ПРИНЦИПЫ ФУНКЦИОНИРОВАНИЯ

МОДУЛЯ ОБМЕНА ИНФОРМАЦИЕЙ ТОРГОВОЙ СИСТЕМЫ АО "КАЗАХСТАНСКАЯ ФОНДОВАЯ БИРЖА" С ВНЕШНИМИ ПРИЛОЖЕНИЯМИ

Раздел 1. ОБЩИЕ ПОЛОЖЕНИЯ

В данном документе описываются принципы работы модуля обмена информацией торговой системы АО "Казахстанская фондовая биржа" (далее – Биржа) с внешними программными приложениями версии 2.34 (далее – Модуль обмена).

Модуль обмена реализован в виде динамической библиотеки trade.dll. Для взаимодействия с указанной библиотекой используется соответствующим образом модифицированное программное обеспечение "Клиент" Торговой системы Биржи (далее – Терминал).

Модуль обмена устанавливается на том же персональном компьютере, что и Терминал, и позволяет осуществлять обмен данными с внешними по отношению к Терминалу программами через разделяемую память оперативной памяти данного компьютера (далее – разделяемая память[1]).

Обмен информацией с Торговой системой Биржи происходит путем вызова внешними программами соответствующих функций Модуля обмена.

Функции Модуля обмена позволяют внешним программам получить текущую информацию о параметрах финансовых инструментов, их котировках, сделках с финансовыми инструментами, заявках, поданных через Терминал, а также осуществить подачу заявок в Торговую систему Биржи.

В указанной версии Модуля обмена внешние программы могут использовать следующие функции, описание которых приведено в разделе 2 "Функции" настоящего документа:

1) для получения информации по инструментам:

   getInstrumentCount;

   getInstrumentByNum;

   getInstrumentByName;

   getInstrumentByNamePtr;

   getInstrumentById;

   getInstrumentByNin;

   getDohodFromPrice;

   getPriceFromDohod;

   getSystemOfQuotation;

2) для получения информации о котировках инструментов:

   getQuotList;

   getQuot;

3) для получения информации о заявках:

   getOrderCount;

   getOrderByNum;

[1] Для хранения заявок и сделок совместно с механизмом shared memory, также используется механизм "File mapping", основанный на отображении разделяемой памяти в файл. Таким образом, при сбоях и некорректном завершении процессов, а также после их перезапуска информация в shared memory восстановится из соответствующих файлов (используются файлы с расширением .map в каталоге Windows). Поэтому, если требуется запустить процессы без устаревшей информации, достаточно удалить соответствующие .map файлы.

getOrderById;

getNextOrder;

getOrderByOrderNum;

для получения информации о сделках:

getDealCount;

getDealByNum;

getDealById;

getNextDeal;

4) для получения информации о репо-обязательствах:

getRepoInfoCount;

getRepoInfoByNum;

getRepoInfoById;

5) для получения информации о транзитных приказах:

getTransitCount;

getTransitByNum;

getTransitById;

getNextTransit;

6) для передачи в торговую систему заявок, запросов на удаление заявок, выяснения состояния такой передачи:

putOrder;

putOrderPtr;

getOrderStatus;

getOrderByNum;

getOrderById;

getNextOrder;

putDelOrder;

getDelOrderStatus;

putConfOrder;

getConfOrderStatus;

putSatisfyOrder;

getSatisfyOrderStatus;

7) для передачи в торговую систему сообщений о подтверждении сделки и выяснения состояния такой передачи:

putDealConfirm;

getDealConfirmStatus;

8) для передачи в торговую систему сообщений о готовности к исполнению сделки закрытия репо и выяснения состояния такой передачи:

putRepoConfirm;

getRepoConfirmStatus;

9) для подтверждения транзитных приказов:

putTransitConfirm;

getTransitConfirmStatus;

10) для очистки соответствующих списков:

clearOrderList;

clearTransitList;

clearDealList;

11) для получения системной информации:

getClientVersion;

getDllVersion;

getTime;

getDate;

getUserNick;

12) для скачивания файлов отчетов:

askFiles;

getFilesStatus.

Раздел 2. ФУНКЦИИ

В данном разделе приведено подробное описание функций, вызываемых внешними программными приложениями, условия их действия и результаты, возвращаемые ими.

Работа с памятью

2.1 Объекты:

Модуль обмена позволяет получать информацию о следующих объектах Торговой системы:

1) инструменты;

2) заявки (все типы заявок);

3) сделки;

4) котировки;

5) репо–обязательства;

6) транзитные приказы.

Информация, получаемая об объектах Торговой системы, содержит информацию об объектах, относящимся ко всем секторам рынков Торговой системы, доступных пользователю Терминала, под идентификатором и паролем которого было установлено соединение с Торговой системой.

2.2 Списки объектов:

По каждому объекту Торговой системы в результате вызова функций по получению количества записей того или иного объекта Торговой системы (getInstrumentCount, getOrderCount, getDealCount и т.д.), в разделяемой памяти создаются списки объектов, в которых каждая запись представляет собой указатель на определенную структуру данных объекта (например, инструмент или заявка).

