		CFETI_ENERGY_INDEX_TYPE_GDD CFETI_ENERGY_INDEX_TYPE_GDM CFETI_ENERGY_INDEX_TYPE_NYM CFETI_ENERGY_INDEX_TYPE_IFE CFETI_ENERGY_INDEX_TYPE_NGI CFETI_ENERGY_INDEX_TYPE_IMO CFETI_ENERGY_INDEX_TYPE_PPA
CFETF_ENERGY_INDEX_TYPE_ID_2	Uint32	Enumerated value for index type of swap instrument. Possible values are as for CFETF_ENERGY_INDEX_TYPE_ID.
CFETF_ENERGY_INDEX_TYPE_2	String	Name of index type for swap instrument
CFETF_ENERGY_INDEX_NAME_2	String	Name of index for swap instrument
CFETF_ENERGY_SHORT_INDEX_NAME_2	String	Description of index for swap instrument
CFETF_COMPOUND_INST_LIST	Bytestream	Encoded list of instruments (Decode with `CFETIDecodeDataField`)
CFETF_SIZE_FORMAT	Uint32	Display format to use for size values. Uses price type definitions from cfeti_fields.h. (e.g. if a size should be displayed to two decimal places the format CFETI_PRICETYPE_DECIMAL_D2 may be used. The default if this field is not present is CFETI_PRICETYPE_DECIMAL_D0.
CFETF_SIZE_DEC_PLACES	Uint32	Number of decimal places to display size to if size format is CFETI_PRICETYPE_DECIMAL_DVAR.
CFETF_SIZE_MIN_DEC_PLACES	Uint32	Minimum number of decimal places to display size to if size format is CFETI_PRICETYPE_DECIMAL_DVAR.
CFETF_CURRENCY_ORDINAL	Uint32	Unique integer number assigned to an FX currency that can be used to determine which currency is base and which is term in an FX spot instrument. The lower numbered currency shall always be the base and the higher numbered currency shall always be the term.
CFETF_BASE_IS_DENOMINATOR	Byte	Flag to indicate whether Price = Term Amount / Base Amount (1) or instead Price = Base Amount / Term Amount (0). This is required for the calculation of the term amount given base amount and price or vice versa for FX Spot trading.
CFETF_INSTRUMENT_CHAIN	Bytestream	Field describes a list of instruments that are somehow associated with the instrument that carries the field. (Decode with `CFETIDecodeDataField`)
CFETF_CHAIN_PARENT_LIST	Bytestream	Field describes the list of instruments that include this instrument in an instrument chain field. (Decode with `CFETIDecodeDataField`)

CFETF_DISPLAY_PROPERTIES	Uint32	Bit-mask field of instrument display properties. Possible values are defined in cfeti_consts.h (with prefix CFETF_DISPLAY_PROPERTIES_ as follows.
		`IS_RESTRICTED_INSTRUMENT`
		`IS_PORTFOLIO_EXCLUDED`
		`HAS_CHAIN`
		`CHAIN_UNRESTRICTED`
		`IS_BENCHMARK`
		A description of each property is given below.

CFETF_SQ_INST_LABEL	String	Optional field that when present is used by the eSpeed application as the text to use as the instrument description in SuperQuads.
CFETF_SQ_BENCHMARK_KEY_LABEL	String	Optional field that when present is used by the eSpeed application as the text to use as a key

		description in SuperQuads.
CFETF_SQ_SPECIFIC_KEY_LABEL	String	Optional field that when present is used by the eSpeed application as the text to use as a key description in SuperQuads.
CFETF_SQ_EGB_KEY_LABEL	String	Optional field that when present is used by the eSpeed application as the text to use as a key description in SuperQuads.
CFETF_PRIORITY_TIME_BID_OFFER	uint32	The approximate maximum time a user will hold the priority in the bid/offer state. In milliseconds.
CFETF_PRIORITY_TIME_TRADE	uint32	The approximate maximum time a user will hold the priority in the trade state. In milliseconds
CFETF_ALT_DESCRIPTION_1	String	Optional field that when present provides an alternative description of the instrument.
CFETF_ALT_DESCRIPTION_2	String	Optional field that when present provides an alternative description of the instrument.
CFETF_ALT_DESCRIPTION_3	String	Optional field that when present provides an alternative description of the instrument.
CFETF_INDICATIVE_BID	Decimal	Indicative bid price. Some systems that publish indicative prices use separate fields for the indicative bid and offer, either in addition or instead of identification of the indicative market through the field CFETF_STATE.
CFETF_INDICATIVE_OFFER	Decimal	Indicative offer price. (See notes above for INDICATIVE_BID).
CFETF_INDICATIVE_STATE	Uint16	State field to specify current state of the indicative market in the fields CFETF_INDICATIVE_BID and CFETF_INDICATIVE_OFFER. This shall always be on of the following values: CFETI_STATE_NO_MARKET CFETI_STATE_BID CFETI_STATE_OFFER CFETI_STATE_BID|CFETI_STATE_OFFER. (See notes above for INDICATIVE_BID).
CFETF_MID_PRICE	Decimal	Indicative mid price
CFETF_MID_YIELD	Decimal	Indicative mid price yield
CFETF_OUTRIGHT_BID	Decimal	Outright bid price. Some systems publish a calculated outright bid price corresponding to the price in the field CFETF_BID. Validity of the field value depends on the value of the field CFETF_STATE indicating whether or not there is a bid present.
CFETF_OUTRIGHT_ASK	Decimal	Outright offer price. Some systems publish a calculated outright offer price corresponding to the price in the field CFETF_ASK. Validity of the field value depends on the value of the field CFETF_STATE indicating whether or not there is an offer present.
CFETF_OUTRIGHT_PRICETYPE	Uint32	Enumerated price format to use when displaying the outright bid and offer
CFETF_OUTRIGHT_DEC_PLACES	Uint32	Number of decimal places to be displayed when showing the outright bid and offer. Valid only when the outright price-type is CFETI_PRICETYPE_DVAR.
CFETF_OUTRIGHT_MIN_DEC_PLACES	Uint32	Minimum number of decimal places to be displayed without truncating the price when showing the outright bid and offer. Valid only when the outright price-type is CFETI_PRICETYPE_DVAR.

