MoTor®

Библиотека имитационного моделирования биржевых торгов

Интерфейс программирования приложений 22 января 2002 г.

Содержание

Введение

Имитационная модель торгов (ИМТ) предназначена для использования в торговых тренажёрах и системах детального анализа хода торгов. Она позволяет воспроизводить реальный ход торгов и создавать новые игровые ситуации.

ИМТ генерирует биржевые торги путём имитации действий участников. Каждый участник торгов задаётся в ИМТ отдельным уникальным алгоритмом — *моделью участника*, которая «наблюдает» за ходом торгов и вырабатывает решения о подаче и снятии заявок. Модель участника строится автоматически по действиям реального биржевого игрока.

Для сопоставления заявок и заключения сделок модель использует торговый автомат, являющийся упрощённым аналогом торгового автомата ММВБ.

Настоящий документ описывает программный интерфейс библиотеки MoTor, позволяющий включать имитационную модель торгов в различные приложения.

Терминология

Событие (Event)	Ввод заявки (enter), снятие заявки (withdraw), сделка (trade) или управляющее событие (control). Управляющие события — моменты начала и завершения периодов открытия, закрытия, нормальных торгов, аукциона.
Торговая сессия (Session)	Последовательность всех событий на торгах данным финансовым инструментом в течение одного торгового дня.
Реальная сессия (Real Session)	Торговая сессия, полученная в ходе реальных торгов данным финансовым инструментом на данной торговой площадке.
Модельная сессия (Model Session)	Торговая сессия, сгенерированная в процессе имитации торгов данным финансовым инструментом с помощью модели торгов.
Торговый автомат (Engine)	Алгоритм сопоставления (matching) заявок, выстраивающий заявки в очередь и генерирующий сделки для данной торговой сессии.
Протокол торгов (Session Events)	Список всех событий, произошедших в сессии, в их хронологической последовательности.
Статистика сессии (Statistics)	Совокупность характеристик сессии, допускающих агрегирование по времени. Например, объём торгов, число сделок, минимальная цена, средневзвешенная цена, и т. д.
Участник торгов (Participant)	Держатель денежной и депозитарной позиции, имеющий возможность наблюдать за ходом торгов, вводить и снимать заявки. В отличие от реальной торговой системы, структурное деление участников на дилеров, трейдеров и клиентские счета не производится.
Досье участника (Dossier)	Совокупность характеристик участника торгов: объём покупок и продаж, число сделок, прибыль, и т. д.

Воспроизведение торгов (Replay)	Процесс генерации выходного потока сделок по заданной входной последовательности ввода и снятия заявок. Воспроизводить можно как модельные, так и реальные сессии. В процессе воспроизведения вся информация о сессии (статистика, досье участников, и др.) обновляется по каждому событию.
Верификация торгового автомата (Engine Verification)	Процесс воспроизведения реальной торговой сессии, при котором генерируемая последовательность сделок сравнивается с реальной. Обнаруженные расхождения могут свидетельствовать о неадекватной реализации торгового автомата в библиотеке MoTor.
Модель участника, модельный участник, робот (Participant Model)	Параметрический алгоритм принятия решений данным участником. В ходе модельных торгов робот с некоторой периодичностью анализирует текущую ситуацию и принимает решение о своих действиях: подать заявку, снять заявку, или ничего не делать. Параметры и *тип* модели можно либо задавать явно, либо определять автоматически путем оптимизации, при этом поведение робота точно или приближённо «подгоняется» под поведение его реального прототипа. Модель участника называется *настроенной*, если все её параметры заданы.
Имитационная модель торгов, ИМТ (Trade Model)	Набор модельных участников, достаточный для имитации торгов в заданной сессии. Модель торгов может состоять из модельных участников различных типов. Модель торгов называется *настроенной*, если все входящие в неё роботы настроены на соответствующих им реальных участников.
Настройка модели торгов (Model Tune)	Процесс воспроизведения торгов, в ходе которого все модельные участники «подгоняются» под поведение их реальных прототипов. Результатом настройки является вычисление всех внутренних параметров для всех модельных участников.
Имитация торгов (Model Run)	Процесс отработки настроенной модели торгов, в ходе которого модельные участники поочерёдно принимают решения о своих действиях. Модель собирает их в единый поток событий и направляет торговому автомату. В результате формируется модельная сессия. В отличие от воспроизведения, при имитации торгов последовательность событий генерируется заново и в общем случае не обязана совпадать с исходной.
Имитационная модель рынка, Рынок, (Market)	Совокупность имитационных моделей торговых сессий, проходивших в течение одного торгового дня по различным финансовым инструментам.