Списки существуют в разделяемой памяти для дальнейших обращений к нему функций Модуля обмена на все время работы Терминала.

2.3 Обновление списков:

Количество записей в списках в каждый момент времени работы Торговой системы может быть различным.

Информация в списках автоматически обновляется, в случае появления дополнительных записей объектов в определенный момент времени в Торговой системе, за исключением объекта котировки, обновление информации по которому осуществляется при каждом вызове функции Модуля обмена, запрашивающей количество записей по объекту котировки (getQuotList).

Для получения точного значения количества записей в списках объектов Торговой системы на определенный момент времени, внешние программы должны осуществлять в такой момент времени вызов функций Модуля обмена, выдающих информацию о количестве записей имеющихся на этот момент времени.

2.4 Удаление информации из списков:

Информация о заявках, сделках и транзитных приказах, помещаемая в разделяемую память, не удаляется из нее, в том числе и после ее прочтения внешними программами.

Для удаления информации о заявках, сделках и транзитных приказах из разделяемой памяти, внешними программами должны осуществляться вызовы специальных функций Модуля обмена (clearOrderList, clearTransitList, clearDealList), которые позволяют удалять информацию о данных объектах из разделяемой памяти, появившуюся в ней до указанной внешними программами даты. Другими словами, списки указанных объектов накапливаются до вызова специальных функций Модуля обмена по их удалению. Например, в массиве записей сделок может накопиться информация за несколько торговых дней и внешняя программа должна самостоятельно определить: когда и какие ненужные записи–сделки удалить из списка в разделяемой памяти.

Информация об инструментах и репо–обязательствах удаляется из списков автоматически, в случае удаления соответствующей информации в определенный момент времени из Торговой системы.

Информация о котировках удаляется из разделяемой памяти автоматически после ее прочтения внешними программами.

2.5 Восстановление разделяемой памяти:

После перезапуска Терминала списки, хранившиеся в разделяемой памяти на момент окончания сеанса работы Терминала с Торговой системой, восстанавливаются с учетом обновлений произошедших в Торговой системе с момента последнего сеанса работы Терминала.

При этом информация в списках: заявки, сделки и транзитные приказы восстанавливается в соответствии с текущим состоянием соответствующих данным объектам таблиц Торговой системы.

Функции, используемые внешними программами

2.6 Заявки

1) Определение количества заявок:

long ***getOrderCount****()*

Данная функция возвращает количество заявок, доступных из Терминала и поданных любым способом (введенных вручную, через функции trade.dll, в том числе с других Терминалов).

2) Получение информации о заявке по номеру в списке:

struct Order ***getOrderByNum****(long)*

Данная функция возвращает информацию о заявке (структура "Order") по порядковому номеру от "0" до "N-1" (где N=getOrderCount()).

3) Получение информации о следующей заявке:

struct Order ***getNextOrder****(long)*

Данная функция возвращает информацию о заявке с наименьшим идентификатором, который больше идентификатора заявки указанного в параметре. В случае если структуры "Order" с таким "Id" нет, то функция возвращает структуру "Order" с пустыми полями.

4) Получение информации о заявке по ее коду:

struct Order ***getOrderById****(long)*

Данная функция возвращает информацию о заявке (структура "Order") по ее идентификатору ("Id"). В случае если заявки с указанным "Id" нет, то функция возвращает структуру "Order" с пустыми полями.

5) Получение информации о заявке по коду, указываемому внешней программой:

struct Order ***getOrderByOrderNum****(long)*

Данная функция возвращает информацию о заявке (структура "Order") по ее номеру. В случае если заявки с таким номером нет, то функция возвращает структуру "Order" с пустыми полями.

6) Подача заявки (по значению)[7] :

char ***putOrder****(struct Order)*

Данная функция помещает запрос на подачу заявки (структура "Order") в разделяемую память. Для корректного функционирования функции достаточно заполнить одно из полей структуры "Order": "ShortName" или "Id". При этом если запрос успешно помещен в разделяемую память, то функция возвращает "0", в случае же если места в разделяемой памяти нет, то возвращается "1".

7) Подача заявки (по ссылке):

char ***putOrderPtr****(struct Order&)*

Данная функция аналогична функции getOrderCount, описание которой приведено в подпункте 1) настоящего пункта за исключением того, что передача параметров осуществляется по ссылке.

8) Получение статуса запроса на подачу заявки:

long ***getOrderStatus****(long)*

Данная функция возвращает код состояния запроса на подачу заявки (размещенного в разделяемой памяти функциями putOrder или putOrderPtr) по его идентификатору (описание кодов состояний приведено в пункте 2.15 настоящего документа).

9) Удаление заявки по ее номеру:

char ***putDelOrder****(struct DelOrder)*

Данная функция помещает запрос на удаление заявки (структура "DelOrder") по ее идентификатору (идентификатор содержится в структуре "DelOrder") в разделяемую память. При этом если запрос успешно помещен в разделяемую память, то функция возвращает "0", в случае же если места в разделяемой памяти нет, то возвращается "1".

10) Получение статуса запроса на удаление заявки по ее номеру:

long ***getDelOrderStatus****(long)*

Данная функция возвращает код состояния запроса на удаление заявки (размещенного в разделяемой памяти функцией putDelOrder) по ее идентификатору (описание кодов состояний приведено в пункте 2.15 настоящего документа).