CFETF_SESSION_BOUNDRY_24H	Bytestream	Where available, this field describes the high/low/open/close for the 24 Hour trading session. See 2.18.3.5 for details. (Decode using CFETIDecodeDataField).
CFETF_SESSION_BOUNDRY_ASIA	Bytestream	Where available, this field describes the high/low/open/close for the Asia trading session. See 2.18.3.5 for details. (Decode using CFETIDecodeDataField).
CFETF_SESSION_BOUNDRY_EUROPE	Bytestream	Where available, this field describes the high/low/open/close for the London trading session. See 2.18.3.5 for details. (Decode using CFETIDecodeDataField).
CFETF_SESSION_BOUNDRY_US_CASH	Bytestream	Where available, this field describes the high/low/open/close for the US Cash market trading session. See 2.18.3.5 for details. (Decode using CFETIDecodeDataField).
CFETF_SESSION_BOUNDRY_US_FUTURES	Bytestream	Where available, this field describes the high/low/open/close for the US Futures market trading session. See 2.18.3.5 for details. (Decode using CFETIDecodeDataField).
CFETF_SESSION_BOUNDRY_NET_24H	Bytestream	Where available, this field describes the net price change for the 24 Hour trading session. See 2.18.3.6 for details. (Decode using CFETIDecodeDataField).
CFETF_SESSION_BOUNDRY_NET_ASIA	Bytestream	Where available, this field describes the net price change for the Asia trading session. See 2.18.3.6 for details. (Decode using CFETIDecodeDataField).
CFETF_SESSION_BOUNDRY_NET_EUROPE	Bytestream	Where available, this field describes the net price change for the London trading session. See 2.18.3.6 for details. (Decode using CFETIDecodeDataField).
CFETF_SESSION_BOUNDRY_NET_US_CASH	Bytestream	Where available, this field describes the net price change for the US Cash market trading session. See 2.18.3.6 for details. (Decode using CFETIDecodeDataField).
CFETF_SESSION_BOUNDRY_NET_US_FUTURES	Bytestream	Where available, this field describes the net price change for the US Futures market trading session. See 2.18.3.6 for details. (Decode using CFETIDecodeDataField).
CFETF_VWA_SESSION_BOUNDRY_24H	Bytestream	Where available, this field describes the volume weighted average for the 24 Hour trading session. See 2.18.3.7 for details. (Decode using CFETIDecodeDataField).
CFETF_VWA_SESSION_BOUNDRY_ASIA	Bytestream	Where available, this field describes the volume weighted average for the Asia trading session. See 2.18.3.7 for details. (Decode using CFETIDecodeDataField).
CFETF_VWA_SESSION_BOUNDRY_EUROPE	Bytestream	Where available, this field describes the volume weighted average for the London trading session. See 2.18.3.7 for details. (Decode using CFETIDecodeDataField).
CFETF_VWA_SESSION_BOUNDRY_US_CASH	Bytestream	Where available, this field describes the volume weighted average for the US Cash market trading session. See 2.18.3.7 for details. (Decode using CFETIDecodeDataField).
CFETF_VWA_SESSION_BOUNDRY_US_FUTURES	Bytestream	Where available, this field describes the volume weighted average for the US Futures market trading session. See 2.18.3.7 for details. (Decode using CFETIDecodeDataField).

