МОСКОВСКАЯ БИРЖА

ASTS Connectivity API

Программный интерфейс
подключения внешних систем
к торгово-клиринговой системе
ASTS Московской Биржи

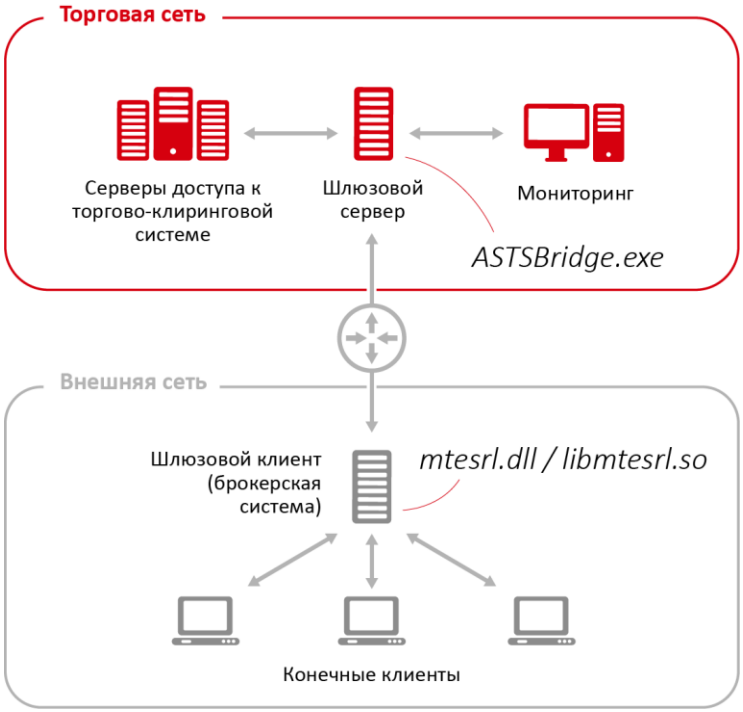
(Библиотека MTESRL v. 4.3)

Содержание

Введение

Программный интерфейс позволяет подключать к Торгово-клиринговой системе ASTS Московской Биржи брокерские системы, системы распространения биржевой информации, бэкофисные приложения, торговые автоматы и другие клиентские приложения.

Архитектура системы приведена на следующей диаграмме:



В данном документе подробно рассматривается создание клиентов программного интерфейса. Все необходимые для этого функции собраны в библиотеке *MTESRL*.

Библиотека MTESRL

Библиотека служит для создания клиентов универсального программного интерфейса, позволяющего подключать к торгово-клиринговым комплексам ASTS Московской Биржи внешние системы. Библиотека обеспечивает двунаправленную связь с торговой/клиринговой системой и содержит функции как для получения информации из торговой системы (сделки, котировки, инструменты, персональная торговая и клиринговая информация и т.п.), так и для выполнения транзакций (постановка/снятие заявок и т.п.). Библиотека позволяет работать со следующими рынками Московской Биржи: фондовый, валютный рынок и рынок драгметаллов, денежный (депозитно-кредитные операции).

Требования к программному и аппаратному обеспечению

Библиотека MTESRL разработана для следующих ОС:

- Windows Vista/7/8/10 или Windows Server 2008/2012/2016, 32- или 64-битная (`mtesrl[64].dll`);
- ОС семейства Linux на платформе x86, 32- или 64-битная (`libmtesrl.so`). Примечание: в Linux-версии библиотек используется соглашение о вызовах функций cdecl.

Существует два варианта библиотеки MTESRL, различающиеся по типу подключения к ТС:

- подключение к торговой системе через шлюз ASTS Bridge по протоколу TCP/IP;

- подключение к торговой системе напрямую. Данный вариант библиотеки может использоваться только при установке внешней системы в вычислительном центре биржи на условиях co-location.

Для нормального функционирования библиотеки MTESRL предъявляются следующие минимальные требования к аппаратному обеспечению компьютера:

- Процессор: Intel Core с частотой 1,4ГГц или выше;
- ОЗУ – 1ГБ или более;
- Жесткий диск с 10 Гбайт свободного пространства для ведения журналов работы;
- Наличие Ethernet сетевой карты.

Сценарий работы с библиотекой

Типичный сценарий работы клиента с программным интерфейсом выглядит так:

1. Регистрация на сервере.
2. Получение описания информационных объектов (типов, таблиц и транзакций).
3. Работа с информационными объектами (таблицами и транзакциями).
4. Сохранение данных, необходимых для восстановления в случае сбоя внешней системы или шлюза (snapshots или checkpoints) [опционально].
5. Завершение сеанса связи.

В подкаталоге Demo каталога установки системы находятся интерфейсные модули к библиотеке и примеры на MS Visual C, Java, C#, Delphi.

Регистрация на сервере

Подключение к ASTS Bridge

Для начала работы с интерфейсом необходимо подключиться к серверу ASTS Bridge. Для этого служит функция `MTEConnect`. Ее следует вызывать до обращения ко всем последующим функциям.

C++

```
int32 WINAPI MTEConnect(char *Params, char *ErrorMsg);
```

Pascal

```
function MTEConnect(Params, ErrorMsg: LPSTR): Integer; stdcall;
```

Аргументы:

Params

Параметры, используемые для установления соединения. Указатель на ASCIIZ-строку, содержащую список параметров, разделенных символами возврата каретки и перевода строки (0x0D, 0x0A) в следующем формате:

```
Parameter1=Value1
Parameter2=Value2
...
ParameterN=ValueN
```

Названия параметров и их допустимые значения зависят от способа соединения конкретной библиотеки с торговой системой. Библиотека MTESRL использует следующие параметры:

Подключение к ASTS Bridge

HOST	список IP-адресов и портов шлюзового сервера ASTS Bridge, разделенный запятыми, например: “194.186.240.85:20006,194.186.240.73:20006”;
PREFERREDHOST	адрес предпочтительного сервера доступа; если не указан, подключение будет выполнено к наименее загруженному серверу из списка HOST;
SERVER	идентификатор сервера, например “EQ_TEST”;
USERID	идентификатор пользователя в торговой/клиринговой системе;
PASSWORD	пароль пользователя в торговой/клиринговой системе;
INTERFACE	идентификатор интерфейса торговой системы, например, “IFCBroker_26”;
BOARDS	список режимов, с которыми будет работать пользователь, например: “TQBR,TQOB,PSEQ” (необязательный параметр, если не задан будут выбраны все доступные режимы);
COMPRESSION	Степень сжатия передаваемых данных: “0” – не сжимать данные; “1” – использовать сжатие ZLIB; “2” – сжимать большие пакеты с помощью BZIP По умолчанию используется уровень сжатия 1. Поддержка BZIP может быть удалена в будущих версиях.

Настройка шифрования и ЭЦП «Валидата»

Validata.ProfileName - имя профиля системы криптографической защиты информации «Валидата» (необязательный параметр, если не задан, то шифрование и ЭЦП будут отключены). Поддерживается также старое имя этого параметра **PROFILENAME**;

Validata.InitFlags - комбинация флагов инициализации «Валидаты»:

1 - не выполнять автоматическое обновление COC (списка отозванных сертификатов) при инициализации;
2 - показывать объекты с истекающим сроком действия при инициализации;
4 - не использовать сетевые справочники;
8 - не выгружать ключ при завершении работы.

Validata.Type, Validata.BasePath и **Validata.LdapPath** - альтернативный способ инициализации «Валидаты» (не через имя профиля). Может оказаться полезным, если «Валидата» устанавливалась под другим пользователем Windows (например, клиентская часть шлюза запускается как сервис). В этом случае в реестре текущего пользователя не будет записей об именах профилей и воспользоваться параметром ProfileName не удастся. Значения этих параметров необходимо уточнить в реестре того пользователя, под которым устанавливалась «Валидата», а именно:

Validata.Type - используемый крипто-провайдер:

Validata CSP - крипто-ядро Validata CSP (xpki1.dll);
Microsoft CSP - крипто-ядро Microsoft для нерезидентов (rpki1.dll).

Validata.BasePath - взять только путь к файлу из ключа реестра, соответствующего выбранному профилю (N = 0,1,2…):

HKEY_CURRENT_USER\Software\Validata\xpki\Profiles\<N>\store_0 (крипто-ядро Validata CSP)

HKEY_CURRENT_USER\Software\Validata\rpki\Profiles\<N>\store_0 (крипто-ядро Microsoft CSP)

Например, если в реестре записано значение «`pse://signed/C:\Users\Test\AppData\Roaming\VALIDATA\rcs\TEST_CRYPT\local.pse`», то в параметр надо передать значение «`C:\Users\Test\AppData\Roaming\VALIDATA\rcs\TEST_CRYPT\`»

Validata.LdapPath - взять строчку целиком из ключа:

HKEY_CURRENT_USER\Software\Validata\rpki\Profiles\<N>\store_2

Настройка шифрования и ЭЦП «ТУМАР»

Шлюз работает с крипто-провайдером «ТУМАР» посредством ПО «Справочник сертификатов» Валидата. В нем необходимо создать два профиля – один для ЭЦП, второй для Установка ПО и настройка профилей описана в документе «Ключи ТУМАР в ASTSBridge.pdf».

Signing.ProfileName - имя профиля «Валидата» для ЭЦП (необязательный параметр, если не задан, то ЭЦП будет отключено);

Encrypt.ProfileName - имя профиля «Валидата» для шифрования (необязательный параметр, если не задан, то шифрование будет отключено);

Signing.InitFlags,

Encrypt.InitFlags - комбинация флагов инициализации:

1 - не выполнять автоматическое обновление COC (списка отозванных сертификатов) при инициализации;

Signing.Type, Signing.BasePath,

Encrypt.Type, Encrypt.BasePath - альтернативный способ инициализации (не через имя профиля). Может оказаться полезным, если ПО «Справочник сертификатов» Валидата устанавливалось под другим пользователем Windows (например, клиентская часть шлюза запускается как сервис). В этом случае в реестре текущего пользователя не будет записей об именах профилей и воспользоваться параметром ProfileName не удастся. Значения этих параметров необходимо уточнить в реестре того пользователя, под которым устанавливалось ПО «Справочник сертификатов» Валидата, а именно:

Signing|Encryption.Type - используемый крипто-провайдер:

Microsoft CSP

Signing|Encryption.BasePath - взять только путь к файлу из ключа реестра, соответствующего выбранному профилю (N = 0,1,2…):

HKEY_CURRENT_USER\Software\Validata\rpki\Profiles\<N>\store_0

Например, если в реестре записано значение «`pse://signed/C:\Users\Test\AppData\Roaming\VALIDATA\rcs\TEST_CRYPT\local.pse`», то в параметр надо передать значение «`C:\Users\Test\AppData\Roaming\VALIDATA\rcs\TEST_CRYPT\`»

Подключение через «встроенный шлюз» на co-location

SERVER	имя сервера доступа торговой системы, например, «GATEWAY»
SERVICE	имя сервиса сервера доступа, например, «gateway»; вместо имени возможно указание номеров портов, например, 18011/18012
BROADCAST	широковещательный адрес для поиска серверов доступа торговой системы, например «10.63.1.255,10.63.3.255,10.61.1.255,10.61.3.255»
PREFBROADCAST	адрес предпочтительного сервера доступа или сети, необязательный параметр
USERID	идентификатор пользователя в торговой/клиринговой системе;
PASSWORD	пароль пользователя в торговой/клиринговой системе;
INTERFACE	идентификатор шлюзового интерфейса, с которым желает работать пользователь;
BOARDS	список режимов, с которыми желает работать пользователь, например: “ TQBR,TQOB,PSEQ” (необязательный параметр, если не задан будут выбраны все доступные режимы);
CACHEFOLDER	каталог для кеширования описания интерфейсов, загружаемых из торговой системы. Если параметр не указан, кеширование не выполняется, а интерфейс скачивается из торговой системы при каждом подключении.