11) Подтверждение заявки по ее номеру или по серийнику:

char ***putConfOrder****(struct ConfOrder)*

Данная функция помещает запрос на подтверждение заявки (структура "ConfOrder") в разделяемую память. При этом если запрос успешно помещен в разделяемую память,

[7] Подача заявок происходит следующим образом. С помощью функции **putOrder** запрос на передачу заявки помещается в shared memory с кодом статуса "err_Waiting". Если торговая система не обнаруживает ошибок, она удаляет этот запрос из shared memory, после чего вызов функции **getOrderStatus** будет возвращать статус "err_NoError". Если же обнаружена ошибка, то ее код заносится в поле "Status" запроса. В этом случае запрос будет удален из shared memory только после того, как внешняя программа считает этот код с помощью функции **getOrderStatus**.

то функция возвращает "0", в случае же если места в разделяемой памяти нет, то возвращается "1".

12) Получение статуса запроса на подтверждение заявки по ее номеру:

long ***getConfOrderStatus****(long)*

Данная функция возвращает код состояния запроса на подтверждение заявки (размещенного в разделяемой памяти функцией putConfOrder) по ее идентификатору (описание кодов состояний приведено в пункте 2.15 настоящего документа).

13) Удовлетворение заявки по ее номеру или по серийнику:

char ***putSatisfyOrder****(struct SatisfyOrder)*

Данная функция помещает запрос на удовлетворение заявки (структура "SatisfyOrder") в разделяемую память. При этом если запрос успешно помещен в разделяемую память, то функция возвращает "0", в случае же если места в разделяемой памяти нет, то возвращается "1".

14) Получение статуса запроса на удовлетворение заявки по ее номеру:

long ***getSatisfyOrderStatus****(long)*

Данная функция возвращает код состояния запроса на удовлетворение заявки (размещенного в разделяемой памяти функцией putConfOrder) по ее идентификатору (описание кодов состояний приведено в пункте 2.15 настоящего документа).

2.7 Инструменты

1) Определение количества доступных инструментов:

long ***getInstrumentCount****()*

Данная функция возвращает количество инструментов, доступных для торговли, котировки по которым находятся в разделяемой памяти.

2) Получение информации об инструменте по номеру инструмента в списке:

struct ***Instrument getInstrumentByNum****(long)*

Данная функция возвращает информацию об инструменте (структура "Instrument") по его порядковому номеру от "0" до "N-1" (где N=getInstrumentCount()).

3) Получение информации об инструменте по его краткому наименованию:

strust Instrument ***getInstrumentByName****(char ShortName[15])*

Данная функция возвращает информацию об инструменте (структура "Instrument") из разделяемой памяти по его краткому наименованию. При этом если наименование инструмента занимает менее чем 15 символов, то остальные символы в "ShortName" должны быть равны "\0". В случае если инструмента с таким кратким наименованием нет, то функция возвращает структуру "Instrument" с пустыми полями.

4) Получение информации об инструменте (по ссылке):

void ***getInstrumentByNamePtr****(char ShortName[15], Instrument&)*

Данная функция возвращает информацию об инструменте по ссылке (структура "Instrument").

5) Получение информации об инструменте по коду инструмента:

struct Instrument ***getInstrumentById****(long)*

Данная функция возвращает информацию об инструменте (структура "Instrument") по идентификатору ("Id") инструмента. В случае если структуры "Instrument" с таким "Id" нет, то функция возвращает структуру "Instrument" с пустыми полями.

6) Получение информации об инструменте по его НИН:

strust Instrument ***getInstrumentByNin****(char NIN[15])*

Данная функция возвращает информацию об инструменте (структура "Instrument") из разделяемой памяти по НИН инструмента. При этом если если НИН занимает менее чем 15 символов, то остальные символы в "NIN" должны быть равны "\0". В случае если инструмента с таким НИН нет, то функция возвращает структуру "Instrument" с пустыми полями.

7) ***Получение доходности по цене инструмента:***

double ***getDohodFromPrice****(long instrid, double price),*

где instrid – идентификатор инструмента, price – его цена.

Данная функция возвращает информацию о доходности инструмента по его идентификатору и цене. В случае если инструмента с указанным идентификатором нет либо отсутствует доходность по данному инструменту, то функция возвращает -1.

8) ***Получение цены по доходности инструмента:***

double ***getPriceFromDohod****(long instrid, double dohod),*

где instrid – идентификатор инструмента, dohod – его доходность.

Данная функция возвращает информацию о цене инструмента по его идентификатору и цене. В случае если инструмента с указанным идентификатором нет либо отсутствует доходность по данному инструменту, то функция возвращает -1.

9) ***Получение информации о предмете котирования инструмента:***

int ***getSystemOfQuotation*** *(long instrid),*

где instrid – идентификатор инструмента.

Данная функция возвращает информацию о предмете котирования инструмента по его идентификатору:

1 – Валюта котирования

2 – Чистая цена в процентах

3 – Грязная цена в процентах

4 – Инструмент авто-репо

5 – Инструмент заблокирован

В случае если инструмента с указанным идентификатором нет, то функция возвращает -1.