CFETF_BENCHMARK_INST_SPEC	Bytestream	This field shall identify an instrument used to publish underlying rates for the instrument that carries this field. (e.g. for an FX option the referenced instrument shall publish the associated FX Spot instrument. The field shall be decoded by applications that receive it using the CFETIDecodeDataField interface. (See CFETI_COMPOUND_INST_LIST)
CFETF_MARKET_AVAILABILITY_NOTIFICATION	String	This field indicates that the market availability has been updated. This field shall be decoded by applications that receive it using the CFETIDecodeDataField interface (See CFETI_MARKET_AVAILABILITY_DESC).
CFETF_MINIMUM_MID_SIZE	Decimal	The minimum size that must be specified on a mid-price order. If the minimum size mid size field is not available the minimum size for a mid-price order must be the same as the minimum size for the instrument.
CFETF_MID_TRADE_SIZE	Decimal	The currently executing mid size. If this value is non-zero then this field indicates a mid-price trade in the size indicated.
CFETF_INSTANT_EXECUTION_DELTA	Decimal	Delta from the touch price at which instant executions shall be displayed. This shall be a decimal value that is the amount of a price tick multiplied by the number if price tiers at which instant execution is indicated.
CFETF_TRADE_PREF	Uint32	Bit-wise combination of the market/order preferences supported for this instrument.
CFETF_TICKER_SYMBOL	String	Ticker symbol for the instrument. Links related instruments to the issuer.
CFETF_CALL_DATE	Uint32 or DateTime	The next date, prior to maturity, on which a callable bond may be redeemed. The Uint32 format is the preferred one, and should be in the format CCYYMMDD.
CFETF_IRS_INST_PROPERTY	Uint32	Specific to interest rate derivatives instruments. Bit-wise combination of properties identifying IRS-specific attributes of the instrument
CFETF_INST_SELECTION_PROPERTIES	Uint32	Bit-wise combination of properties identifying attributes of the instrument used for instrument selection.
CFETF_BID_LOCK_1 to CFETF_BID_LOCK_10	Decimal	Selected lock price for the corresponding price tier, displayed using the same price format rule as the prevailing lock price. Valid if the instrument is in a bid or sell state at this tier.
CFETF_ASK_LOCK_1 to CFETF_ASK_LOCK_10	Decimal	Selected lock price for the corresponding price tier, displayed using the same price format rule as the prevailing lock price. Valid if the instrument is in a offer or buy state at this tier.
CFETF_AT_MARKET_BID_1 to CFETF_AT_MARKET_BID_10	Decimal	Un-rounded bid price for the corresponding price tier, displayed using the same price format rule as the bid prices. Valid if the instrument is in a bid or sell state at this tier.
CFETF_AT_MARKET_ASK_1 to CFETF_AT_MARKET_ASK_10	Decimal	Un-rounded offer price for the corresponding price tier, displayed using the same price format rule as the offer prices. Valid if the instrument is in an offer or buy state at this tier.
CFETF_BID_CHANGE	Decimal	Difference between the bid price and the closing price. Valid only if instrument is in a bid state.
CFETF_ASK_CHANGE	Decimal	Difference between the offer price and the closing price. Valid only if instrument is in an offer state.

CFETF_DEAL_STRUCTURE	Uint32	Enumerated deal structure. Possible values are defined in cfeti_consts.h with prefix CFETI_DEAL_STRUCTURE_.
CFETF_TRADE_TYPE	Uint32	Enumerated trade type. Possible values are defined in cfeti_consts.hy with prefix CFETI_TRADE_TYPE_.

The field CFETF_PRICE_TICK_RULES indicates rules for price increment for US REPO instruments. This can be one of the following values:

Rule	Description
CFETI_FIXED	This is the default.
CFETI_USREPO1 CFETI_BRADY1	

The field CFETF_PRICING_METHOD indicates the method used to price the instrument. This can be one of the following values:

Method
CFETI_PRICING_METHOD_PRICE
CFETI_PRICING_METHOD_BASIS
CFETI_PRICING_METHOD_PRICE_SPREAD
CFETI_PRICING_METHOD_YIELD
CFETI_PRICING_METHOD_YIELD_SPREAD
CFETI_PRICING_METHOD_VOLATILITY

The fields CFETF_STATE_1 though CFETF_STATE_10 fields indicate the current trading state of the instrument at the corresponding price tier. This can be a combination of the following states:

State	Description
CFETI_STATE_NO_MARKET	There are no bids or offers for this instrument.
CFETI_STATE_BID	There is at least one bid on the system for this instrument.
CFETI_STATE_OFFER	There is at least one offer on the system for this instrument.
CFETI_STATE_UNCLEAR_BID	The first customer on the offer is able to sell at the bid price while the bid is unclear. A sell order submitted by any other customer can only be filled when the bid is no longer unclear. The time for which the bid is unclear is determined by the trading system.
CFETI_STATE_UNCLEAR_OFFER	The first customer on the bid is able to buy at the offer price while the offer is unclear. A buy order submitted by any other customer can only be filled when the offer is no longer unclear. The time for which the offer is unclear is determined by the trading system.
CFETI_STATE_BUY	Instrument is currently trading on the offer side.
CFETI_STATE_SELL	Instrument is currently trading on the bid side.

CFETI_STATE_CHECK_CREDIT	Signifies that the aggressor in the trade is subject to credit check (i.e. when the aggressor has given verbal instruction to trade rather than electronic).
CFETI_STATE_TRADE_ENDING	Combined with other flags in trading state to indicate that a trade is about to end. The flag may be cleared in a subsequent update if further matches are made during the current trade.
CFETI_STATE_INDICATIVE_BID	The bid that is present is indicative and there is no tradable bid at this price. (Note: CFETI_STATE_BID shall also be set when there is an indicative bid).
CFETI_STATE_INDICATIVE_OFFER	The offer that is present is indicative and there is no tradable offer at this price. (Note: the state CFETI_STATE_OFFER shall also be set when there is an indicative offer).
CFETI_STATE_ERROR	This signifies that the currently executing trade is in error. The original bid/offer lists are reinstated and the instrument remains in an error state until either a pre-configured time elapses or new markets or orders are received for this instrument.
CFETI_STATE_NON_TRADABLE	No trades can be executed on the instrument.