LOGLEVEL	Уровень внутреннего логирования транспортного протокола: “0” – логирование отключено (значение по умолчанию); “1“ – “30“ – логирование включено, число задает подробность логирования.
COMPRESSION	сжатие трафика: “0” – не сжимать данные; “1” – сжимать данные (значение по умолчанию).
IPSRCORDER	задает предпочтительные IP адреса источника (с каких сетевых интерфейсов следует устанавливать соединение). Порядок адресов в перечислении используется для определения предпочтительного сервера. Если параметр RestrictList=0, то выполняются попытки установить соединение и со всех остальных сетевых интерфейсов, не перечисленных в IpSrcOrder, но с меньшим приоритетом. Если параметр RestrictList=1, то попытки установления соединения выполняются только с указанных сетевых интерфейсов, например, “192.168.126.1, 192.168.56.1”.
RESTRICTLIST	“0“ - поиск серверов доступа идет со всех доступных сетевых интерфейсов (значение по умолчанию); “1“ - поиск идет только с тех интерфейсов, которые указаны в IpSrcOrder.

Также для всех способов подключения поддерживаются следующие параметры:

TIMEOUT	время ожидания выполнения запроса сервером (торговой системой). Для шлюзовой mtesrl.dll – в миллисекундах, для embedded mtesrl.dll ("встроенный" шлюз) - в секундах. Значение по умолчанию - 30 секунд. Если за указанное время от сервера не будет получен ответ, начнется процедура реконнекта. Если обрыв связи будет диагностирован раньше истечения таймаута, реконнект начнется раньше;
LOGGING	Строка в формате “N,M”, где первое число N – уровень логирования вызовов API MTESRL. “0” отключить логирование операций (не создавать log-файл) “1” – логировать только ошибки “2” – логировать все вызовы функция “3” – логировать содержимое таблиц “4” – логировать содержимое таблиц и номера полей “5” - логировать сообщения транспортного протокола (только для встроенной версии библиотеки); Второе число M – уровень сбора статистики по соединению. Статистика пишется в отдельный файл вида «mtesrl-YYYMMDD-<userid>-stats.log». “0” – не собирать статистику; “1” – собирать статистику по времена исполнения запросов и размеру ответов Торговой системы; “2” - собирать статистику и распределение запросов по запросам таблицам. По умолчанию включен уровень логирования 2,2. Для полного отключения логирования необходимо задать строку “LOGGING=0,0” Срок хранения лог-файлов - 7 календарных дней. При вызове функции MTEConnect более старые файлы автоматически удаляются.
RETRIES	количество попыток восстановить связь с ASTS Bridge Server в случае коммуникационных проблем. (по умолчанию 10);
CONNECTTIME	максимальное время подключения или реконнекта в миллисекундах. По умолчанию 1 минута (60000). Может быть задано любое значение в диапазоне от 5 до 300 сек. Процедура реконнекта длится не более RETRIES попыток и не дольше CONNECTTIME миллисекунд в зависимости от того, что наступит

	быстрее. Значение данного таймаута является приблизительным и может отличаться от заданного на несколько секунд.
LOGFOLDER	каталог для создания лог-файлов работы библиотеки. По умолчанию лог-файлы создаются в одном каталоге с библиотекой.
FEEDBACK	текстовая строка в произвольном формате, описывающая клиентскую систему, подключающуюся к шлюзу. Например, «FondAnalytic v3.5.456, e-mail: admin@example.com».
LANGUAGE	язык сообщений, выдаваемых шлюзом и клиентской библиотекой MTESRL. Для изменения языка сообщений, выдаваемых торговой системой, необходимо воспользоваться транзакцией CHANGE_LANGUAGE. Значения: “Russian” или “English”.

ErrorMsg

Указатель на буфер размером не менее 256 байт, куда в случае возникновения ошибки будет помещена строка с описанием ошибки.

Возвращаемое значение:

В случае успеха функция возвращает дескриптор установленного соединения (значение большее или равное MTE_OK). Полученный дескриптор соединения используется в дальнейшем при вызове всех функций MTExxxx.

При возникновении ошибки возвращается один из кодов ошибки MTE_xxxx. При этом в аргумент ErrorMsg помещается описание проблемы.

Пример:

Установка соединения с сервером.

C++

```
int32 Idx;
char ErrorMsg[255];
...
Idx = MTEConnect(“HOST=192.168.0.10:15005\rSERVER=EQ_TEST\r
USERID=MU0000100001\rINTERFACE=IFCBroker_26”, ErrorMsg);
if( Idx < MTE_OK )
{
fprintf(stderr, “Ошибка при установке соединения: %s”, ErrorMsg);
      exit(1);
}
else
  fprintf(stdout, “Соединение установлено.”);
```

Pascal

```
Idx: Integer;
ErrorMsg: TMTEErrorMsg;
...
Idx := MTEConnect('HOST=192.168.0.10:15005'#13#10'SERVER=EQ_TEST'
#13#10'USERID=MU0000100001'#13#10'INTERFACE=IFCBroker_26',
@ErrorMsg);
if Idx < MTE_OK then
begin
  Writeln('Ошибка при установке соединения: ' + ErrorMsg);
  Halt;
end
else
  Writeln('Соединение установлено.');
```

Выбор списка режимов

Обычно список режимов, с которыми желает работать пользователь, задается в параметре BOARDS= при вызове функции `MTEConnect`. Однако список режимов может быть выбран и позднее с помощью функции `MTESelectBoards`. Разрешается использовать только один способ

выбора режимов: либо в параметре BOARDS= при вызове функции `MTEConnect`, либо функцией `MTESelectBoards`. После вызова `MTESelectBoards` необходимо закрыть все таблицы и открыть их заново, поскольку содержимое таблиц зависит от выбранных режимов.

C++

```
int32 WINAPI MTESelectBoards(int32 Idx, char * BoardsList,
                         char *result);
```

Pascal

```
function MTESelectBoards(Idx: Integer; BoardList: LPSTR;
                         ResultMsg: LPSTR): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, для которого нужно получить информацию.

BoardList

Указатель на строку, содержащую список идентификаторов режимов, разделенных запятой. Например, “TQBR,TQNE,RPMA”.

ResultMsg

Указатель на буфер размером не менее 256 байт, куда в случае успешного выполнения будет помещена строка текста с результатом обработки транзакции торговой системой.

Возвращаемое значение:

Если транзакция была обработана торговой системой, возвращается следующее:

MTE_OK – режимы выбраны;
MTE_TRANSREJECTED – запрос обработан, но был отвергнут торговым сервером (указан недопустимый режим, нет прав на выполнение и т.п.);
MTE_TSMR - фатальный сбой при выполнении запроса (потеря соединения с торговой системой и т.п.).

При этом в аргумент *ResultMsg* помещается строка текста с результатом обработки запроса торговой системой.

При возникновении ошибки возвращается один из кодов ошибки MTE_xxxx. Значение поля *ResultMsg* при этом не определено.

Получение служебной и сервисной информации

Получение информации о шлюзе

Клиент может получать служебную и сервисную информацию о серверной части шлюза и соответствующем сервере доступа торгово-клиринговой системы с помощью функции `MTEGetServInfo`.

C++

```
int32 WINAPI MTEGetServInfo(int32 Idx, char ** ServInfo, int *Len);
```

Pascal

```
function MTEGetServInfo(Idx: Integer; var ServInfo: LPSTR;
                         var Len: Integer): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, для которого нужно получить информацию.

ServInfo

Указатель на указатель на буфер в который будут помещены данные возвращаемые функцией.

Len

Указатель на переменную в которую будет помещено значение длины данных возвращаемых функцией.

Возвращаемое значение:

В случае успеха функция возвращает MTE_OK и помещает в аргумент ServInfo указатель на буфер следующего формата.

Поле	**Тип данных (IBM PC)**	**Длина поля**	**Описание**
Connected_To_Micex	INTEGER	4	Описывает состояние соединения с ТС. Возможное значение: 0 - не соединен; 1 - соединен с «боевой» ТС; 2 - соединен с тестовой ТС.
Session_Id	INTEGER	4	Внутренний номер текущей торговой сессии. Изменяется каждую торговую сессию.
MICEX_Sever_Name	CHAR	33	Логическое имя серверов доступа ТС, например, GATEWAY, FOND_GATEWAY и т.д. Позволяет определить к какой ТС подсоединен шлюз: корп. рынок, тестовый и т.д.
Version_Major	CHAR	1	Старший номер версии сервера шлюза.
Version_Minor	CHAR	1	Младший номер версии сервера шлюза.
Version_Build	CHAR	1	Номер сборки версии шлюза. Это и предыдущие поля определяют версию шлюза в порядке Major.Minor.Build.
Beta_version	CHAR	1	Является ли данный релиз бета-версией. Возможное значение: если не 0, то это бета и соответственно, это ее номер.
Debug_flag	CHAR	1	Является ли данный релиз отладочной версией. Возможное значение: значение больше 0 указывает на то, что версия является отладочной.
Test_flag	CHAR	1	Является ли данный релиз тестовой версией. Возможное значение: значение больше 0 указывает на то, что версия является тестовой.
Start_Time	INTEGER	4	Время старта новой сессии (задается в конфигурационном файле шлюза) в формате ЧЧММСС. Внимание: это целочисленное значение.
Stop_Time_Min	INTEGER	4	Время завершения работы и автоматической остановки шлюза (задается в конфигурационном файле шлюза) в формате ЧЧММСС. Внимание: это целочисленное значение.
Stop_Time_Max	INTEGER	4	Равно Stop_Time_Min.
Next_Event	INTEGER	4	Следующее ожидаемое событие в Диспетчере Расписания для соответствующего интерфейсного сервера. Возможные значения: 0 - ожидается старт новой торговой сессии, 1 - ожидается окончание текущей торговой сессии.

Event_Date	INTEGER	4	Дата ожидаемого события в формате ДДММГГГГ. Внимание: это целочисленное значение.
BoardsSelected	нультерминированная строка символов	пере-менная	Список идентификаторов выбранных режимов торгов, разделенный запятыми.
UserID	CHAR, нультерминированная строка	13	Идентификатор пользователя используемый интерфейсным сервером для данного соединения.
SystemId	CHAR	1	Тип торговой системы, возможные значения: “P” - фондовый рынок или денежный рынок; “C” - валютный рынок; “F” – срочный рынок.
ServerIp	нультерминированная строка символов	пере-менная	IP-адрес сервера доступа торговой системы, например, «195.1.3.51».

При возникновении ошибки возвращается один из кодов ошибки MTE_xxxx.

ПОЛУЧЕНИЕ ИНФОРМАЦИИ О ВЕРСИИ КЛИЕНТСКОЙ БИБЛИОТЕКИ

Клиент может получить информацию о версии клиентской библиотеки шлюза с помощью функции `MTEGetVersion`.

C++

```
char * WINAPI MTEGetVersion();
```

Pascal

```
function MTEGetVersion: LPSTR; stdcall;
```

Аргументы:

отсутствуют

Возвращаемое значение:

Указатель на ASCIIZ-строку, содержащую текстовое описание версии клиентской библиотеки, например, «MTESrl library 3.8.93».

ПОЛУЧЕНИЕ ИНФОРМАЦИИ О СОСТОЯНИИ СОЕДИНЕНИЯ

Для получения текущего состояния соединения с ASTS Bridge может быть использована функция `MTEConnectionStatus`.