2.8 Котировки

1) ***Определение количества котировок:***

long ***getQuotList****(long)*

Данная функция посылает запрос на получение количества записей в списке котировок по идентификатору ("Id") инструмента и возвращает количество записей в списке котировок.

2) ***Получение котировки из списка котировок:***

struct ***ShortQuot getQuot****(long)*

Данная функция возвращает котировку (структура "ShortQuot") из списка котировок по ее порядковому номеру от "0" до "N-1" (где N=getShortQoutList(long)).

2.9 Сделки

1) ***Определение количества сделок:***

long ***getDealCount****()*

Данная функция возвращает количество сделок, доступных данному пользователю из клиентского приложения.

2) ***Получение информации о сделке по номеру в списке:***

struct Deal ***getDealByNum****(long)*

Данная функция возвращает информацию о сделке (структура "Deal") по ее порядковому номеру от "0" до "N-1" (где N=getDealCount()).

3) ***Получение информации о сделке по ее коду:***

struct Deal ***getDealById****(long)*

Данная функция возвращает информацию о сделке (структура "Deal") по ее идентификатору ("Id"). В случае если структуры "Deal" с указанным "Id" нет, то функция возвращает структуру "Deal" с пустыми полями.

4) ***Получение информации о следующей сделке:***

struct Deal ***getNextDeal****(long)*

Данная функция возвращает информацию о сделке с наименьшим идентификатором, большим нежели указанный в параметре. В случае если структуры "Deal" с наименьшим "Id" нет, то функция возвращает структуру "Deal" с пустыми полями.

5) ***Подтверждение сделки по ее номеру:***

char ***putDealConfirm****(struct DealConfirm)*

Данная функция помещает запрос на подтверждение сделки (структура "DealConfirm") по ее идентификатору (идентификатор содержится в структуре "DealConfirm") в разделяемую память. При этом если запрос успешно помещен в разделяемую память, то функция возвращает "0", в случае же если места в разделяемой памяти нет, то возвращается "1".

6) ***Получение статуса запроса на подтверждение сделки по ее номеру:***

long ***getDealConfirmStatus****(long)*

Данная функция возвращает код состояния запроса на подтверждение сделки (размещенного в разделяемой памяти функцией putDealConfirm) по ее идентификатору (описание кодов состояний передачи приведено в пункте 2.15 настоящего документа).

2.10 Репо-обязательства

1) ***Определение количества репо-обязательств:***

long ***getRepoInfoCount****()*

Данная функция возвращает количество репо-обязательств, доступных из клиентского приложения.

2) ***Получение информации о репо-обязательстве по номеру в списке:***

struct RepoInfo ***getRepoInfoByNum****(long)*

Данная функция возвращает информацию о репо-обязательстве (структура "RepoInfo") по его порядковому номеру от "0" до "N-1" (где N=getRepoInfoCount()).

3) ***Получение информации о репо-обязательстве по его коду:***

struct RepoInfo ***getRepoInfoById****(long)*

Данная функция возвращает информацию о репо-обязательстве (структура "RepoInfo") по его идентификатору ("Id"). В случае если структуры "RepoInfo" с таким "Id" нет, то функция возвращает структуру "RepoInfo" с пустыми полями.

4) ***Сообщение о готовности исполнения сделки закрытия репо:***

char ***putRepoConfirm****(struct RepoConfirm)*

Данная функция помещает запрос на подтверждение сделки закрытия репо в разделяемую память. При этом если запрос успешно помещен в разделяемую память, то функция возвращает "0", в случае же если места в разделяемой памяти нет, то возвращается "1".

5) ***Получение статуса сообщения о готовности исполнения сделки закрытия:***

long ***getRepoConfirmStatus****(long)*

Данная функция возвращает код состояния сообщения о готовности исполнения сделки закрытия репо (размещенного в разделяемой памяти функцией putRepoConfirm) по его идентификатору (описание кодов состояний передачи приведено в пункте 2.15 настоящего документа).

2.11 Транзитные приказы

1) ***Определение количества транзитных приказов:***

long ***getTransitCount****()*

Данная функция возвращает количество транзитных приказов, доступных из клиентского приложения и поданных любым способом (введенных вручную, через функции этой dll, в том числе с других рабочих мест)

2) ***Получение информации о транзитном приказе по номеру в списке:***

struct Transit ***getTransitByNum****(long)*

Данная функция возвращает информацию о транзитном приказе (структура "Transit") по его порядковому номеру от "0" до "N-1" (где N=getTransitCount()).

3) ***Получение информации о транзитном приказе по его коду:***

struct Transit ***getTransitById****(long)*

Данная функция возвращает информацию о транзитном приказе (структура "Transit") по его идентификатору ("Id"). В случае если структуры "Transit" с таким "Id" нет, то функция возвращает структуру "Transit" с пустыми полями.

4) ***Получение информации о следующем приказе:***

struct Transit ***getNextTransit****(long)*

Данная функция возвращает информацию о приказе с наименьшим идентификатором, большим нежели указанный в параметре. В случае если структуры "Transit" с таким "Id" нет, то функция возвращает структуру "Transit" с пустыми полями.