Macros are provided in `cfeti.h` to retrieve state from a `CFETF_STATE` market data field. Each macro takes the value of a `CFETF_STATE` market data field as its argument. The macros provided are as follows:

Macro	Description
CFETI_IS_NO_MARKET	Instrument is in no market state.
CFETI_IS_BID	Bid flag is set.
CFETI_IS_OFFER	Offer flag is set.
CFETI_IS_BID_OR_OFFER	At least one of bid and offer flag is set.
CFETI_IS_BID_AND_OFFER	Both bid and offer flags are set.
CFETI_IS_UNCLEAR_BID	Bid flag and Unclear Bid flag is set.
CFETI_IS_UNCLEAR_OFFER	Offer flag and Unclear Offer flag is set.
CFETI_IS_BUY	Buy flag is set.
CFETI_IS_SELL	Sell flag is set.
CFETI_IS_TRADING	At least one of the buy and sell flags is set.
CFETI_IS_TRADE_ENDING	Trade-ending flag is set. Applications should also test trading state.
CFETI_IS_ERROR	Instrument is in error state.
CFETI_IS_TRADABLE	Non-tradable flag is not set.
CFETI_BID_IS_INDICATIVE	Bid flag and indicative bid flags are both set.
CFETI_OFFER_IS_INDICATIVE	Offer flag and indicative offer flags are both set.

The fields CFETF_STATE_EXT_1 though CFETF_STATE_EXT_10 fields indicate additional attributes of the trading state of the instrument at the corresponding price tier. This can be a combination of the following states:

State	Description
CFETI_STATE_EXT_LAST_BID	The current bid price is a last bid. The corresponding state field shall include the bits CFETI_STATE_BID and CFETI_STATE_INDICATIVE_BID.
CFETI_STATE_EXT_LAST_OFFER	The current offer price is a last offer. The corresponding state field shall include the bits CFETI_STATE_OFFER and CFETI_STATE_INDICATIVE_OFFER.

The bid and offer list fields CFETF_BID_LIST_1 to CFETF_BID_LIST_10 and CFETF_ASK_LIST_1 to CFETF_ASK_LIST_10 contain lists of market participants that can be decoded using CFETIDecodeDataField. The price code element of the decoded participants is a bit-mask that may be set to a combination of values and tested using the macros indicated below.

Code	Macro	Description
CFETI_PRICE_GOOD_TILL_CANCEL	CFETI_PRICE_IS_GOOD_TIL L_CANCEL	The price is a firm price.
CFETI_PRICE_ALL_OR_NO NE	CFETI_PRICE_IS_ALL_OR_N ONE	The price is an all-or-none price.
CFETI_PRICE_LIMIT	CFETI_PRICE_IS_LIMIT	The price is a limit price.
CFETI_PRICE_CHECK_CRE DIT	CFETI_PRICE_IS_CHECK_CR EDIT	The price is submitted by verbal instruction, not electronic for a giveup business.
CFETI_PRICE_OPTION_1		The order preference CFETI_TRADE_OPTION_1 was specified for the market/order.
CFETI_PRICE_OPTION_2		The order preference CFETI_TRADE_OPTION_2 was specified for the market/order.
CFETI_PRICE_OPTION_3		The order preference CFETI_TRADE_OPTION_3 was specified for the market/order.
CFETI_PRICE_OPTION_EX CLUDE	CFETI_PRICE_IS_EXCLUDED	The price and size is not good for the decoding user. Per configuration for eSpeed FX.

For European Repos the price options are used to indicate the mechanisms that can be used to clear the trade. Synonyms for the price options defined above are provided as follows and correspond one-to-one with the trade options specified for this business.

```
CFETI_PRICE_OPTION_1        CFETI_PRICE_CLEARING_LCH
CFETI_PRICE_OPTION_2        CFETI_PRICE_CLEARING_CLEARNET
CFETI_PRICE_OPTION_3        CFETI_PRICE_CLEARING_INTERBANK
```

For Interest Rate Derivatives the first and third price options are used to indicate the mechanisms that can be used to clear the trade. The second price option is used to indicate that the market is a derived spread price. Synonyms for the price options defined above are provided as follows and correspond one-to-one with the trade options specified for this business.

```
CFETI_PRICE_OPTION_1        CFETI_PRICE_CLEARING_LCH
CFETI_PRICE_OPTION_2        CFETI_PRICE_DERIVED_SPREAD
CFETI_PRICE_OPTION_3        CFETI_PRICE_CLEARING_INTERBANK
```

For US Treasuries the third price option is used to indicate that the entry in the participant list is a mid price.

```
CFETI_PRICE_OPTION_3        CFETI_PRICE_MID
```

For eSpeed FX the first price option is used to indicate that the entry in the participant list is contributed by a market maker.