C++

```
int32 WINAPI MTEConnectionStatus(int32 Idx);
```

Pascal

```
function MTEConnectionStatus(Idx: Integer): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, для которого нужно получить информацию.

Возвращаемое значение:

Один из следующих кодов MTE_xxx:

MTE_OK	Соединение установлено
MTE_INVALIDCONNECT	Указан недопустимый дескриптор соединения
MTE_SRVUNAVAIL	Недоступен сервер ASTS Bridge

MTE_TEUNAVAIL	Недоступна торговая система

СБОР СТАТИСТИКИ ПО СОЕДИНЕНИЮ

Для получения статистической информации по соединению (флаги соединения, объемы переданных данных и т.п.) предназначена функция `MTEConnectionStats`.

C++

```
int32 WINAPI MTEConnectionStats(int32 Idx, ConnectionStats * Stats);
```

Pascal++

```
function MTEConnectionStats(Idx: Integer; var Stats: TMTEConnStats):
            Integer; stdcall;
```

Idx

Дескриптор соединения, для которого нужно получить информацию.

Возвращаемое значение:

В случае успеха функция возвращает MTE_OK и заполняет структуру Stats статистической информацией о соединении. Структура Stats имеет следующий формат:

Size	int32	Входное поле, должно быть заполнено sizeof(Stats)
Properties	uint32	Флаги соединения, комбинация значений ZLIB_COMPRESSED, FLAG_ENCRYPTED, FLAG_SIGNING_ON
SentPackets	uint32	Число пакетов, отправленных на сервер ASTS Bridge
RecvPackets	uint32	Число пакетов, полученных с сервера ASTS Bridge
SentBytes	uint32	Количество байт, отправленных на сервер ASTS Bridge, с учетом сжатия
RecvBytes	uint32	Количество байт, полученных с сервера ASTS Bridge, с учетом сжатия
ServerIpAddress	uint32	IP-адрес сервера ASTS Bridge
ReconnectCount	uint32	Число повторных подключений (reconnect) к серверу ASTS Bridge
SentUncompressed	uint32	Количество байт, отправленных на сервер ASTS Bridge, без учета сжатия
RecvUncompressed	uint32	Количество байт, полученных с сервера ASTS Bridge, без учета сжатия
ServerName	char[64]	Идентификатор сервера ASTS Bridge
TsmrPacketSize	uint32	Размер пакета протокола TSMR, в байтах (только для версии co-location)
TsmrSent	uint32	Количество байт, отправленных в ТС по протоколу TSMR (только для версии co-location)
TsmrRecv	uint32	Количество байт, полученных из ТС по протоколу TSMR (только для версии co-location)

При возникновении ошибки возвращается один из кодов ошибки MTE_xxxx.

ПОЛУЧЕНИЕ ОПИСАНИЯ ИНФОРМАЦИОННЫХ ОБЪЕКТОВ

Описание информационных объектов торговой системы содержит список таблиц, транзакций, их

полей и некоторых вспомогательных объектов, доступных клиенту. Для получения описания используются функции `MTEStructure, MTEStructure2` или `MTEStructureEx`. Функции `MTEStructure2` и `MTEStructureEx` возвращают расширенный набор характеристик объектов торговой системы (см. Приложение 1. Формат буфера для функций MTEStructure, MTEStructure2 и MTEStructureEx).Функция `MTEStructureEx` полностью покрывает возможности старых функций: вызов `MTEStructure` аналогичен вызову `MTEStructureEx` с параметром *Version=0*, вызов `MTEStructure2` аналогичен вызову `MTEStructureEx` с параметром *Version=2*.

C++

```
int32 WINAPI MTEStructure(int32 Idx, MTEMsg **Msg);
int32 WINAPI MTEStructure2(int32 Idx, MTEMsg **Msg);
int32 WINAPI MTEStructureEx(int32 Idx, int32 Version, MTEMsg **Msg);
```

Pascal

```
function MTEStructure(Idx: Integer; var Msg: PMTEMsg): Integer; stdcall
function MTEStructure2(Idx: Integer; var Msg: PMTEMsg):Integer;stdcall;
function MTEStructureEx(Idx: Integer; Version: Integer; var Msg:
      PMTEMsg):Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, для которого нужно получить информацию.

Version

[Только для функции MTEStructureEx] Значение от 0 до 4 - требуемая версия структуры описания информационных объектов ТС. Старшие версии структуры содержат более подробные характеристики объектов ТС.

Начиная с версии, 3 в данном параметре могут указываться опции, позволяющие получить дополнительную информацию об интерфейсе ТС. Опции комбинируются между собой и с номером версии с помощью бинарного оператора OR.

В настоящий момент поддерживаются следующие опции:

Версия	Опция	Назначение

>= 3	STRUCTURE_LOCALIZATION = 0x0100	Поля «Заголовок» и «Описание» в интерфейсе передаются на всех языках, поддерживаемых ТС. Вместо типа String для этих полей используется конструкция:
		`КолвоЯзыков Integer`
		`Строка1 String`
		`Строка2 String`
		`...`
		`СтрокаN String`
		Каждая строка содержит в начале префикс языка «ru:», «en:» или «uk:», например, «ru:Номер заявки», «en:Order number».

Msg

Адрес переменной (имеющей тип "указатель на TMTEMsg/MTEMSG"), куда будет помещен указатель на буфер, содержащий описание информационных объектов. Память под данный буфер выделяется и освобождается библиотекой. Формат буфера для функций MTEStructure и MTEStructure2 описан в приложении 1. Структура TMTEMsg определена так:

C++

```
typedef struct {
      int32_t  DataLen; // Длина следующих далее данных
      char     Data[1]; // Псевдо-переменная
```

```
} MTEMSG;
// данные длиной DataLen следуют непосредственно за данными
// этой структуры.
```

Pascal

```
PMTEMsg = ^TMTEMsg;
TMTEMsg = record
  DataLen: Integer;  // Длина следующих далее данных
  Data: record end;  // Данные переменной длины
end;
```

Возвращаемое значение:

В случае успеха функция возвращает MTE_OK и помещает в аргумент Msg указатель на буфер с описанием.

При возникновении ошибки возвращается один из кодов ошибки MTE_xxxx. Если возвращен код ошибки MTE_TSMR, поле Data структуры Msg содержит текст сообщения об ошибке длиной DataLen символов.

Пример:

Получение описания доступных информационных объектов для сеанса с номером `Idx`.

C++

```
int32 Idx;              // Инициализирована вызовом MTEConnect
char ErrorMsg[255];
MTEMsg *Msg;
char *Data;
int32 err;
...
if ((err = MTEStructure(Idx, &Msg)) != MTE_OK ) {
      if (Err == MTE_TSMR) {
            Data = (char *)(Msg + 1);
            fprintf(stderr, "Ошибка: %s\n", Data );
      } else
            fprintf(stderr, "Ошибка: %s\n", MTEErrorMsg(Err));
} else
      fprintf("Описание информационных объектов получено.\n");
Data = (char *)(Msg + 1); // Собственно данные
```

Pascal

```
Idx: Integer;           // Инициализирована вызовом MTEConnect
Err: Integer;
Msg: PMTEMsg;
S: string;
Data: PAnsiChar;
...
Err := MTEStructure(Idx, Msg);
if Err <> MTE_OK then
  if Err = MTE_TSMR then begin
    SetString(S, @Msg.Data, Msg.DataLen);
    Writeln('Ошибка: ' + S);
  end else
    Writeln('Ошибка: ' + MTEErrorMsg(Err))
else
  Writeln('Описание информационных объектов получено.);
Data := @Msg.Data;      // собственно данные
```

РАБОТА С ИНФОРМАЦИОННЫМИ ОБЪЕКТАМИ

Работа с информационными объектами включает работу с таблицами и выполнение транзакций.

ВЫПОЛНЕНИЕ ТРАНЗАКЦИЙ

Операции, такие как постановка или снятие заявки, называемые также транзакциями, выполняются с помощью функций `MTEExecTrans, MTEExecTransIP` и `MTEExecTransEx`.

C++

```
int32 WINAPI MTEExecTrans(int32 Idx, char *TransName, char *Params,
                          char *ResultMsg);
int32 WINAPI MTEExecTransIP(int32 Idx, char *TransName, char *Params,
                          char *ResultMsg, int32 ClientIP);
```

Pascal

```
function MTEExecTrans(Idx: Integer; TransName, Params,
            ResultMsg: LPSTR): Integer; stdcall;
function MTEExecTransIP(Idx: Integer; TransName, Params,
            ResultMsg: LPSTR; ClientIP: Integer): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, на котором выполняется транзакция.

TransName

Указатель на ASCIIZ-строку с именем транзакции. Допустимые имена могут быть получены вызовом функций `MTEStructure/MTEStructure2/MTEStructureEx`.

Params

Указатель на ASCIIZ-строку, содержащую параметры транзакции. Длина строки и ее содержимое должны соответствовать описанию входных полей транзакции, полученном с помощью функций `MTEStructure/MTEStructure2/MTEStructureEx`. Все поля должны быть представлены в текстовом виде в формате торговой системы в соответствие со своим типом (см. приложение 1) следующим образом:

ftChar	Дополняется справа пробелами до длины, указанной в описании поля. Например, для поля типа ftChar(12) строка `"USER"` должна быть представлена как `"USER "`
ftInteger	Дополняется слева нулями до нужной длины. Например, значение 127 с типом ftInteger(10) преобразуется в строку `"0000000127"`.
ftFixed	Оставляется два знака после десятичной точки, убирается десятичная точка, дополняется слева нулями до нужной длины. Например, значение 927,4 с типом ftFixed(8) преобразуется в строку `"00092740"`
ftFloat	Оставляется N знаков после десятичной точки, убирается десятичная точка, дополняется слева нулями до нужной длины. Значение N зависит от формата представления цен для финансового инструмента, к которому относится данное поле. Например, значение 26,75 с типом ftFloat(9) для инструмента с N = 4 преобразуется в строку `"000267500"`
ftDate	Представляется в формате YYYYMMDD. Например значение 24 августа 1999г. преобразуется к "19990824"
ftTime	Представляется в формате HHMMSS. Например значение 16:27:39 преобразуется к "162739"
ftFloatPoint	Дополняется слева нулями до нужной длины, десятичная точка остается. Например, значение 5,617 с типом ftFloatPoint(9) преобразуется в строку `"00005.617"`
Примечание:	в полях любого типа можно передать пустое значение (NULL), для этого следует использовать строку пробелов нужной длины.

ClientIp

(для функции `MTEExecTransIP`) IP-адрес клиента, он имени которого выполняется данные транзакции. Используется в интерфейсах для технических центров и региональных бирж.

ResultMsg

Указатель на буфер размером не менее 256 байт, куда в случае успешного выполнения будет помещена строка текста с результатом обработки транзакции торговой системой.

Возвращаемое значение:

Если транзакция была обработана торговой системой, возвращается следующее:

MTE_OK - транзакция выполнена;
MTE_TRANSREJECTED - транзакция обработана, но была отвергнута торговым сервером (недопустимые параметры, нет прав на выполнение и т.п.);
MTE_TSMR - фатальный сбой при выполнении транзакции (потеря соединения с торговой системой и т.п.).

При этом в аргумент *ResultMsg* помещается строка текста с результатом обработки транзакции торговой системой.

При возникновении ошибки возвращается один из кодов ошибки MTE_xxxx. Значение поля *ResultMsg* при этом не определено.

Пример:

Допустим в описании информационных объектов, полученном с помощью MTEStructure, определена транзакция "Поставить заявку" со следующими полями:

```
ORDER                     // Имя транзакции
  BuySell: ftChar(1)      // "B" - покупка, "S" - продажа
  SecCode: ftChar(17)     // код инструмента
  Price: ftFloat(9)       // цена
  Quantity: ftInteger(10) // кол-во лотов
```

Приведенный ниже фрагмент кода ставит заявку на покупку 14 лотов инструмента "0CURRUSD000000TOD" по цене 26,15 (для этого инструмента формат представления цен содержит 4 знака после десятичной точки):

C++

```
int32 Idx; // Инициализирована вызовом MTEConnect
int32 Err;
char *ResultMsg;
...
Err = MTEExecTrans(Idx, “ORDER”,
“B0CURRUSD000000TOD0002615000000000014”, ResultMsg);
if( Err == MTE_OK )
  fprintf(stdout,”Транзакция выполнена: %s\n”, ResultMsg);
else if( Err == MTE_TSMR )
  fprintf(stdout,”Транзакция НЕ выполнена: %s\n”, ResultMsg);
else fprintf(stderr,”Ошибка: %s\n”, MTEErrorMsg(Err));
```

Pascal