5) ***Подача подтверждения транзитного приказа:***

char ***putTransitConfirm****(struct TransitConfirm)*

Данная функция помещает запрос на подтверждение транзитного приказа (структура "TransitConfirm") в разделяемую память. При этом если запрос успешно помещен в разделяемую память, то функция возвращает "0", в случае же если места в разделяемой памяти нет, то возвращается "1".

6) ***Получение статуса подтверждения транзитного приказа:***

long ***getTransitConfirmStatus****(long)*

Данная функция возвращает код состояния подтверждения транзитного приказа (размещенного в разделяемой памяти функцией putTransitConfirm) по его идентификатору (описание кодов состояний приведено в пункте 2.15 настоящего документа).

2.12 Очистка памяти

void ***clearOrderList****(long), void* ***clearTransitList****(long), void* ***clearDealList****(long)*

Данные функции осуществляют очистку соответствующих списков заявок, транзитных приказов и сделок соответственно в разделяемой памяти, не включая указанную дату.

2.13 Системные функции

long ***getClientVersion*** – получение версии Терминала;

long ***getDllVersion*** – получение версии Модуля обмена;

long ***getTime*** – получение серверного времени;

long ***getDate*** – получение серверной даты;

long ***getLoadStatus*** – статус начальной загрузки.

std::string ***getUserNick*** – ник пользователя под которым запущен терминал.

2.14 Функции для скачивания файлов отчетов

void ***askFiles****(int)* – запрос на скачивание нескаченных файлов отчетов за указанную дату;

int ***getFilesStatus****()* – состояние закачки файлов: 0 – закачка не производится, 1 – идет закачка.

2.15 Коды состояний (статусы)

Коды состояний заявок, запросов на удаление заявок, сообщений о готовности исполнения сделки закрытия репо, которые могут возвращаться функциями: **getOrderStatus**, **getTransitConfirmStatus**, **getRepoOrderStatus**, **getDelOrderStatus**, **getRepoConfirmStatus**:

err_Waiting	*= 1*	–	*ожидается реакция системы*
err_NoError	*= 0*	–	*принято системой*
err_NoPermissions	*= 1000*	–	*не хватает полномочий*
err_OrderDisabled	*= 1400*	–	*запрещена подача объектов*
err_OrderTypeDisabled,	*= 1401*	–	*запрещена подача объектов указанного типа*
err_OrderCheckLot,	*= 1402*	–	*указано неверное количество*
err_OrderCheckMax,	*= 1403*	–	*превышены лимиты*
err_OrderPriceStep,	*= 1404*	–	*нарушен минимальный шаг изменения цены*
err_InstrumentBlocked,	*= 1414*	–	*инструмент блокирован*
err_TrdAccLimit,	*= 1501*	–	*превышены лимиты по торговому счету*
err_TradeAccountBlocked,	*= 1503*	–	*торговый счет заблокирован*
err_CashAccLimit	*= 1550*	–	*превышены лимиты по денежному счету*
err_CashAccOpenPosLimit,	*= 1551*	–	*превышены лимиты по открытым позициям*
err_OrderWrongTrdAccID	*= 1700*	–	*неправильный номер торгового счета*
err_OrderAlienFirmCashAcc,	*= 1701*	–	*указан номер денежного счета, который не принадлежит данному участнику торгов*
err_OrderNoRightsOnTrdAcc,	*= 1702*	–	*нет прав на торговый счет*
err_OrderNoRightsOnCashAcc,	*= 1703*	–	*нет прав на денежный счет*
err_OrderWrongExpireDate,	*= 1710*	–	*неправильная дата истечения*
err_OrderWrongExpireTime,	*= 1711*	–	*неправильное время истечения*
err_CmdBadSatisfyOrderID,	*= 1712*	–	*неправильный идентификатор заявки*
err_OrderWrongDirection,	*= 1713*	–	*неправильное направление*
err_OrderWrongPrice,	*= 1714*	–	*неправильная цена*
err_OrderWrongQuantity,	*= 1715*	–	*неправильное количество*
err_OrderWrongKredId,	*= 1716*	–	*неправильный идентификатор предмета репо*
err_OrderWrongFirm,	*= 1717*	–	*не найден код контрагента*
err_OrderWrongPriceOrQuantity,	*= 1718*	–	*неправильная цена или количество*
err_OrderWrongRazdel,	*= 1719*	–	*неправильный раздел счета ДЕПО*
err_ err_OrderWrongDelOrderId,	*= 1720*	–	*неправильный идентификатор заявки*

err_CmdBadInstrumentID = 2000 – указан неправильный идентификатор ("Id") инструмента