CFETI_PRICE_OPTION_1 CFETI_PRICE_OPTION_MME

The field CFETF_ALT_INST_1 holds the name of an instrument that is associated with the one containing the field. This is an instrument that can be subscribed to in the same way as any other instrument. For example, in a Basis instrument the field may contain the name of the Future.

By default prices are in 32nds unless the CFETF_PRICETYPE field exists in which case the price type – and hence the rules for the display of that price - shall be determine from its value. Possible values are listed in the field below.

CFETF_PRICETYPE	Description
CFETI_PRICETYPE_32ND	32nds
CFETI_PRICETYPE_DECIMAL	Synonym for CFETI_PRICETYPE_DECIMAL_D4
CFETI_PRICETYPE_DECIMAL_D0	Integer prices
CFETI_PRICETYPE_DECIMAL_D1	NNN.N
CFETI_PRICETYPE_DECIMAL_D2	NNN.NN
CFETI_PRICETYPE_DECIMAL_D2_PLUS	NNN.NN[+] (optional + = 0.005)
CFETI_PRICETYPE_DECIMAL_D3	NNN.NNN
CFETI_PRICETYPE_DECIMAL_D4	NNN.NNNN
CFETI_PRICETYPE_DECIMAL_F4	NNN.[246] (1/8ths)
CFETI_PRICETYPE_DECIMAL_F8	NNN.[1-7] (1/8ths)
CFETI_PRICETYPE_DECIMAL_F16	NNN.NN (00-16)
CFETI_PRICETYPE_DECIMAL_F32	NNN.NN (00-31)
CFETI_PRICETYPE_DECIMAL_F64	NNN.NN[+] (00-31)
CFETI_PRICETYPE_DECIMAL_F128	NNN.NN[2+6]
CFETI_PRICETYPE_DECIMAL_F256_8TH_32ND	NNN.NN[123+567]
CFETI_PRICETYPE_DECIMAL_D4_PLUS	If the price is less than 10, display the price with maximum 4 decimal places plus optional + to represent 0.00005. Otherwise display the price with maximum 3 decimal places plus optional + to represent 0.005.
CFETI_PRICETYPE_DECIMAL_DVAR	Price should be displayed to N decimal places. After M decimal places, if trailing digits are zero they should be replaced with space. N,M are specified by fields CFETF_PRICE_DEC_PLACES and CFETF_PRICE_MIN_DEC_PLACES, or CFETF_SETTLEMENT_PRICE_DEC_PLACES and CFETF_SETTLEMENT_PRICE_MIN_DEC_PLACES for a settlement price.
CFETI_PRICETYPE_DECIMAL_D2_BILL	NNN.NN[1,2(+),3]
CFETI_PRICETYPE_DECIMAL_D2_WI	NNN.NN[2,4(+),6]
CFETI_PRICETYPE_DECIMAL_D2_BILL_256	NNN.NN[1,2(+),3]
CFETI_PRICETYPE_DECIMAL_D2_WI_256	NNN.NN[1,2,3,4(+),5,6,7]
CFETI_PRICETYPE_MMTS_F256	NNN.NN[123+567] where NN denotes decimal places 00 to 99 and [123+567] partial ticks.

Macros are provided to test for 32nds and decimal – CFETI_PRICETYPE_IS_32ND() and CFETI_PRICETYPE_IS_DECIMAL() respectively. The same enumeration is also used for the display price type CFETF_DISPLAY_PRICETYPE.

The following possible values are defined for the bit-mask field CFETF_DISPLAY_PROPERTIES

Constant	Description
CFETI_DISPLAY_PROPERTIES_IS_RESTRICTED_INSTRUMENT	The eSpeed front-end application shall prevent subscription to this instrument in the trading grids.
CFETI_DISPLAY_PROPERTIES_IS_PORTFOLIO_EXCLUDED	The eSpeed front-end-application shall prevent users from saving this instrument in their portfolio files.
CFETI_DISPLAY_PROPERTIES_HAS_CHAIN	The instrument has a chain defined in the field CFETF_INSTRUMENT_CHAIN.
CFETI_DISPLAY_PROPERTIES_CHAIN_UNRESTRICTED	The instrument chain display specifications are not enforced by the eSpeed front-end application.
CFETI_DISPLAY_PROPERTIES_IS_BENCHMARK	The instrument is designated to be a benchmark instrument.

The following possible values are defined for the bit-mask field CFETF_IRS_INST_PROPERTY.

Constant (prefix CFETI_IRS_PROPERTY_)	Description
OUTRIGHT	IRS outright instrument
TRACKING	IRS tracking instrument
CROSS	IRS off-market-cross instrument
BASIS	IRS basis instrument
YIELD_SPREAD	IRS yield spread
VS_1M_FLOATING	Vs. 1 month floating rate
VS_3M_FLOATING	Vs. 3 month floating rate
VS_6M_FLOATING	Vs. 6 month floating rate
VS_OIS	Vs. overnight index swap rate
VS_BUND	Vs. Bund future
VS_BOBL	Vs. Bobl future
VS_SCHATZ	Vs. Schatz future
DEFAULT_DERIVE	Derivation is the default for this instrument
DEFAULT_HEDGE	Hedge is the default for this IRS instrument

The following possible values are defined for the bit-mask field CFETF_INST_SELECTION_PROPERTIES.