```
Idx: Integer;           // Инициализирована вызовом MTEConnect
Err: Integer;
ResultMsg: TErrorMsg;
...
Err := MTEExecTrans(Idx, 'ORDER',
'B0CURRUSD000000TOD0002615000000000014', @ResultMsg);
case Err of
  MTE_OK: Writeln('Транзакция выполнена: ' + ResultMsg);
  MTE_TSMR, MTE_TRANSREJECTED: Writeln('Транзакция НЕ выполнена:
' + ResultMsg);
  else Writeln('Ошибка: ' + MTEErrorMsg(Err));
end;
```

Примечание: Для каждого соединения, транзакции и информационные запросы передаются в торговую систему последовательно: отправка новой транзакции или запроса возможна только после получения ответа на предыдущий запрос. Для исключения задержек, рекомендуется:

- Использовать раздельные соединения для передачи транзакций и для запросов таблиц торговой системы.

- При больших пиковых частотах транзакций, использовать несколько соединений с балансировкой их загрузки.

Новые транзакции Торговой системы могут возвращать несколько ответов, либо возвращать строку-ответ длиной более 255 символов. Для таких транзакций рекомендуется использовать функцию `MTEExecTransEx`, которая возвращает массив ответов торговой системы и текстовые сообщения неограниченной длины:

C++

```
int32 WINAPI MTEExecTransEx(int32 Idx, char *TransName, char *Params,
                  int32 ClientIp, MTEExecTransResult *Reply);
```

Pascal

```
function MTEExecTransEx(Idx: Integer; TransName, Params: LPSTR;
  ClientIp: Integer; var Reply: TMTEExecTransResult): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, на котором выполняется транзакция.

TransName

Указатель на ASCIIZ-строку с именем транзакции. Допустимые имена могут быть получены вызовом функции `MTEStructure/MTEStructure2/MTEStructureEx`.

Params

Указатель на ASCIIZ-строку, содержащую параметры транзакции. Длина строки и ее содержимое должны соответствовать описанию входных полей транзакции, полученном с помощью функций `MTEStructure/MTEStructure2/MTEStructureEx`. Все поля должны быть представлены в текстовом виде в формате торговой системы (см. `MTEExecTrans`).

ClientIp

IP-адрес клиента, он имени которого выполняется данные транзакции. Используется в интерфейсах для технических центров и региональных бирж.

Reply

Указатель на структуру, в которую будет помещен результат исполнения транзакции и ответ торговой системы (ТС). Структура `TMTEExecTransResult / MTEExecTransResult` определена так:

C++

```
typedef struct TransResult {
      // количество записей в поле replies
      uint32_t replyCount;
      // указатель на массив записей MTETransReply
      MteTransReply* replies;
} MteTransResult;

// один ответ торговой системы
typedef struct TransReply {
      int32_t errCode;  // код возврата (см. Возвращаемое
значение)
      int32_t msgCode;  // номер сообщения ТС (который
указывается в тексте сообщения в круглых скобках)
      char*   msgText;  // текстовое сообщение ТС
      int32_t paramCount; // количество параметров в ответе ТС
      // указатель на массив записей MTETransParam
      MteTransParam* params;
} MteTransReply;
```

```
// дополнительный параметр в ответе на транзакцию
typedef struct TransParam {
        char* name;     // идентификатор параметра
        char* value;    // значение параметра
} MteTransParam;
```

Pascal

```
TMTEExecTransResult = record
  // количество записей в поле Replies
  ReplyCount: Longword;
  // указатель на массив записей TMTETransReply
  Replies: PMTETransReplies;
end;

// один ответ торговой системы
TMTETransReply = record
  ErrCode: TMTEResult;  // код возврата (см. Возвращаемое
значение)
  MsgCode: Integer;     // номер сообщения ТС (который
указывается в тексте сообщения в круглых скобках)
  MsgText: PAnsiChar;   // текстовое сообщение ТС
  ParamCount: Integer;  // количество параметров в ответе ТС
  // указатель на массив записей TMTETransParam
  Params: PMTETransParams;
end;

// дополнительный параметр в ответе на транзакцию
TMTETransParam = record
  Name: PAnsiChar;      // идентификатор параметра
  Value: PAnsiChar;     // значение параметра
end;
```

Большинство транзакций торговой системы возвращает ровно один ответ, соответственно поле `ReplyCount` равно 1 и `Replies` содержит 1 запись. Пример транзакции, возвращающей более одного ответа – транзакция ORDER_AMEND (изменение заявки).

Возвращаемое значение:

Если транзакция была обработана торговой системой, возвращается следующее:

- MTE_OK - транзакция выполнена;
- MTE_TRANSREJECTED - транзакция обработана, но была отвергнута торговым сервером (недопустимые параметры, нет прав на выполнение и т.п.);
- MTE_TSMR - фатальный сбой при выполнении транзакции (потеря соединения с торговой системой и т.п.).

Дополнительные параметры, которые могут присутствовать в ответе. Количество параметров определяется полем ParamCount.

- ST – время начала обработки транзакции ядром торговой системы в формате ST=HHMMSSmicroseconds;
- ON – номер зарегистрированной заявки;
- IN – публичный номер заявки в потоке рыночных данных по протоколу FAST; указывается только для тех заявок, которые подлежат публикации в FAST-потоке.

РАБОТА С ТАБЛИЦАМИ

Работа с таблицами включает в себя следующие шаги:

1. Открытие таблицы
2. Периодический запрос изменений
3. Закрытие таблицы

ОТКРЫТИЕ ТАБЛИЦЫ

Работа с таблицей торговой системы начинается с вызова функции `MTEOpenTable`. Функция открывает таблицу и возвращает часть или все текущее содержимое таблицы.

C++

```
int32 WINAPI MTEOpenTable(int32 Idx, char *TableName, char *Params,
                          int32 Complete, MTEMSG **Msg);
```

Pascal

```
function MTEOpenTable(Idx: Integer; TableName, Params: LPSTR;
                  Complete: BOOL; var Msg: PMTEMsg): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, полученный с помощью вызова `MTEConnect`.

TableName

Указатель на ASCIIZ-строку с именем таблицы. Допустимые имена могут быть получены вызовом функции `MTEStructure/MTEStructure2/MTEStructureEx`.

Params

Указатель на ASCIIZ-строку, содержащую параметры таблицы. Длина строки и ее содержимое должны соответствовать описанию входных полей таблицы, полученном с помощью функций `MTEStructure/MTEStructure2/MTEStructureEx`. Все поля должны быть представлены в текстовом виде в формате торговой системы (см. `MTEExecTrans`).

Complete

Флаг, позволяющий запросить все содержимое таблицы или только часть. Используется следующим образом:

TRUE Функция возвращает всю информацию, содержащуюся в данный момент в таблице. Выполняет столько обращений к торговой системе, сколько нужно для получения всех данных. При большом объеме таблицы (например "Сделки") может выполняться долго и даже привести к потере соединения из-за таймаута. Если все содержимое сразу не требуется и чтобы уменьшить время выполнения, следует использовать значение FALSE.

FALSE Функция возвращается только часть данных или вообще ничего, в зависимости от типа таблицы. Выполняет не более одного обращения к торговой системе. Остальные данные рассматриваются как обновления и должны дочитываться в цикле запроса изменений с помощью вызовов `MTEAddTable/MTERefresh`.

Msg

Адрес переменной (имеющей тип "указатель на `TMTEMsg/MTEMSG`"), куда в случае успеха будет помещен указатель на буфер, содержащий информацию из открытой таблицы. Формат буфера описан в приложении 2.

Возвращаемое значение:

В случае успеха функция возвращает дескриптор открытой таблицы (значение большее или равное MTE_OK). Полученный дескриптор используется в дальнейшем при вызове функции MTEAddTable.

При возникновении ошибки возвращается один из кодов ошибки MTE_xxxx. Если возвращен код ошибки MTE_TSMR, поле Data структуры Msg содержит текст сообщения об ошибке длиной DataLen символов.

ЗАПРОС ИЗМЕНЕНИЙ

Запрос изменений выполняется в пакетном режиме, т.е. одновременно запрашиваются изменения по нескольким открытым таблицам. Для этого запрос сначала формируется путем нескольких вызовов MTEAddTable, а затем выполняется с помощью MTERefresh. Вызов других функций библиотеки (кроме MTEErrorMsg) между двумя этими функциями запрещен.

Функция MTEAddTable добавляет в очередь (пакет запросов) запрос на получение изменений в таблице, с момента предыдущего запроса изменений.

C++

```
int32 WINAPI MTEAddTable(int32 Idx, int32 HTable, int32 Ref);
```

Pascal

```
function MTEAddTable(Idx, HTable, Ref: Integer): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, полученный с помощью вызова MTEConnect.

HTable

Дескриптор таблицы, полученный с помощью вызова MTEOpenTable.

Ref

Дополнительный параметр, используемый клиентом по своему усмотрению. Применяется обычно для идентификации информации, предназначенной данной таблице, в буфере, возвращаемом функцией MTERefresh.

Возвращаемое значение:

Один из кодов ошибки MTE_xxxx.

Функция MTERefresh отправляет на сервер пакет запросов на получение изменений, сформированный вызовами MTEAddTable, и возвращает эти изменения.

C++

```
int32 WINAPI MTERefresh(int32 Idx, MTEMSG **Msg);
```

Pascal

```
function MTERefresh(Idx: Integer; var Msg: PMTEMsg): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, полученный с помощью вызова MTEConnect.

Msg

Адрес переменной (имеющей тип "указатель на TMTEMsg/MTEMSG "), куда в случае успеха будет помещен указатель на буфер, содержащий полученные обновления. Формат буфера описан в приложении 3.

Возвращаемое значение:

В случае успеха функция возвращает MTE_OK и помещает в аргумент Msg указатель на полученные данные.

При возникновении ошибки возвращается один из кодов ошибки MTE_xxxx. Если возвращен код ошибки MTE_TSMR, поле Data структуры Msg содержит текст сообщения об ошибке длиной DataLen символов.

ЗАКРЫТИЕ ТАБЛИЦЫ

По окончании работы с таблицей ее необходимо закрыть, используя функцию MTECloseTable. После вызова этой функции дескриптор таблицы не может более использоваться.

C++

```
int32 WINAPI MTECloseTable(int32 Idx, int32 HTable);
```

Pascal

```
function MTECloseTable(Idx, HTable: Integer): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, полученный с помощью вызова MTEConnect.

HTable

Дескриптор закрываемой таблицы, полученный с помощью вызова MTEOpenTable.

Возвращаемое значение:

Один из кодов ошибки MTE_xxxx.

ПРИМЕР РАБОТЫ С ТАБЛИЦАМИ

Допустим в описании информационных объектов, полученном с помощью MTEStructure, определены таблицы "Ценные бумаги" и "Сделки" со следующими входными полями:

```
SECURITIES                // Имя таблицы (Ценные бумаги)
  Market: ftChar(4)       // Код рынка
  Board: ftChar(4)        // Код режима торгов

TRADES                    // Таблица "Сделки" без параметров
```

В следующем фрагменте кода показано как следует работать с таблицами торговой системы. Таблицы открываются, периодически запрашивается изменение их содержимого, и затем таблицы закрываются.