Раздел 3. СТРУКТУРЫ ДАННЫХ, ИСПОЛЬЗУЕМЫХ ФУНКЦИЯМИ TRADE.DLL

3.1 Структура заявки

*struct **Order***

{

long Id;	*– идентификатор инструмента*
char ShortName[25];	*– наименование инструмента*
long OrderId;	*– идентификатор заявки*
long OrderNum;	*– номер заявки, который может указать внешняя программа при ее подаче через функцию **PutOrder**)*
char Type;	*– тип заявки:* *"1" – лимитированная,* *"2" – репо,* *"3" – прямая* *"4" – рыночная*
char Direction;	*– направление заявки:* *"1" – покупка,* *"2" – продажа,* *"0" – не определено*
double Price;	*– цена*
long Quantity;	*– количество*
double Volume;	*– объем*
long Date;	*– дата подачи*
long TIme;	*– время подачи*
char TrdAcc[15];	*– торговый счет*
long ClntAcc;	*– клиентский счет (счет, указываемый внешними программами)*
long Status;	*– статус заявки согласно приложению 1 к настоящему документу*
char Firm[15];	*– код контрагента*
char Currency;	*– валюта:* *"0" – в тенге,* *"1" – в валюте торгов*
double ClosePrice;	*– цена закрытия*
long CloseDate ;	*– дата закрытия*
long KredId;	*– предмет репо – код инструмента (для заявок на рынке автоматического репо)*
long KredCnt;	*– количество предмета репо (для заявок на рынке автоматического репо)*
char MM;	*– маркет-мэйкеры*
long DelDate;	*– дата снятия*
long DelTime;	*– время снятия*