Constant (prefix CFETI_INST_SELECTION_)	Description
OUTRIGHT	Outright (cash) instrument
BASIS	Basis instrument
SWITCH	Spread (switch) instrument

4. Appendix C - eSpeed API Changes for BGC FX Spot Trading System

For BGX FX Spot, we have added support for three new features: iceberg orders; discretion orders; and decimalized orders. For eSpeed FX Spot, support for iceberg orders and discretion orders will be available too.

4.1 Business Requirement Description

4.1.1 Decimalized Orders

Introduce decimalized orders into FX Spot as follows:

- Allow orders to be priced in 0.1 pip increments.
- The trading system will not publish the extra decimal in the market data price.
- The trading system will continue to match with best price and time priority.

For example, user "a" places bid at 1.2605 for 1 million. Then user "b" places bid at 1.26052 for 2 million. The trading system will publish market data:

Bid price 1.2605
Bid size 3
Bid participant list
Size 1
Size 2

Then, when an offer for 1 million is submitted at price 1.2605, the bid owned by user "b" will trade for 1 million at price 1.26052 preserving the price/time matching priority.

4.1.2 Discretion Orders

Introduce discretion orders into FX Spot as follows:

- Allow orders to be priced with a hidden price discretion where the discretion is a multiple of 0.1 pip price.
- The trading system will not publish the discretion value in the market data.
- The trading system will match in the following manner:
    - A resting discretion bid will aggress a new offer when the offer price (or offer discretion) matches with the bid's discretion.
    - A resting discretion offer will aggress a new bid when the bid price (or bid discretion) matches with the offer's discretion.

Two examples follow.

Scenario one. Assume user "a" places bid at 1.2605 for 1 million. Then user "b" places bid at 1.2605 with discretion 0.2 for 2 million. The trading system will publish market data:

Bid price 1.2605
Bid size 3
Bid participant list
    Size 1
    Size 2

When an offer for 1 million is submitted at price 1.2605, the bid owned by user “a” will trade for 1 million at price 1.2605. User “a” trades first because its price didn’t need discretion to match.

Scenario two. Assume user “a” places bid at 1.2605 for 1 million. Then user “b” places bid at 1.2605 with discretion 1.0 for 2 million. The trading system will publish market data:

Bid price 1.2605
Bid size 3
Bid participant list
    Size 1
    Size 2

When an offer for 1 million is submitted at price 1.2606, the bid owned by user “b” will trade for 1 million at price 1.2606. User “b” will be the aggressor.

The difference between a discretion order and a decimalized order is how the trading system decides the sequence of execution. The rule is that decimalized orders trade first when a match exists. Then, discretion orders will trade second, if a match exists using the discretion.

4.1.3 Iceberg Orders

Introduce iceberg orders into FX Spot as follows:

- Allow orders to have visible size and hidden size.
- The trading system will not publish the hidden size in the market data.
- The trading system will match by price, visibility, and then time.
    - First all visible quantity in price/time order
    - Then, all hidden quantity in price/time order.
- After a match has completed, when an iceberg's display size has been completely traded, the trading system may or may not replenish the iceberg's display size. The replenished display size will be placed at the bottom of the visible book. The remaining hidden size will keep its place in the hidden book.

For example, user "a" places bid at 1.2605 with display size of 3 million, hidden size of 10 million and wants the iceberg order to replenish display size. Then user "b" places bid at 1.2605 for 2 million. The trading system will publish market data:

Bid price 1.2605
Bid size 5
Bid participant list
Size 3
Size 2

Later, when an offer for 4 million is submitted at price 1.2605, the bid owned by user "a" will trade for 3 million at price 1.2605 and the bud owned by user "b" will partially trade for 1 million at 1.2605. The trading system will then replenish user "a" display size for 3 million and place it at the end of the stack. The market update would then appear as:

Bid price 1.2605
Bid size 4
Bid participant list
Size 1
Size 3

4.2 API Changes.

The following describes how to post markets and orders with these features.

4.2.1 Trading System Properties

The “trading system properties” that are published upon user connection are:

- CFETI_ICEBERG_ENABLED
- CFETI_DISCRETION_ENABLED
- CFETI_DECIMALIZED_PRICE_ENABLED

Furthermore, we’ll have additional fields within the instrument data description returned from a “subscribe accepted” response.

- CFETF_ICEBERG_MINIMUM_DISPLAY_SIZE
- CFETF_DISCRETION_TICK
- CFETF_DECIMAL_TICK

4.2.2 Decimalized Orders

If CFETI_DECIMAL_ENABLED is true, then the additional variables to use within the CFETI_MARKET and CFETI_ORDER are:

- Assign the non-decimalized price to the “price” variable.
- Assign the extra decimal price to the “decimalRange” variable.

Notes:

- The decimal range is a multiple of the CFETF_DECIMAL_TICK value of the instrument’s price tick value.