C++

```
int32 Idx; // Инициализирована вызовом MTEConnect
MTEMsg *Msg;
char *Data;
int32 HSecurs, Htrades;
...

HSecurs = MTEOpenTable(Idx, "SECURITIES", "CURR    ", 1 /*True*/,
&Msg);
Data = (char *)(Msg + 1);
...
// Обработка полученных данных
...
HTrades = MTEOpenTable(Idx, "TRADES", "", 0/*False*/, Msg);
Data = (char *)(Msg + 1);
...
```

```
// Обработка полученных данных
...

do
{
  MTEAddTable(Idx, HSecurs, 0);
  MTEAddTable(Idx, HTrades, 1);
  MTERefresh(Idx, &Msg);
  Data = (char *)(Msg + 1);
  ...
  // Обработка обновлений
  ...
}while( !Terminated );

MTECloseTable(Idx, HSecurs);
MTECloseTable(Idx, HTrades);
```

Pascal

```
Idx: Integer;           // Инициализирована вызовом MTEConnect
Msg: PMTEMsg;
HSecurs, HTrades: Integer;
Data: PAnsiChar;
...

HSecurs := MTEOpenTable(Idx, 'SECURITIES', 'CURR    ', True,
Msg);
...
// Обработка полученных данных
...
HTrades := MTEOpenTable(Idx, 'TRADES', '', False, Msg);
...
// Обработка полученных данных
...

repeat
  MTEAddTable(Idx, HSecurs, 0);
  MTEAddTable(Idx, HTrades, 1);
  MTERefresh(Idx, Msg);
  Data := @Msg.Data;
  ...
  // Обработка обновлений
  ...
until Terminated;

MTECloseTable(Idx, HSecurs);
MTECloseTable(Idx, HTrades);
```

ЗАМЕЧАНИЯ ПО РАБОТЕ С ТАБЛИЦАМИ

Замечание 1. Во избежание разрыва соединения по таймауту со стороны сервера рекомендуется: во-первых, в настройке шлюза не устанавливать слишком маленькое (меньше 60 секунд) значение DisconnectIfIdleFor; во-вторых, регулярно (примерно с интервалом в 30 секунд) поддерживать соединение в активном состоянии – например, запрашивая обновление таблицы TESYSTIME.

Замечание 2. Большинство таблиц торговой системы можно открывать (как в одном, так и в нескольких экземплярах) и закрывать произвольное число раз в течение сеанса связи с сервером. Однако, некоторые таблицы могут быть открыты только один раз в течение сеанса. К таким таблицам относится, например, таблицы ORDERS ("Заявки"), TRADES ("Сделки"), NEGDEALS ("Адресные (внесистемные) заявки"), ALL_TRADES ("Все сделки"), POSITIONS ("Позиции по деньгам"), HOLDINGS ("Позиции по инструментам"), RM_INDICATIVE ("Параметры процентных

рисков"). Если такую таблицу закрыть, а затем открыть вновь, то содержимое таблицы, полученное в первый раз, вновь получено не будет, а будут приходить только изменения.

В связи с вышесказанным, такие таблицы рекомендуется открывать только один раз в течение сеанса связи и закрывать их только при завершении сеанса.

Замечание 3. Для таблиц с установленным флагом "tfClearOnUpdate - Очищать при обновлении" (кроме таблицы EXT_ORDERBOOK) определен следующий порядок обработки обновлений: когда таблица должна быть полностью очищена КолвоСтрок устанавливается равным 1, то есть, возвращается одна строка со значением ДлинаДанных = 0 (см. приложение 2).

Для таблицы EXT_ORDERBOOK существуют два режима запроса информации по котировкам:

1. Для получения информации по одному инструменту, в запросе задаются непустые значения для полей "Режим" и "Инструмент";
2. Для получения информации по котировкам всех доступных инструментов в одном запросе, поля "Режим" и "Инструмент" заполняются символами пробела.

Соответственно, для первого способа в случае, когда таблица котировок должна быть очищена в ответ на запрос, приходит таблица с одной строкой, содержащей следующие значения: КолвоПолей =2 и ДлинаДанных = (длина поля "Режим" + длина поля "Инструмент"). В этой строке содержатся только поля "Режим" и "Инструмент". Для второго случая в ответе на запрос могут содержаться несколько таких строк (в которых присутствуют только значения полей "Режим" и "Инструмент"), что означает очистку значений котировок для данных инструментов.

Обратите внимание на еще два момента: при первом запросе таблицы для всех доступных инструментов, т.е. при открытии таблицы, могут прийти строки изначально характеризующие пустую таблицу котировок, как это описано в предыдущем абзаце. Это связано с логикой выдачи информации Торговой Системой: по данным бумагам в течение торговой сессии выставлялись заявки, но все они были сняты на момент запроса. При последующих запросах выдается информация только по инструментам, по которым произошли изменения в котировках.

Второй момент: тестовое приложение TEClient.exe отображает в окне котировок для всех инструментов (т.е. при открытии таблицы с пустыми полями «Режим» и «Инструмент») только данные последнего запроса на изменения, т.е. только те котировки, в которых произошли изменения. Информация по инструментам, у которых не было изменений в котировках, будет отсутствовать.

Замечание 4. Максимальная частота информационных запросов в единицу времени регламентируется утверждённым биржей документом «Требования, предъявляемые ПАО Московская Биржа к сопряжению внешних программно-технических средств (ВПТС) с Программно-техническим комплексом Технического центра (ПТК ТЦ)». Для исключения задержек в получении информации при пиках активности рынка допускается адаптивный интервал между запросами: если объем полученного пакета данных превышает 30 кбайт, немедленно отправить новый запрос на обновление. Если объем данных в пакете не превышает этой величины, следующий запрос может быть отправлен через обычный интервал времени.

Замечание 5. При обработке буфера со строками таблицы построчная информация должна быть объединена на основе значений в ключевых полях. В некоторых случаях, например при первичном открытии таблицы SECURITIES, в одном буфере может встречаться несколько записей по одной и той же строке, которые необходимо объединить. Помимо этого, как описано ниже в приложении, буфер может содержать как полную строку (включая статические значения), так и только информацию по изменениям. Код программы рекомендуется писать исходя из предположения, что получение частичного набора полей возможно для любой таблицы.

Оптимизация использования памяти

Все функции библиотеки MTESRL, возвращающие указатель на буфер с информацией (указатель на структуру `PMTEMsg/MTEMSG`, например, `MTEStructure`, `MTERefresh` и другие) используют в качестве приемного буфера одну и ту же область памяти (в рамках одного соединения, для разных соединений используются разные области памяти). Назовем такие функции информационными.

Если при очередном вызове информационной функции размер получаемых данных превышает размер выделенного для приема буфера, происходит выделение (Reallocation) большего блока

памяти. Таким образом максимальный размер выделенной памяти равен размеру максимального полученного блока информации. Вся выделенная память освобождается при завершении соединения с помощью функции `MTEDisconnect`.

Существует возможность освободить память, используемую в качестве приемного буфера, в произвольный момент времени, не завершая соединения. Для этого предназначена функция `MTEFreeBuffer`. Эту функцию следует вызывать только после обработки всех принятых данных. Следует помнить, что это приведет к необходимости выделения памяти при следующем вызове одной из информационных функций. Частый вызов функции `MTEFreeBuffer` может отрицательно повлиять на производительность.

C++

```
int32 WINAPI MTEFreeBuffer(int32 Idx);
```

Pascal

```
function MTEFreeBuffer(Idx: Integer): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, полученный с помощью вызова `MTEConnect`, для которого необходимо освободить память.

Возвращаемое значение:

Один из кодов ошибки `MTE_xxxx`.

Функция является устаревшей и поддерживается для совместимости со старыми системами пользователей.

Восстановление после сбоя на ASTS Bridge

В процессе эксплуатации системы иногда может потребоваться перезапуск внешней системы или шлюза в связи с возникновением критической ошибки. При этом необходимо обеспечить восстановление работоспособности системы в кратчайшее сроки. В таких ситуациях рекомендуется использовать следующую технологию: внешняя система с определенной периодичностью производит резервное сохранение данных загруженных таблиц и состояний внутренних структур шлюза в файлах; в случае сбоя системы используются данные из сохраненных файлов для восстановления системы к состоянию, которое она имела на момент резервного сохранения данных.

Библиотека MTESRL позволяет начать получение данных от ASTS Bridge не с “нуля”, а с некоторого момента. Для этого предварительно должен быть сохранен “снимок” состояния открытых таблиц. Впоследствии, например, в случае потери соединения с сервером шлюза, можно восстановить состояние открытых таблиц и продолжить получение информации.

Сохранение состояния внутренних структур шлюза

Резервное сохранение состояния внутренних структур шлюза производится после запроса изменений в таблицах и их обработки. Данную операцию можно производить после каждого запроса изменений или после выполнения некоторого количества запросов. Как правило, на ряду с сохранением состояний внутренних структур шлюза, сохраняются также текущее состояние всех таблиц внешней системы. При этом обеспечивается сохранение полного текущего состояния системы состоящей из внешней системы и шлюза. Ниже приведен подробный сценарий работы в таких случаях.

Для получения текущего состояния открытых на сервере таблиц используется функция `MTEGetSnapshot`.

C++

```
int32 WINAPI MTEGetSnapshot(int32 Idx, char ** Snapshot, int *Len);
```

Pascal

```
function MTEGetSnapshot(Idx: Integer; var Snapshot: LPSTR;
                        var Len: Integer): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, для которого необходимо получить «снимок» открытых таблиц.

Snapshot

Адрес переменной, куда в случае успеха будет помещен указатель на «снимок».

Len

Адрес переменной, куда в случае успеха будет помещена длина «снимка» (буфера, указатель на который находится в *Snapshot*).

Возвращаемое значение:

В случае успеха функция возвращает `MTE_OK`.

При возникновении ошибки возвращается один из кодов ошибки `MTE_xxxx`. Если возвращен код ошибки `MTE_TSMR`, аргумент *Snapshot* указывает на текст сообщения об ошибке, а аргумент *Len* содержит длину этого сообщения.

«Снимок» открытых на сервере таблиц может рассматриваться просто как буфер некоторых двоичных данных. Его содержимое не несет для клиента никакой смысловой нагрузки.

В следующем фрагменте кода предполагается, что внешняя система выполнила подключение к ASTS Bridge, получила структуру данных, открыла таблицы и перешла к циклу получения изменений по таблицам:

C++

```
int32 Idx;        // Инициализирована вызовом MTEConnect
MTEMsg *Msg;
char *DataPtr;
int32 *TablesIdx; // массив индексов полученных при MTEOpenTable
вызовах
int32 i,NumTables;// количество обновляемых таблиц
char *SnapshotBuf;// указатель на буфер для резервного сохранения
int32 SnapshotLen;// длина буфера для резервного сохранения
...
do
{
  for( i = 0; i < NumTables; i++ )
      MTEAddTable(Idx, TablesIdx[i], i);
  MTERefresh(Idx, &Msg);
  DataPtr = (char *)(Msg + 1);
  ...
  // Обработка обновлений
  ...
  // Получения буфера состояний внутренних структур шлюза
  MTEGetSnapshot(Idx, &SnapshotBuf, &SnapshotLen);
  // сохранение буфера в файле
  ...
  // сохранение состояния ВС
  ...
}while( !Terminated );
```

Pascal

```
Idx: Integer;           // Инициализирована вызовом MTEConnect
Msg: PMTEMsg;
DataPtr: PChar;
```

```
TablesIdx: array of Integer; // массив индексов полученных при
MTEOpenTable вызовах
i, NumTables: Integer;  // количество обновляемых таблиц
SnapshotBuf: PChar;     // указатель на буфер для резервного
сохранения
SnapshotLen: Integer;   // длина буфера для резервного сохранения
...
repeat
  for i := 0 to NumTables – 1 do
      MTEAddTable(Idx, TablesIdx[i], i);
  MTERefresh(Idx, Msg);
  DataPtr = @Msg.Data;
  ...
  // Обработка обновлений
  ...
  // Получения буфера состояний внутренних структур шлюза
  MTEGetSnapshot(Idx, SnapshotBuf, SnapshotLen);
  // сохранение буфера в файле
  ...
  // сохранение состояния ВС
  ...
until Terminated;
```

Восстановление состояния внутренних структур шлюза

Для получения списка открытых таблиц, содержащихся в заданном снепшоте, можно воспользоваться функцией `MTEGetTablesFromSnapshot`. Данная функция может быть вызвана до или после вызова функции `MTESetSnapshot`.