long PayDate; – *дата расчетов*

double RepoTax; – *ставка репо*

double SupportKoeff; – *коэффициент обеспечения*

double RiskLevel; – *уровень риска*

long ExpireDate; – *дата истечения*

long ExpireTime; – *время истечения*

char Razdel; – *раздел счета депо для случая продажи*

"0" – основной раздел,

"1" – первичное размещение

char Period; – *периодичность заявки*

"0" – в указанных ценах,

"1" – в указанных доходностях

char Settlement; – *расчеты*

"0" – Депозитарий,

"1" – Регистратор

char UserNick[15]; – *ник пользователя*

}

3.2 Структура информации об инструменте

*struct **Instrument***

{

long Id; – *идентификатор инструмента*

char MarketName[15]; – *наименование рынка*

char SectorName[15]; – *наименование сектора*

char GroupName[15]; – *наименование группы*

char ShortName[25]; – *краткое наименование инструмента*

char Name[250]; – *полное наименование инструмента*

char Status; – *статус торгов согласно приложению 2 к настоящему документу*

char SesNum; – *номер сессии*

double Open; – *курс открытия*

double Ask; – *цена предложения*

double Bid; – *цена покупки*

double Last; – *цена последней сделки*

long LastVolume; – *объем последней сделки в инструменте*

double Average; – *средняя цена*

double Max; – *максимальный курс*

double Min; – *минимальный курс*

long Volume; – *объем торгов в инструменте*

double VolTotal; – *объем торгов в контрвалюте (как правило, в тенге)*

long OrderCnt; – *количество заявок*

long DealCnt; – *количество сделок*

double UstKurs; – *официальный курс*

double KaseKurs; – *курс KASE*

long CurrId; – *ID валюты торгов*

char NIN[15]; – *НИН инструмента*

char IsCupon; – *вид бумаги:*

"0" – дисконтная,
"1" – купонная

long Nominal – *номинал*

char Method; – *база исчисления ценной бумаги (30/360, ACT/360 и т.д.)*

char Base[250]; – *база дат выплат купона*

double CuponTax; – *купонная ставка*

long CuponCnt; – *количество выплат в году*

long PastPayDate; – *дата последней выплаты купона*

long NextPayDate – *дата следующей выплаты купона*

long XDate; – *дата, с которой не начисляется накопленный интерес*

double AccInt; – *текущий накопленный интерес*

double StartKurs; – *курс доллара США на дату начала обращения*

double Koeff; – *коэффициент индексации курса доллара США*

long Days; – *количество дней до погашения*

long CloseDate; – *дата закрытия*

long OpenDate; – *дата открытия*

long Lot; – *лот*

double KorrCnt; – *множитель количества*

double KorrPrc; – *множитель цены*

long VisPrec; – *количество знаков после запятой*

char IsBlocked; – *признак того, что инструмент заблокирован:*

"1" – заблокирован,
"0" – нет

char KursMethod; – *метод пересчета курса*

long MinVol; – *минимальное количество инструмента в заявке*

double LastDealPriceDisp; – *предыдущее отклонение от цены последней сделки*

char IsDohod; – *доходность по инструменту*

"1" – есть доходность,

"0" – нет

char NameEng[250]; – *наименование инструмента на английском языке*

}

3.3 Структура информации о сделке

struct ***Deal***

{

long DealId;	*– порядковый номер сделки*
long Id;	*– идентификатор инструмента*
char ShortName[25];	*– наименование инструмента*
char BS;	*– направление сделки:* *"1" – покупка,* *"2" – продажа,* *"0" – не определено*
double Price;	*– цена сделки*
double Volume;	*– объем сделки*
long Date;	*– дата подачи*
long Tlme;	*– время подачи*
char TrdAcc[15];	*– торговый счет*
long ClntAcc;	*– клиентский счет*
long OrderId;	*– код заявки*
char NIN[15];	*– НИН инструмента*
double AccInt;	*– накопленный интерес*
long Days;	*– количество дней до погашения*
char Firm[15];	*– код контрагента*
double Yield;	*– доходность*
long OpenDate;	*– дата открытия*
long CloseDate;	*– дата закрытия*
long Quantity;	*– количество*
double ClosePrice;	*– цена закрытия*
long KredId;	*– предмет репо – код инструмента*
long KredCnt;	*– количество предмета репо*
long OrderNum;	*– номер заявки*
char Status;	*– статус сделки согласно приложению 3 к настоящему документу*
long SystemId;	*– системный Id*
char UserNick[ShortNameLen];	*– ник пользователя*
char Type;	*– тип сделки:* *"1" – простая,* *"2" – репо-открытия,* *"3" – прямая,* *"4" – репо-закрытия*
char MM;	*– маркет-мэйкеры*
long SettlDate;	*– дата расчета*
long CDDate;	*– дата расчета в ЦД*
long CDTime;	*– время расчета в ЦД*

char Razdel; – *раздел счета депо*
"0" – основной раздел,
"1" – первичное размещение
char KredNIN[15]; – *НИН инструмента*

}

3.4 Структура информации о репо-обязательстве

struct ***RepoInfo***

{

long CloseDate; – *дата закрытия*
long Id; – *идентификатор инструмента*
double ClosePrice; – *цена закрытия*
long Quantity; – *количество*
char SellTrdAcc[15]; – *торговый счет продавца*
char BuyTrdAcc[15]; – *торговый счет покупателя*
double OpenPrice; – *цена открытия*
long OpenDate; – *дата открытия*
long OpenDealId; – *код сделки открытия*
long CloseDealId; – *код сделки закрытия*
charSellConfirm; – *готовность продавца к исполнению сделки закрытия*
char BuyConfirm; – *готовность покупателя к исполнению сделки закрытия*
long KredId; – *предмет репо – код инструмента*
long KredCnt; – *количество предмета репо*
double KredOpenPrice; – *цена открытия по предмету репо*
double KredClosePrice; – *цена закрытия по предмету репо*
double CloseVol; – *объем закрытия*

}

3.5 Структура транзитного приказа

struct ***Transit***

{

long TransitId; – *идентификатор приказа*
long Id; – *идентификатор инструмента*
long OrderId; – *идентификатор заявки, поданной на основе приказа*
long Status; – *статус*
char Direction; – *направление приказа:*
"1" – покупка,
"2" – продажа,
"0" – не определено
double Price; – *цена*
long Quantity; – *количество*

double Volume; – *объем*

long QuantityRest; – *остаток*

long Date; – *дата подачи*

long Tlme; – *время подачи*

charTrdAcc[15]; – *торговый счет*

long KredId; – *предмет репо – код инструмента*

long KredCnt; – *количество предмета репо*

}

3.6 Структура котировки

struct ***ShortQuot***

{

double Price; – *цена*

long BuyVolume; – *количество покупки*

long SellVolume; – *количество продажи*

long RemVolume; – *количество, поданное в заявках участника по данной цене*

}

3.7 Структура запроса на удаление заявки

struct ***DelOrder***

{

long Id; – *идентификатор инструмента*

long OrderId; – *номер заявки*

long SystemId; – *серийник заявки*

long Status; – *статус запроса*

}

3.8 Структура запроса на подтверждение транзитного приказа

struct ***TransitConfirm***

{

long Id; – *идентификатор инструмента*

long TransitId; – *идентификатор приказа*

char Direction; – *действие:*

"1" – подтвердить,
"2" – отменить

long Status; – *статус подтверждения*

}

3.9 Структура сообщения о готовности исполнения сделки закрытия репо

struct ***RepoConfirm***

{

long Id; – *идентификатор инструмента*

long OpenDealId; – *код сделки открытия*

char ToDo; – *действие:*

"1" – подтвердить продажу,
"2" – подтвердить покупку,
"4" – отменить продажу,
"5" – отменить покупку

long Status; – *статус подтверждения*

}

3.10 Структура запроса на удовлетворение прямой заявки

struct ***SatisfyOrder***

{

long SystemId; – *идентификатор заявки*

char Direction; – *направление:*

"1"–покупка,
"2" – продажа

double Price; – *цена*

long Quantity; – *количество*

char TrdAcc[ShortNameLen]; – *торговый счет*

char Razdel; – *раздел счета депо для случая продажи*

long Status; – *статус*

}

3.11 Структура запроса на подтверждение заявки

struct ***ConfOrder***

{

long Id; – *идентификатор инструмента*

long OrderId; – *номер заявки*

long SystemId; – *идентификатор заявки*

long Status; – *статус подтверждения*

}

Приложение 1
к Принципам функционирования
Библиотеки доступа к Торговой системе
АО "Казахстанская фондовая биржа"
из внешних приложений