Объекты и структуры данных

Основные объекты

Все функции библиотеки работают с объектами четырёх основных типов — торговыми сессиями, моделями торгов, сериями имитаций и таблицами данных. Основные объекты создаются и уничтожаются пользователем библиотеки. Пользователь не имеет доступа к их внутренней структуре.

```
typedef const void* mtObject;   // интерфейсный объект
typedef mtObject mtSession;     // торговая сессия
typedef mtObject mtModel;       // модель торгов
```

Простые типы данных

Простые типы данных вводятся для повышения гибкости и удобочитаемости кода, упрощают возможные в будущем переходы на новые типы данных (например расширение представления цен с float до double).

```
typedef const char* mtString;      // текстовая строка
typedef time_t mtTime;             // дата-время
typedef double mtModelTime;        // mtTime + дробная часть секунды
typedef double mtPrice;            // цена
typedef double mtValue;            // объём в деньгах
typedef double mtQuantity;         // количество (объём в штуках)
typedef short mtBool;              // булевское 0/1
typedef long mtNumber;             // целое число
typedef unsigned long mtEventType; // тип события
typedef unsigned long mtModelClass; // тип модели участника
```

Тип mtEventType предназначен для представления всевозможных типов событий, хранения битовой маски типов и статусов заявки и классификации действий участников:

```
   // типы основных событий
const int mtET_ORDER         = 1;     // 1-заявка 0-не заявка
const int mtET_WITHDRAW      = 1<<4;  // W тип события: 1-снятие 0-заявка
const int mtET_TRADE         = 1<<11; // T тип события: 1-сделка 0-заявка
   // типы управляющих событий
const int mtET_START         = 1<<16; // ( старт торгов
const int mtET_STOP          = 1<<17; // ) стоп торгов
const int mtET_AUCT_BEGIN    = 1<<18; // [ перед совершением всех аукционных сделок
const int mtET_AUCT_END      = 1<<19; // ] после совершения  всех аукционных сделок
const int mtET_NORMAL_BEGIN  = 1<<20; // < начало нормального периода торгов
const int mtET_NORMAL_END    = 1<<21; // > конец нормального периода торгов
const int mtET_OPEN_BEGIN    = 1<<22; // [ начало периода открытия
const int mtET_OPEN_END      = 1<<23; // ] конец периода открытия (перед сделками)
const int mtET_CLOSE_BEGIN   = 1<<24; // [ начало периода закрытия
const int mtET_CLOSE_END     = 1<<25; // ] конец периода закрытия (перед сделками)
   // подтипы заявок
const int mtET_BID           = 1<<1;  // B/S заявка: 1-бид 0-аск
const int mtET_MARKET        = 1<<2;  // M заявка: 1-рыночная 0-лимитная
```

```
const int mtET_ACTIVE        = 1<<3;  // ! заявка: 1-активная 0-пассивная
const int mtET_FILL_AND_KILL = 1<<5;  // F тип заявки: заполнить и удалить
const int mtET_ALL_OR_NOTHING= 1<<6;  // N тип заявки: всё или ничего
   // состояния заявок
const int mtET_MATCHED       = 1<<7;  // + статус заявки: реализовавшаяся
const int mtET_WITHDRAWN     = 1<<8;  // - статус заявки: снятая
const int mtET_QUOTED        = 1<<9;  // * статус заявки: стоящая в котировках
   // подтипы сделок
const int mtET_TRADE_BIDINIT = 1<<12; // b/s инициатор сделки: 1-бид 0-аск
   // периоды заявок и сделок
const int mtET_OPEN_PERIOD   = 1<<13; // O заявка или сделка периода открытия
const int mtET_CLOSE_PERIOD  = 1<<14; // C заявка или сделка периода зыкрытия
const int mtET_AUCT_PERIOD   = 1<<15; // A заявка или сделка периода аукциона
   // действия участников
const int mtET_DO_NOTHING    = 0;     // бездействие
const int mtET_NEXT_DECISION = 1<<26; // следующее принятие решения
```

Тип mtModelClass — допустимые типы встроенных моделей участника. В дальнейшем предполагается расширение множества типов моделей.

```
const int mtMC_NO_MODEL      = 0;    // нет модели
   // интерполяционные модели с поправкой цен:
const int mtMC_PROTOCOL      = 1;    // без поправки цен
const int mtMC_PROTOCOL_CBO  = 2;    // относительно противоположной очереди
const int mtMC_PROTOCOL_MPP  = 3;    // относительно (bid+ask)/2
const int mtMC_PROTOCOL_MOV  = 4;    // относительно скользящей средней
const int mtMC_PROTOCOL_ORD  = 5;    // отн. своей или противоположной очереди
const int mtMC_PROTOCOL_LTP  = 6;    // относительно цены последней сделки
const int mtMC_PROTOCOL_SBO  = 7;    // относительно своей очереди
```

Возвратные функции

Возвратные функции (callback functions) — это распространённый способ определять обработчики событий в не объектно-ориентированных библиотеках.

```
typedef int (*mtVoidFunc) ();
                                     // внешний обработчик холостого ожидания
typedef int (*mtEventFunc) (mtEvent &event);
                                     // внешний обработчик события
typedef int (*mtEventTypeFunc) (mtEventType &etype);
                                     // внешний обработчик управляющего события
```

Вспомогательные объекты

Вспомогательные объекты используются для передачи информации в функции и из функций библиотеки. В отличие от основных объектов их внутренняя структура открыта для пользователя библиотеки.