C++

```
int32 WINAPI MTEGetTablesFromSnapshot(int32 Idx, char * Snapshot,
      int Len, MTESnapTable **SnapTables);
```

Pascal

```
function MTEGetTablesFromSnapshot(Idx: Integer; Snapshot: LPSTR;
      Len: Integer, var SnapTables: PMTESnapTables): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, полученный с помощью вызова `MTEConnect`.

Snapshot

Указатель на буфер, в котором помещены данные полученные с помощью вызова `MTEGetSnapshot`.

Len

Длина данных в передаваемом буфере.

SnapTables

Указатель на указатель на структуру `MTESnapTable`, куда в случае успеха будет помещен указатель на сформированный буфер открытых таблиц. Память под данный буфер выделяется библиотекой. При повторных вызовах функции используется тот же самый буфер, поэтому результат должен быть сохранен внешней системой. Ниже описан формат буфера открытых таблиц:

C++

```
typedef struct SnapTable {
      int32 Htable;     // Дескриптор открытой таблицы
      char* TableName   // Указатель на ASCIIZ-строку с именем
таблицы.
      char* Params;     // Указатель на ASCIIZ-строку с
параметрами с которыми открывалась таблица.
```

```
} MteSnapTable;
```

Pascal

```
TMTESnapTable = record
  HTable: Integer;          // Handle of table
  TableName: PAnsiChar;     // char, Zero-byte terminated, Table
Name
  Params: PAnsiChar;        // char, Zero-byte terminated,
Parameters provided on open table
end;

PMTESnapTables = ^TMTESnapTables;
TMTESnapTables = array [0..999999] of TMTESnapTable;
```

Возвращаемое значение:

В случае отрицательного значения код возврата трактуется как один из кодов ошибки `MTE_xxxx`.

В случае успешного выполнения запроса функция возвращает неотрицательное значение, равное количеству открытых таблиц, и возвращает указатель на сформированный массив структур открытых таблиц `MTESnapTable` через параметр `SnapTables`.

Восстановление состояния внутренних структур шлюза производится при перезапуске системы или шлюза после сбоев для приведения системы к состоянию на момент сохранения резервной копии. Данная операция должна производится только в рамках текущей торговой сессии и должна правильно восстанавливать состояние внешней системы на тот момент, когда происходило резервное сохранение состояния системы (см. `MTEGetSnapshot`). В результате данной операции все открытые таблицы на шлюзе и дескрипторы этих таблиц восстанавливаются. То есть, непосредственно после восстановления системы, можно использовать дескрипторы таблиц, существовавшие до возникновения сбоя. Для восстановления состояния шлюза служит функция `MTESetSnapshot`.

C++

```
int32 WINAPI MTESetSnapshot(int32 Idx, char * Snapshot, int Len,
                            char *ErrorMsg);
```

Pascal

```
function MTESetSnapshot(Idx: Integer; Snapshot: LPSTR; Len: Integer;
                        ErrorMsg: LPSTR): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, для которого восстанавливается состояние.

Snapshot

Указатель на буфер, содержащий предварительно снятый «снимок».

Len

Длина буфера, на который указывает Snapshot.

ErrorMsg

Указатель на буфер размером не менее 256 байт, куда будет помещена строка текста с результатом восстановления состояния.

Возвращаемое значение:

Если функция была обработана торговой системой, возвращается следующее:

MTE_OK – восстановление выполнено;
MTE_TSMR - торговая система не смогла восстановить состояние.

При этом в аргумент *ErrorMsg* помещается строка текста с результатом, возвращенным торговой системой.

При возникновении ошибки возвращается один из кодов ошибки MTE_xxxx. Значение поля *ErrorMsg* при этом не определено.

В следующем фрагменте кода предполагается, что внешняя система выполнила резервное сохранение своего состояния и состояния шлюза в один из моментов до произошедшего сбоя. Производится полный перезапуск системы, включая сервер шлюза (Аналогично можно действовать при перезапуске только внешней системы или только сервера шлюза). Система выполнила подключение к ASTS Bridge и получила описание структуры данных с сервера:

C++

```
int32 Idx;              // Инициализирована вызовом MTEConnect
MTEMsg *Msg;
char *DataPtr;
int32 *TablesIdx;       // массив индексов для открытых таблиц
int32 i, NumTables;     // количество обновляемых таблиц
char *SnapshotBuf;      // указатель на буфер для данных которые
будут использованы при восстановлении состояния сервера шлюза
int32 SnapshotLen;      // длина буфера
...
// Восстановление состояния внешней системы из сохраненных данных
// При этом восстанавливаются значения NumTables и массива
индексов открытых таблиц
...
// Загрузка сохраненного, после последнего вызова
// MTEGetSnapshot, буфера из файла
// (инициализация и загрузка буфера SnapshotBuf)
...
//Восстановление состояния внутренних структур шлюза
MTESetSnapshot(Idx, SnapshotBuf, SnapshotLen);
//переход к циклу нормальной работы внешней системы
do
{
  for( i = 0; i < NumTables; i++ )
      MTEAddTable(Idx, TablesIdx[i], i);
  MTERefresh(Idx, &Msg);
  DataPtr = (char *)(Msg + 1);
  ...
  // Обработка обновлений
  ...
}while( !Terminated );
```

Pascal

```
Idx: Integer;           // Инициализирована вызовом MTEConnect
Msg: PMTEMsg;
DataPtr: PChar;
TablesIdx: array of Integer; // массив индексов для открытых
таблиц
i, NumTables: Integer;  // количество обновляемых таблиц
SnapshotBuf: PChar;     // указатель на буфер для данных которые
будут использованы при восстановлении состояния сервера шлюза
SnapshotLen: Int32;     // длина буфера
...
// Восстановление состояния внешней системы из сохраненных данных
// При этом восстанавливаются значения NumTables и массива
индексов открытых таблиц
...
// Загрузка сохраненного, после последнего вызова
// MTEGetSnapshot, буфера из файла
// (инициализация и загрузка буфера SnapshotBuf)
...
//Восстановление состояния внутренних структур шлюза
MTESetSnapshot(Idx, SnapshotBuf, SnapshotLen);
```

```
//переход к циклу нормальной работы внешней системы
repeat
  for i := 0 to NumTables – 1 do
      MTEAddTable(Idx, TablesIdx[i], i);
  MTERefresh(Idx, Msg);
  DataPtr = @Msg.Data;
  ...
  // Обработка обновлений
  ...
until Terminated;
```

Алгоритм восстановления после сбоя

Предположим, что мы:

1. Установили соединение с сервером ASTS Bridge с помощью `MTEConnect`;
2. Открыли несколько таблиц с помощью `MTEOpenTable` и сохранили их дескрипторы в переменных hTable1, hTable2, ..., hTableN;
3. Выполняли транзакции и запрашивали обновления информационных таблиц, периодически сохраняя «снимок» состояния с помощью функции `MTEGetSnapshot`;
4. Допустим, в какой-то момент соединение с сервером ASTS Bridge было нарушено. Процедура восстановления будет выглядеть так:
5. Заново устанавливаем соединение с ASTS Bridge с помощью `MTEConnect`;
6. Вызываем `MTESetSnapshot` с последним сохраненным «снимком»;
7. Теперь можем пользоваться старыми дескрипторами таблиц hTable1, hTable2, ..., hTableN, открытыми в предыдущем сеансе. Вызывать `MTEOpenTable` не нужно. Последующие вызовы функции `MTERefresh` будут возвращать обновления таблиц, накопленные после сохранения Snapshot.

Если данные, полученные до обрыва связи, были сохранены, использование механизма Get/Set Snapshot позволяет существенно уменьшить время получения всех обновлений таблиц после восстановления соединения.

Выборочное открытие таблиц из снепшота

Существует также альтернативный вариант восстановления после сбоя. Вместо того, чтобы сохранять и восстанавливать полное состояние всех таблиц, можно восстановить из снепшота только некоторые, особо объемные таблицы (например, «Заявки» и «Сделки»), а остальные таблицы открыть обычным способом через `MTEOpenTable`. При этом отпадает необходимость хранения вместе со снепшотом списка открытых таблиц и их дескрипторов. Достаточно сохранить только сам снепшот, а затем открывать таблицы, содержащиеся в нем, с помощью функции `MTEOpenTableAtSnapshot`. Данные по таблицам, отрытым таким способом, начнут приходить не с «нуля», а начиная с момента, когда был сделан соответствующий снепшот. Вызывать **`MTESetSnapshot`** в данном сценарии не требуется.

C++

```
int32 WINAPI MTEOpenTableAtSnapshot (int32 Idx, char* TableName,
      char* Params, char* Snapshot, int SnapshotLen, MTEMsg **Msg);
```

Pascal

```
function MTEOpenTableAtSnapshot(Idx: Integer;
      TableName, Params, Snapshot: PAnsiChar;
      SnapshotLen: Integer; var Msg: PMTEMsg): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, полученный с помощью вызова `MTEConnect`.

TableName

Указатель на ASCIIZ-строку с именем таблицы. Допустимые имена могут быть получены вызовом функции `MTEStructure/MTEStructure2/MTEStructureEx`.

Params

Указатель на ASCIIZ-строку, содержащую параметры таблицы. Длина строки и ее содержимое должны соответствовать описанию входных полей таблицы, полученном с помощью функций `MTEStructure/MTEStructure2/MTEStructureEx`. Все поля должны быть представлены в текстовом виде в формате торговой системы (см. `MTEExecTrans`).

Snapshot

Указатель на буфер, содержащий снепшот. Запрашиваемая таблица с указанными параметрами должна содержаться в данном снепшоте, иначе функция вернет ошибку MTE_TSMR. Если в данном параметре передан нулевой указатель, то функция ведет себя аналогично вызову `MTEOpenTable` с параметром `Complete=FALSE`.

SnapshotLen

Длина буфера, содержащего снепшот.

Msg

Адрес переменной (имеющей тип "указатель на `TMTEMsg/MTEMSG`"), куда в случае успеха будет помещен указатель на буфер, содержащий порцию обновлений по открытой таблице. Формат буфера описан в приложении 2.

Возвращаемое значение:

В случае успеха функция возвращает дескриптор открытой таблицы (значение большее или равное `MTE_OK`). Полученный дескриптор используется в дальнейшем при вызове функции `MTEAddTable`.

При возникновении ошибки возвращается один из кодов ошибки `MTE_xxxx`. Если возвращен код ошибки `MTE_TSMR`, поле Data структуры `Msg` содержит текст сообщения об ошибке длиной `DataLen` символов.

Следующий фрагмент кода демонстрирует выборочное открытие таблицы «Заявки» из снепшота:

C++

```
int32 Idx;              // Инициализирована вызовом MTEConnect
MTEMsg *Msg;
char *DataPtr;
char *Snapshot;
int32 Len;
int32 HSecurs, HTrades;
...
HSecurs = MTEOpenTable(Idx, "SECURITIES", "EQBR    ", 1 /*True*/,
      &Msg);
// Обработка полученных данных
...
HTrades = MTEOpenTable(Idx, "TRADES", "", 0 /*False*/, &Msg);
```

```
// Обработка полученных данных
...
// Здесь произошел сбой, сохраняем снепшот в файл и закрываем
таблицы
MTEGetSnapshot(Idx, &Snapshot, &Len);
MTECloseTable(Idx, HSecurs);
MTECloseTable(Idx, HTrades);
...
// Начинаем восстановление, загружаем снепшот из файла и
открываем таблицы
HSecurs = MTEOpenTable(Idx, “SECURITIES”, “EQBR    “,
      1 /*True*/, &Msg);
// Таблица SECURITIES открыта с «нуля», обработка данных
...
HTrades = MTEOpenTableAtSnapshot(Idx, “TRADES”, “”, Snapshot,
      Len, &Msg);
// Таблица TRADES открыта из снепшота, обработка данных
...
do {
  MTEAddTable(Idx, HSecurs, 0);
  MTEAddTable(Idx, HTrades, 1);
  MTERefresh(Idx, &Msg);
  DataPtr = (char *)(Msg + 1);
  // Обработка обновлений
  ...
} while (!Terminated);