The following table describes what price and decimal range to set to obtain the desired matching price.

`Given CFETF_DECIMAL_TICK value of 0.1`

Instrument	Side	Desired Price	Decimal Range	Price
GBP/USD	Bid	1.87074	4	1.8707
USD/JPY	Bid	118.582	2	118.58
EUR/GBP	Bid	0.67068	3	0.67065
GBP/USD	Ask	1.87066	4	1.8707
USD/JPY	Ask	118.578	2	118.58
EUR/GBP	Ask	0.67062	3	0.67065

4.2.3 Discretion Orders

If CFETI_DISCRETION_ENABLED is true, then the additional variables to use within the CFETI_MARKET and CFETI_ORDER are:

- Assign the price to the “price” variable.
- Assign the discretion’s range to the “discretionRange” variable.

Notes:

- The discretion range is a multiple of the CFETF_DISCRETION_TICK value of the instrument’s price tick value.

The following table describes the possible trading prices for given discretion markets.

Given CFETF_DISCRETION_TICK value of 0.1

Instrument	Side	Price	Discretion Range	Possible Trade Price
GBP/USD	Bid	1.8707	20	1.8709
GBP/USD	Bid	1.8707	4	1.87074
USD/JPY	Bid	118.58	10	118.59
USD/JPY	Bid	118.58	2	118.582
EUR/GBP	Bid	0.67065	3	0.67068
EUR/GBP	Bid	0.67065	10	0.67075
GBP/USD	Ask	1.8707	20	1.8705
GBP/USD	Ask	1.8707	4	1.87066
USD/JPY	Ask	118.58	10	118.57
USD/JPY	Ask	118.58	2	118.578
EUR/GBP	Ask	0.67065	3	0.67062
EUR/GBP	Ask	0.67065	10	0.67055

4.2.4 Iceberg Orders

If CFETI_ICEBERG_ENABLED is true, then the additional variables to use within the CFETI_MARKET and CFETI_ORDER are:

- The CFETI_ICEBERG constant must be “or’d” into the “preference2” bitmask.
- Assign the iceberg’s display size to the “reserveInitialSize” variable.
- Assign the iceberg’s replenishing size to the “reserveMinSize” variable.
- Assign the total size to the “size” variable.

Notes:

- The value of “reserveInitialSize” must exceed the trading system’s CFETF_ICEBERG_MINIMUM_DISPLAY_SIZE
- The value of “reserveMinSize” can be zero. If the reserveMinSize is non-zero, then it must exceed the CFETF_ICEBERG_MINIMUM_DISPLAY_SIZE value.

4.2.5 Other Items

If all three order enhancements are enabled, then a user may post a market, or an order, with all three features.