Статусы заявок

1 – L – osLimit

2 – T – osTrade

3 – LT – osLimit+ osTrade

8 – u – osUserRemoved

9 – Lu – osUserRemoved + osLimit

10 – Tu – osUserRemoved + osTrade

11 – LTu – osUserRemoved + osLimit + osTrade

16 – o – osOperRemoved

17 – Lo – osOperRemoved + osLimit

18 – To – osOperRemoved + osTrade

19 – Lto – osOperRemoved + osLimit+ osTrade

32 – s – osSysRemoved

33 – Ls – osSysRemoved + osLimit

35 – LTs – osSysRemoved + osLimit+ osTrade

64 – D – osDeal

65 – LD – osDeal + osLimit

66 – TD – osDeal + osTrade

67 – LTD – osDeal + osLimit + osTrade

72 – uD – osDeal + osUserRemoved

73 – LuD – osDeal + osUserRemoved + osLimit

74 – TuD – osDeal + osUserRemoved + osTrade

75 – LTuD – osDeal + osUserRemoved + osLimit + osTrade

80 – oD – osDeal + osOperRemoved

81 – LoD – osDeal + osOperRemoved + osLimit

82 – ToD – osDeal + osOperRemoved + osTrade

83 – LToD – osDeal + osOperRemoved + osLimit + osTrade

96 – sD – osDeal + osSysRemoved

97 – LsD – osDeal + osSysRemoved + osLimit

99 – LTsD – osDeal + osSysRemoved + osLimit + osTrade

136 – ub – osBrokRemoved + osUserRemoved

137 – Lub – osBrokRemoved + osUserRemoved + osLimit

138 – Tub – osBrokRemoved + osUserRemoved + osTrade

139 – LTub – osBrokRemoved + osUserRemoved + osLimit + osTrade

200 – uDb – osBrokRemoved + osDeal + osUserRemoved

201 – LuDb – osBrokRemoved + osDeal + osUserRemoved + osLimit

202 – TuDb – osBrokRemoved + osDeal + osUserRemoved + osTrade

203 – LTuDb – osBrokRemoved + osDeal + osUserRemoved + osLimit + osTrade

265 – LuW – osWaitsConf + osUserRemoved + osLimit

266 – TuW – osWaitsConf + osUserRemoved + osTrade

273 – LoW – osWaitsConf + osOperatorRemoved + osLimit

274 – ToW – osWaitsConf + osOperatorRemoved + osTrade

393 – LubW – osWaitsConf + osBrokRemoved + osUserRemoved + osLimit

394 – TubW – osWaitsConf + osBrokRemoved + osUserRemoved + osTrade

529 – LoR – osRejectedConf + osOperatorRemoved + osLimit

530 – ToR – osRejectedConf + osOperatorRemoved + osTrade,

где:

osLimit – лимитированная заявка

osTrade – рыночная заявка

osUserRemoved – заявка удалена пользователем

osOperRemoved – заявка удалена оператором

osSysRemoved – заявка удалена системой

osDeal – по заявке заключена сделка

osBrokRemoved – заявка удалена брокером

osWaitsConf – заявка ожидает подтверждения

osRejectedConf – заявка отвергнута подтверждателем

Приложение 2
к Принципам функционирования
Библиотеки доступа к Торговой системе
АО "Казахстанская фондовая биржа"
из внешних приложений

Статусы торгов

1 – C – Закрыт

3 – T – Непрерывный встречный аукцион

5 – t – Непрерывный встречный аукцион (приостановлен)

11 – x – предварительный фиксинг

12 – X – фиксинг

18 – F – Франкфуртский аукцион (открытое размещение)

19 – U – Аукцион по цене отсечения

20 – A – Аукцион по заявленной цене

21 – f – Франкфуртский аукцион (приостановлен)

22 – u – Аукцион по цене отсечения (приостановлен)

23 – a – Аукцион по заявленной цене (приостановлен)

24 – P – Предварительные торги

25 – F – Франкфуртский аукцион

26 – F – Франкфуртский аукцион

Приложение 3
к Принципам функционирования
Библиотеки доступа к Торговой системе
АО "Казахстанская фондовая биржа"
из внешних приложений

Статусы сделок

0 – dsConfirmed – "Done"

1 – dsRejectedConf – "Rejected by Investor"

2 – dsRejectedPart – "Rejected by Partner"

3 – dsRejectedSys – "Rejected by CD"

4 – dsWaitConf – "Waits for Confirm"

5 – dsWaitPart – "Waits for Partner"

6 – dsWaitSys – "Waits for CD"

7 – dsWaitsBuyer – "Waits for Buyer"

8 – dsPaidBuyer – "Paid by Buyer"

9 – dsUnpaid – "Unpaid"

10 – dsNoteDelivered – "Note Delivered"

11 – dsUndelivered – "Undelivered"

12 – dsWaitsChange – "Waits for change note"

13 – dsWaitsAgree – "Waits for agreement"

14 – dsRejectDepo – //на клиенте не встречается

15 – dsRejectedSysNoMoney – "Rejected by CD(cash problem)"

16 – dsRejectedSysNoSecur – "Rejected by CD(sec.problem)"

17 – dsWaitsAgreeDoubleNoMoney – "Waits for double agreement(cash problem)"

18 – dsWaitsAgreeDoubleNoSecur – "Waits for double agreement(sec.problem)"

19 – dsNoAgreementNoMoney – "No agreement(cash problem)"

20 – dsNoAgreementNoSecur – "No agreement(sec.problem)"

21 – dsRejectRepo – "No agreement(chg repo)"