MTECloseTable(Idx, HSecurs);
MTECloseTable(Idx, HTrades);
```

Pascal

```
Idx: Integer;           // Инициализирована вызовом MTEConnect
Msg: PMTEMsg;
HSecurs, HTrades: Integer;
Snapshot: PAnsiChar;
Len: Integer;
DataPtr: PAnsiChar;
...
HSecurs := MTEOpenTable(Idx, 'SECURITIES', 'EQBR    ', True,Msg);
// Обработка полученных данных
...
HTrades := MTEOpenTable(Idx, 'TRADES', '', False, Msg);
// Обработка полученных данных
...
// Здесь произошел сбой, сохраняем снепшот в файл и закрываем
таблицы
MTEGetSnapshot(Idx, Snapshot, Len);
MTECloseTable(Idx, HSecurs);
MTECloseTable(Idx, HTrades);
...
// Начинаем восстановление, загружаем снепшот из файла и
открываем таблицы
HSecurs := MTEOpenTable(Idx, 'SECURITIES', 'EQBR    ', True,Msg);
// Таблица SECURITIES открыта с «нуля», обработка данных
...
HTrades := MTEOpenTableAtSnapshot(Idx, 'TRADES', '', Snapshot,
Len, Msg);
// Таблица TRADES открыта из снепшота, обработка данных
...
repeat
  MTEAddTable(Idx, HSecurs, 0);
  MTEAddTable(Idx, HTrades, 1);
```

```
  MTERefresh(Idx, Msg);
  DataPtr := @Msg.Data;
  // Обработка обновлений
  ...
until Terminated;

MTECloseTable(Idx, HSecurs);
MTECloseTable(Idx, HTrades);
```

Завершение сеанса связи

По окончании работы с рынком клиент должен вызвать функцию `MTEDisconnect`.

C++

```
int32 WINAPI MTEDisconnect(int32 Idx);
```

Pascal

```
function MTEDisconnect(Idx: Integer): Integer; stdcall;
```

Аргументы:

Idx

Дескриптор соединения, полученный с помощью вызова `MTEConnect`, которое надо закрыть.

Возвращаемое значение:

Один из кодов ошибки `MTE_xxxx`.

Пример:

Закрываем соединение, имеющее дескриптор Idx.

C++

```
int32 Idx; // Инициализирована вызовом MTEConnect
int32 Err;
...
Err = MTEDisconnect(Idx);
if (Err != MTE_OK)
   fprintf(stderr, "Ошибка: %s\n", MTEErrorMsg(Err));
else
   fprintf(stdout,"Сеанс работы с рынком завершен\n");
```

Pascal

```
Idx: Integer;           // Инициализирована вызовом MTEConnect
Err: Integer;
...
Err := MTEDisconnect(Idx);
if Err <> MTE_OK then Writeln(MTEErrorMsg(Err)
  else Writeln('Сеанс работы с рынком завершен');
```

Сообщения об ошибках

Все функции библиотеки возвращают коды ошибок MTE_xxxx. Для получения текстового описания по коду ошибки могут использоваться функции `MTEErrorMsg` или `MTEErrorMsgEx`.

C++

```
char * WINAPI MTEErrorMsg(int32 ErrorCode);
char * WINAPI MTEErrorMsgEx(int32 ErrorCode, char *Language);
```

Pascal

```
function MTEErrorMsg(ErrCode: Integer): LPSTR; stdcall;
function MTEErrorMsgEx(ErrCode: Integer; Language: PAnsiChar): LPSTR;
      stdcall;
```

Аргументы:

ErrorCode

Один из кодов MTE_xxxx.

Language

Требуемый язык, на котором должно быть получено сообщение об ошибке. Допустимые значения: “`English`”, “`Russian`”, “`Ukrainian`”. Если задан недопустимый язык, будет получено сообщение на английском языке. Функция `MTEErrorMsg` всегда возвращает сообщение на английском языке.

Возвращаемое значение:

Указатель на ASCIIZ-строку, содержащую текстовое описание ошибки.

Коды ошибок

ID	Код	Описание
MTE_OK	0	Нет ошибок.
MTE_CONFIG	-1	Ошибка в конфигурации: подключение к неправильному серверу, на шлюзовом сервере не указаны сервисы, неверные значения параметров в конфигурационном файле.
MTE_SRVUNAVAIL	-2	Сервер не доступен. Не запущен ASTS Bridge Server, недоступна торговая система, либо нарушена связь.
MTE_LOGERROR	-3	При вызове MTEConnect не удалось создать log-файл.
MTE_INVALIDCONNECT	-4	Задан недопустимый дескриптор соединения. Не было вызова MTEConnect, либо уже был вызвана функция MTEDisconnect.
MTE_NOTCONNECTED	-5	Соединение с указанным дескриптором было разорвано вследствие возникновения ошибки (не в результате вызова MTEDisconnect). Ошибка в ASTS Bridge Server, торговая система завершила работу, либо нарушена связь.
MTE_WRITE	-6	Ошибка записи в порт. Ошибка в ASTS Bridge Server, либо нарушена связь.
MTE_READ	-7	Ошибка чтения из порта. Ошибка в ASTS Bridge Server, либо нарушена связь.
MTE_TSMR	-8	Ошибка на уровне протокола взаимодействия с торговой системой, либо торговая система недоступна.
MTE_NOMEMORY	-9	Недостаточно памяти для выполнения операции.
MTE_ZLIB	-10	Ошибка при сжатии/распаковке передаваемых данных.
MTE_PKTINPROGRESS	-11	Была вызвана функция MTEAddTable без последующего вызова MTERefresh. Во время сборки пакета запросов на обновление вызов других функций библиотеки невозможен.
MTE_PKTNOTSTARTED	-12	Была вызвана функция MTERefresh без предварительного вызова MTEAddTable. Сначала необходимо сформировать пакет запросов на обновление.
MTE_FATALERROR	-13	Непредвиденная критическая ошибка.
MTE_INVALIDHANDLE	-14	Неверный дескриптор таблицы. Дескриптор не был получен вызовом MTEOpenTable, либо таблица уже закрыта с помощью MTECloseTable.
MTE_DSROFF	-15	Связь по последовательному порту нарушена (отсутствует сигнал DSR). Возможно нарушена целостность последовательного кабеля, либо последовательный порт закрыт

		на одной из сторон соединения. Применимо к устаревшим версиям шлюза.
MTE_UNKNOWN	-16	Во время выполнения функции произошла непредвиденная ошибка.
MTE_BADPTR	-17	Одной из функций MTExxxx() в качестве аргумента передан недопустимый указатель.
MTE_TRANSREJECTED	-18	Торговая система обработала запрос и вернула код ошибки. Транзакция не выполнена.
MTE_TEUNAVAIL	-19	Торговая система временно недоступна. Сервер продолжает попытки подключения к ТС или находится в ожидании торговой сессии.
MTE_NOTLOGGEDIN	-20	Клиент пытается отправить запрос через сервер после того, как тот установил новое подключение к Торговой системе. Требуется заново подключить клиента к серверу.
MTE_WRONGVERSION	-21	Сервер не поддерживает эту версию клиентской библиотеки.
MTE_LOGON	-30	При соединении с сервером были указаны неверные регистрационные параметры: USERID, PASSWORD и т.п.
MTE_TOOSLOWCONNECT	-31	Слишком медленный канал связи не дает корректно завершить процедуру коннекта/реконнекта.
MTE_CRYPTO_ERROR	-32	Ошибка при шифровании/расшифровке, создании/проверке ЭЦП.
MTE_THREAD_ERROR	-33	Клиент пытается использовать одно соединение в двух потоках. Например, пытается вызвать функцию MTExxxx() в то время, как ещё не завершил работу предыдущий вызов MTExxxx().
MTE_NOTIMPLEMENTED	-34	Вызываемая функция отсутствует в данной версии клиентской библиотеки.
MTE_ABANDONED	-35	Возвращается функцией MTEDisconnect (вызванной в другом потоке), в случае если рабочий поток был остановлен с помощью вызова TerminateThread.

Приложение 1. Формат буфера для функций MTEStructure, MTEStructure2 и MTEStructureEx

Поле Data структуры TMTEMsg/MTEMSG (описание структуры см. в разделе «Получение описания информационных объектов»), указатель на которую возвращает функция `MTEStructure`, имеет следующий формат (описание элементарных типов String, Integer и т.п. см. прил. 4; в случае структуры, возвращаемой функцией `MTEStructure`, перед каждым полем типа String передаётся 4 байта, содержащие длину этой строки). Поля и значения, передающиеся только в функциях `MTEStructure2 (аналогична вызову MTEStructureEx с Version=2) и MTEStructureEx с Version>=2`, помечены красным цветом:

```
        поле                    тип
TInterface:
        ИмяИнтерфейса           String
        ЗаголовокИнтерфейса     String
        ОписаниеИнтерфейса      String          // только MTEStructureEx с Version>=2
        НомерMsgSet             String          // только MTEStructureEx с Version>=4
        ПеречислимыеТипы        TEnumTypes
        Таблицы                 TTables
        Транзакции              TTransactions
```

Описание информационных объектов состоит из трех блоков: описание перечислимых типов, таблиц и транзакций.

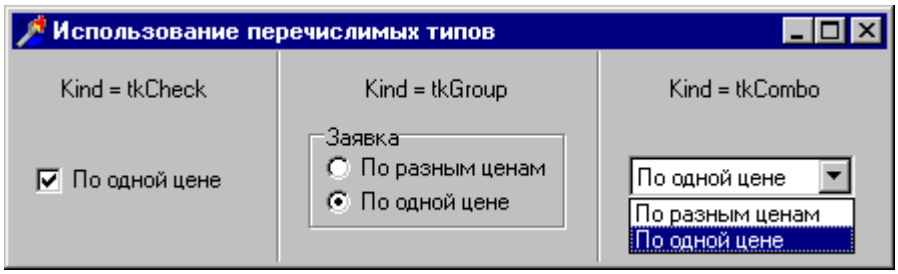
```
TEnumTypes:
        КолвоТипов              Integer
        Тип1                    TEnumType
        Тип2                    TEnumType
        ...
        ТипN                    TEnumType

TEnumType:
        Имя                     String
        Заголовок               String
        Описание                String          // только MTEStructureEx с Version>=2
        Размер                  Integer
        Тип                     TEnumKind
        КолвоКонстант           Integer
        Константа1              TEnumConst
        Константа2              TEnumConst
        ...
        КонстантаN              TEnumConst

TEnumConst для MTEStructure:
        Строка                  string          // в формате «Значение=ДлинноеОписание»


TEnumConst для MTEStructureEx с Version>=2:
        Значение                string
        ДлинноеОписание         string
        КраткоеОписание         string

TEnumKind:                      Integer
        ekCheck = 0
        ekGroup = 1
        ekCombo = 2
```

Перечислимые типы используются для описания допустимых значений полей таблиц и транзакций. Описание типа может выглядеть, например, так:

```
        'TCurrency'             // Имя
        'Валюта'                // Описание
        4                       // Размер
        ekCombo                 // Предпочтительный вид представления - "Тип"
```

```
        3                   // Кол-во констант
        'RUR =Рубли'        // Константа 1
        'USD =Доллары'      // Константа 2
        'EUR =Евро'         // Константа 3
```

Поле "Размер" (=4) указывает размер допустимых значений для полей, имеющих данный тип.
Поле "Тип" (=ekCombo) задает предпочтительный способ представления поля, используемый при создании формы ввода параметров. Например, поле с типом ekCombo может быть представлено в виде списка значений. Возможные варианты показаны на следующем рисунке:



Для функции MTEStructure константы состоят из двух частей - допустимого значения (всегда длиной "Размер") и описания этого значения, разделенных символом равенства (=).
Для функций MTEStructure2 и MTEStructureEx с *Version>=2* значения констант и их описания передаются в отдельных полях.