4.3 Sample Code Fragments

```
      // The following code are examples to clarify the new FX Spot
enhancements.
      //
      // Supported preferences for BGC FX Spot "markets"
      //    CFETI_MARKET_LIMIT_PRICE
      //    CFETI_MARKET_SIZE_IS_INCREMENTAL
      //    CFETI_MARKET_SIZE_IS_TOTAL
      //    CFETI_MARKET_CANCELS_MARKET
      //    CFETI_MARKET_NO_RESPONSE
      //    CFETI_MARKET_CANCEL_ALL_FOR_ISSUE_SAME_SIDE
      //
      // Supported preferences2 for BGC FX Spot "markets".
      //    CFETI_IS_ICEBERG
      //

      //
      // Post a GBP/USD bid with price 1.87022 for 1 million.
      //
      CFETI_RC postMarketBidDecimal (
                                    const char *MySession,
                                    CFETI_TRADE_SESS_ID MyTradingSession
                                    )
      {
          CFETI_MARKET_DESC mktbid;

          memset(((char *) &mktbid), 0, sizeof (mktbid));
          mktbid.tradeInstrument = "GBP/USD_SP";
          mktbid.side            = CFETI_MARKET_BID;
          mktbid.preferences     = CFETI_MARKET_SIZE_IS_TOTAL |
CFETI_MARKET_LIMIT_PRICE;
          mktbid.price           = 1.8702;
          mktbid.size            = 1000000.0;   // 1 million size.
          mktbid.decimalRange    = 2;           // 2 times the
CFETF_DECIMAL_TICK value.

          // Maximum 8 bytes for comments. Passed onto trade confirmations.
          mktbid.userData     = (void *) "Bob";
          mktbid.userDataSize = 4;

          CFETI_RC result =   CFETIPostMessage (
              MySession,
              MyTradingSession,
              CFETC_POST_MARKET,
              (CFETI_CMDDATA) & mktbid,
              0
              );
          return (result);
      }
```

```
     //
     // Post a EUR/SEK bid with price 9.2125 for 1 million with discretion
of 25 pips.
     //
     CFETI_RC postMarketBidDiscretion (
                                       const char *MySession,
                                       CFETI_TRADE_SESS_ID
MyTradingSession
                                       )
     {
         CFETI_MARKET_DESC mktbid;

         memset(((char *) &mktbid), 0, sizeof (mktbid));
         mktbid.tradeInstrument = "EUR/SEK_SP";
         mktbid.side            = CFETI_MARKET_BID;
         mktbid.preferences     = CFETI_MARKET_SIZE_IS_TOTAL |
CFETI_MARKET_LIMIT_PRICE;
         mktbid.price           = 9.2125;
         mktbid.size            = 1000000.0;   // 1 million size.
         mktbid.discretionRange = 250;
         // 250 times the CFETF_DISCRETION_TICK value which should be
equal to 25 pips.

         // Maximum 8 bytes for comments. Passed onto trade confirmations.
         mktbid.userData     = (void *) "Sally";
         mktbid.userDataSize = 6;

         CFETI_RC result =   CFETIPostMessage (
             MySession,
             MyTradingSession,
             CFETC_POST_MARKET,
             (CFETI_CMDDATA) & mktbid,
             0
             );
         return (result);
     }
```

```
//
// Post a EUR/USD bid iceberg market
//     with price 1.2541
//     with display size 3 million,
//     with hidden size 12 million,
//     with replenish size of 3 million.
//
CFETI_RC postMarketBidIceberg (
                                const char *MySession,
                                CFETI_TRADE_SESS_ID MyTradingSession
                                )
{
    CFETI_MARKET_DESC mktbid;

    memset(((char *) &mktbid), 0, sizeof (mktbid));
    mktbid.tradeInstrument    = "EUR/USD_SP";
    mktbid.side               = CFETI_MARKET_BID;
    mktbid.preferences        = CFETI_MARKET_SIZE_IS_TOTAL |
CFETI_MARKET_LIMIT_PRICE;
    mktbid.preferences2       = CFETI_ICEBERG;
    mktbid.price              = 1.2541;
    mktbid.size               = 15000000.0;   // The total size (3 +
12) million.
    mktbid.reserveInitialSize =  3000000.0;   // The initial display
size - must exceed
    //   CFETF_ICEBERG_MINIMUM_DISPLAY_SIZE
    mktbid.reserveMinSize     =  3000000.0;   // The replenish size.
This can be 0.0.

    // Maximum 8 bytes for comments. Passed onto trade confirmations.
    mktbid.userData     = (void *) "Edward";
    mktbid.userDataSize = 7;

    CFETI_RC result =   CFETIPostMessage (
        MySession,
        MyTradingSession,
        CFETC_POST_MARKET,
        (CFETI_CMDDATA) & mktbid,
        0
        );
    return (result);
}
```

```
      //
      // The following sample shows how one can decode a "market data
update" for a participant
      // list and determine if the corresponding size is not good to the
user.
      //
      void ProcessMarketData (
                              CFETI_INSTRUMENT_DATA cmdData,
                              const char *MySession,
                              CFETI_TRADE_SESS_ID MyTradingSession
                              )
      {
          CFETI_DECODE_DATA_DESC dataDesc;

          for (int i = 0; i < cmdData->numFields; i++)
          {
              unsigned int fieldId = cmdData->fieldTable[i].fieldId;

              if (fieldId > CFETI_LAST_FIELD)
                  continue;

              switch (fieldId)
              {
              case CFETF_BID_LIST_1:
              case CFETF_BID_LIST_2:
              case CFETF_BID_LIST_3:
              case CFETF_BID_LIST_4:
              case CFETF_BID_LIST_5:
                  {
                      CFETI_PARTICIPANT_LIST participants = NULL;
                      dataDesc.field          = &cmdData->fieldTable[i];
                      dataDesc.instrumentName = cmdData->instName;
                      dataDesc.instClass      = NULL;
                      CFETIDecodeDataField (
                          MySession,
                          MyTradingSession,
                          &dataDesc,
                          (void **)&participants);
                      if (participants != NULL)
                      {
                          for (int j = 0; j < participants-
>numParticipants; j++)
                          {
                              if (CFETI_PRICE_IS_EXCLUDED (participants-
>participant[j].code))
                              {
                                  fprintf (stdout, "Participant size is
excluded to me: %f, %s \n",
                                      participants->participant[j].size,
dataDesc.instrumentName);
                              }
                          }
                      }
                      break;
                  }

              case CFETF_ASK_LIST_1:
              case CFETF_ASK_LIST_2:
              case CFETF_ASK_LIST_3:
              case CFETF_ASK_LIST_4:
              case CFETF_ASK_LIST_5:
```

```
                {
                    CFETI_PARTICIPANT_LIST participants = NULL;
                    dataDesc.field          = &cmdData->fieldTable[i];
                    dataDesc.instrumentName = cmdData->instName;
                    dataDesc.instClass      = NULL;
                    CFETIDecodeDataField (
                        MySession,
                        MyTradingSession,
                        &dataDesc,
                        (void **)&participants);
                    if (participants != NULL)
                    {
                        for (int j = 0; j < participants-
>numParticipants; j++)
                        {
                            if (CFETI_PRICE_IS_EXCLUDED (participants-
>participant[j].code))
                            {
                                fprintf (stdout, "Participant size is
excluded to me: %f, %s \n",
                                    participants->participant[j].size,
dataDesc.instrumentName);
                            }
                        }
                    }
                    break;
                }

            default:
                break;
            }
        }
    }
```