```
TTables:
        КолвоТаблиц         Integer
        Таблица1            TTable
        Таблица2            TTable
        ...
        ТаблицаN            TTable

TTable:
        Имя                 String
        Заголовок           String
        Описание            String      // только MTEStructureEx с Version>=2
        ИндексСистемы       Integer     // только MTEStructureEx с Version>=2
        Атрибуты            TTableFlags
        ВходныеПоля         TFields
        ВыходныеПоля        TFields

TTableFlags:                Integer
        tfUpdateable    = 1
        tfClearOnUpdate = 2
        tfOrderbook     = 4             // только MTEStructureEx с Version>=2
```

Список входных полей таблицы используется при формировании строки параметров для функции MTEOpenTable.
Список выходных параметров позволяет разбирать буферы, возвращаемые функциями MTEOpenTable и MTERefresh.

Поле «ИндексСистемы» содержит номер подсистемы Торговой системы, которая обрабатывает данный запрос. Пакет обновлений, формируемый вызовами MTEAddTable, может содержать только запросы с одинаковым «ИндексомСистемы». В настоящее время на всех рынках, кроме срочного, этот индекс равен 0, и все таблицы могут обновляться одним вызовом MTERefresh. На срочном рынке работают две подсистемы: собственно торговая и система риск-менеджмента – поэтому все запросы на обновление должны разбиваться на два пакета в соответствие с «Индексом системы».

Атрибуты таблицы могут комбинироваться (то есть значение будет равно 3) и имеют следующие значения:

tfUpdateable - таблица является обновляемой. Для нее можно вызывать функции `MTEAddTable/MTERefresh`;

tfClearOnUpdate - старое содержимое таблицы должно удаляться при получении каждого обновления с помощью функций `MTEAddTable/MTERefresh`.

tfClearOrderbook – таблица имеет формат котировок и должна обрабатываться соответсвующим образом (см. Замечания по работе с таблицами).

```
TFields:
       КолвоПолей          Integer
       Поле1               TField
       Поле2               TField
       ...
       ПолеN               TField

TField:
       Имя                 String
       Заголовок           String
       Описание            String       // только MTEStructureEx с Version>=2
       Размер              Integer
       Тип                 TFieldType
       КолвоДесятичЗнаков  Integer      // только MTEStructureEx с Version>=2
       Атрибуты            TFieldFlags
       ПеречислимыйТип     String
       ЗначениеПоУмолчанию String

TFieldType:                Integer
       ftChar    = 0
       ftInteger = 1
       ftFixed   = 2
       ftFloat   = 3
       ftDate    = 4
       ftTime    = 5
       ftFloatPoint = 6              // только MTEStructureEx с Version>=3

TFieldFlags:               Integer
       ffKey       = 0x01
       ffSecCode   = 0x02
       ffNotNull   = 0x04
       ffVarBlock  = 0x08            // только MTEStructureEx с Version>=2
```

Атрибуты поля (TFieldFlags) могут комбинироваться и имеют следующие значения:

ffKey Поле является ключевым. Строки таблицы с совпадающими значениями ключевых полей должны объединяться в одну строку.

ffSecCode Поле содержит код финансового инструмента. Рекомендуется учитывать данный флаг при автоматизации процедуры определения числа знаков после запятой в числовых полях типа ftFloat.

ffNotNull Поле не может быть пустым.

ffVarBlock Поле входит в группу полей, которые могут повторяться несколько раз.

Примечание. В списке выходных полей таблицы, в отличие от входных, отсутствует поле "ЗначениеПоУмолчанию".
"Размер" задает длину поля в символах.
"КолвоДесятичЗнаков" задает знаков после запятой для полей типа ftFixed.
"ПеречислимыйТип" может содержать имя перечислимого типа, к которому относится поле, или пустую строку.
"Значение по умолчанию" может использоваться при создании формы ввода параметров.
Все поля представлены в текстовом виде в формате торговой системы (см. `MTEExecTrans`).

```
TTransactions:
       КолвоТранзакций     Integer
```

```
        Транзакция1          TTransaction
        Транзакция2          TTransaction
        ...
        ТранзакцияN          TTransaction

TTransaction:
        Имя                  String
        Заголовок            String
        Описание             String       // только MTEStructureEx с Version>=2
        ИндексСистемы        Integer      // только MTEStructureEx с Version>=2
        ВходныеПоля          TFields
```

Список входных полей транзакции используется при формировании строки параметров для функции MTEExecTrans.

ПРИЛОЖЕНИЕ 2. ФОРМАТ БУФЕРА ДЛЯ ФУНКЦИИ MTEOPENTABLE

Поле Data структуры TMTEMsg/MTEMSG (описание структуры см. в разделе «Получение описания информационных объектов»), указатель на которую возвращает функция MTEOpenTable, содержит строки запрошенной таблицы и имеет следующий формат (описание элементарных типов String, Integer и т.п. см. прил. 4):

```
        поле                 тип
TMTETable:
        Ref                  Integer
        КолвоСтрок           Integer
        Строка1              TMTERow
        Строка2              TMTERow
        ...
        СтрокаN              TMTERow
```

Поле "Ref" используется при запросе изменений сразу по нескольким таблицам с помощью функций MTEAddTable/MTERefresh. Оно содержит значение, переданное в качестве третьего параметра функции MTEAddTable(Idx, HTable, Ref). По значению этого поля можно определить, какой таблице (дескриптор HTable) соответствует полученная структура TMTETable. В буфере, возвращаемом MTEOpenTable, значение поля "Ref" не определено.

```
TMTERow:
        КолвоПолей           Byte
        ДлинаДанных          Integer
        НомераПолей          Byte[КолвоПолей]
        ДанныеПолей          Byte[ДлинаДанных]
```

Строки таблицы имеют переменную длину и могут содержать разное число полей.

Поле "КолвоПолей" содержит число полей таблицы, присутствующих в данной строке. Если значение это поля равно 0, в строке присутствуют все поля таблицы (см. MTEStructure).

Поле "ДлинаДанных" содержит суммарный размер полей таблицы в данной строке.

Поле "НомераПолей" имеет переменную длину. Его размер равен значению поля "КолвоПолей". Поле содержит номера полей (по одному байту на номер), присутствующих в данной строке. Номер поля соответствует порядковому номеру выходного поля в описании информационных объектов (см. MTEStructure). Если "КолвоПолей" равно 0, значит "НомераПолей" отсутствует, а в качестве номеров полей следует брать последовательность номеров 0, 1, 2, 3 … N.

Поле "ДанныеПолей" (размером "ДлинаДанных" байт) содержит набор значения полей таблицы. Количество полей определяются значением "КолвоПолей", а их суммарная длина - "ДлинаДанных". Длина и тип каждого конкретного поля определяются в описании информационных объектов (см. MTEStructure). Все поля представлены в текстовом виде в формате торговой системы (см. приложение 5).

Пример:

Допустим в описании информационных объектов, полученном с помощью MTEStructure, определена таблица "Сделки" со следующими выходными полями:

```
TRADES                    // "Сделки"
  TradeNum: ftInteger(12) // Номер сделки
  TradeTime: ftChar(6)    // Время сделки
  BuySell: ftChar(1)      // "B" - покупка, "S" - продажа
  SecCode: ftChar(17)     // код инструмента
  Price: ftFloat(9)       // цена
  Qty: ftInteger(10)      // кол-во лотов
```

Вызвана функция:

```
MTEOpenTable(Idx, 'TRADES', '', True, Msg);
```

В результате в поле Msg.Data содержится следующая информация

```
{
  0x00000000,             // Поле "Ref"
  0x00000002,             // Получено 2 строки
  0x04,                   // В первой строке 4 поля
  0x00000030,             // Длина данных 48 байт
  #0#3#4#5,               // Номера полей 0, 3, 4, 5:
// это поля "TradeNum", "SecCode", "Price", "Qty" из описания
  '0000001205670CURRUSD000000TOD0002579000000000037'
// Значения полей: 120567, "0CURRUSD000000TOD", 25.79, 37
  0x02,                   // Во второй строке 2 поля
  0x17,                   // Длина данных 23 байта
  #1#3,                   // Номера полей 1, 3:
// это поля "TradeTime" и "SecCode" из описания
  '1029530CURRUSD000000TOM'
// Значения полей: "10:29:53" и "0CURRUSD000000TOM"
}
```

Приложение 3. Формат буфера для функции MTERefresh

Поле Data структуры TMTEMsg/MTEMSG (описание структуры см. в разделе «Получение описания информационных объектов»), указатель на которую возвращает функция `MTEOpenRefresh`, содержит несколько таблиц торговой системы и имеет следующий формат (описание элементарных типов String, Integer и т.п. см. прил. 4):

поле	тип
TMTETables:	
КолвоТаблиц	Integer
$Таблица_1$	TMTETable
$Таблица_2$	TMTETable
...	
$Таблица_N$	TMTETable

Таким образом, буфер содержит несколько таблиц. Формат буфера таблицы описан в приложении 2.

Приложение 4. Элементарные типы

Для представления элементарных типов в библиотеке MTESRL используются следующие структуры:

`Byte`

Один байт.

`Integer`

Четыре байта в формате процессоров x86 (сначала наименее значащий байт).

`String`

Структура следующего вида:

```
ДлинаСтроки: Integer
ТекстСтроки: Byte[ДлинаСтроки]
```

`Byte[N]`

Массив байт длиной N.

Приложение 5. Форматирование табличных данных, возвращаемых торговой системой

Возвращаемые торговой системой табличные данные, в зависимости от типа поля форматируются следующим образом:

ftChar

Текстовая строка, дополненная справа пробелами до длины, указанной в описании поля.

ftInteger

Значения полей типа ftInteger (целые числа) передаются в текстовом представлении и дополняются слева нулями до нужного размера.

ftFloat

(в html описании структуры интерфейса – тип «PRICE»)

Значения полей типа ftFloat (вещественные числа) передаются в текстовом представлении без десятичной точки. Количество знаков после десятичной точки в полях типа ftFloat для конкретной ценной бумаги определяется значением поля "DECIMALS" таблицы "SECURITIES".

В полях типа ftFloat обязательно должны присутствовать DECIMALS знаков после запятой. Например, число 465,39 для ценной бумаги с DECIMALS = 4 должно быть представлено как "4653900". Значение "46539" в этом случае будет воспринято торговой системой как 4,6539.

ftFixed

В полях типа ftFixed значения (вещественные числа) также передаются в текстовом представлении без десятичной точки. По умолчанию поля данного типа имеют два знака после десятичной точки. Однако при использовании функции `MTEStructure2` и `MTEStructureEx` с *Version>=2* (см. Приложение 1) в структуре передается точное число десятичных знаков.

ftDate

Значения в полях типа ftDate передаются в виде текстовой строки YYYYMMDD.

ftTime

Значения в полях типа ftTime передаются в виде текстовой строки формата HHMMSS.

ftFloatPoint

(в html описании структуры интерфейса – тип «FLOAT»)

Значения полей типа ftFloatPoint (вещественные числа) передаются в текстовом представлении с десятичной точкой и дополняются слева нулями до нужного размера. Этот тип доступен при получении структуры информационных объектов с помощью `MTEStructureEx` с *Version>=3* (см. Приложение 1). При использовании функций MTEStructure и MTEStructure2 тип передается как строка (ftChar). Положение десятичной

точки внутри числа не фиксировано. Десятичная точка, а также возможный знак числа, учитываются при подсчете длины. Например, ftFloatPoint(9): "001.45712", ftFloatPoint(16): "-0000012071000.5".

Примечание

В полях любого типа может быть передано пустое значение (NULL), для этого используется строка пробелов нужной длины.