Финансовый инструмент: информация о финансовом инструменте, торгуемом в данной сессии. Устанавливается и считывается функциями mtSessionSetSecurity и mtSessionGetSecurity.

```
struct mtSecurity {
   mtString Name;           // код инструмента  в торговой системе
   mtPrice PriceTick;       // шаг цены
```

```
   int Decimals;           // число десятичных знаков цены
   double Tax;             // комиссионные в процентах
   mtTime OpenBegin;       // начало периода открытия
   mtTime OpenEnd;         // конец периода открытия
   mtTime NormalBegin;     // начало нормального периода
   mtTime NormalEnd;       // конец нормального периода
   mtTime CloseBegin;      // начало периода закрытия
   mtTime CloseEnd;        // конец периода закрытия
};
```

Уровень цен в очереди: информация о всех заявках, стоящих в очереди в текущий момент времени по одной и той же цене.

```
struct CPriceLevel {
   mtPrice price;          // цена
   mtQuantity quantity;    // суммарный видимый объём заявок по данной цене
   mtQuantity total;       // суммарный общий объём заявок по данной цене
   mtOrder* first;         // заявка, пришедшая первой по данной цене
   CPriceLevel* next;      // следующий уровень в очереди
   CPriceLevel* prev;      // предыдущий уровень в очереди
};
```

Заявка, подаваемая в торговую систему: информация, сообщаемая участником при подаче заявки в торговую систему с помощью функции mtEngineOrderEnter.

```
struct mtOrderEnter {
   mtTime Time;            // время подачи заявки, число секунд с полуночи
   mtPrice Price;          // цена заявки
   mtQuantity Quantity;    // объём заявки
   mtQuantity QtyHidden;   // скрытый объем
   mtEventType Mask;       // типы и статусы заявки
   mtString PartID;        // символьный идентификатор участника
   long PartNo;            // порядковый номер участника (если PartID пуст)
   long OrderNo;           // номер заявки в торговой системе
};
```

Заявка, стоящая в очереди: первая часть записи совпадает с mtOrderEnter, остальные данные служат для хранения текущей очереди в виде связного списка.

```
struct mtOrder {
   mtTime Time;            // время подачи заявки, число секунд с полуночи
   mtPrice Price;          // цена заявки
   mtQuantity Quantity;    // объём заявки
   mtQuantity QtyHidden;   // скрытый объем
   mtEventType Mask;       // типы и статусы заявки
   mtString PartID;        // символьный идентификатор участника (если PartNo<0)
   long PartNo;            // порядковый номер участника (если PartID пуст)
   long OrderNo;           // номер заявки в торговой системе
   mtQuantity QtyBase;     // основной объем (для заявок со скрытым объемом)
   mtOrder *Next;          // следующая заявка в очереди
   mtOrder *Prev;          // предыдущая заявка в очереди
   mtOrder *PartNext;      // следующая заявка этого участника
   mtOrder *PartPrev;      // предыдущая заявка этого участника
   CPriceLevel *Level;     // уровень цены
```

```
};
```

Запись в протоколе событий: информация о транзакции — заявке, снятии или сделке. Запись в протокол выполняется автоматически, считывание производится функциями mtEngineGetEventsCount и mtEngineGetEvent.

```
struct mtEvent {
   mtTime Time;            // время события, число секунд с полуночи
   mtTime FinalTime;       // время снятия или реализации заявки
   mtPrice Price;          // цена события
   mtQuantity Quantity;    // объём события
   mtQuantity QtyHidden;   // скрытый объем заявки
   mtEventType Mask;       // типы и статусы события
   long PartNo;            // номер участника-инициатора
   long EventNo;           // номер события в протоколе событий
   long BidNo;             // для сделки: номер заявки bid в протоколе событий
   long AskNo;             // для сделки: номер заявки ask в протоколе событий
};
```

Скользящая средневзвешенная, вычисляющаяся по ходу торгов. Средневзвешенные присоединяются к сессии функцией mtSessionSetMovingAverage и считываются функцией mtSessionGetMovingAverage во время торгов или после.

```
struct mtMovingAverage {
   mtTime tLength;         // длина периода в секундах
   long iFirstTrade;       // номер первой сделки в периоде
   long iLastTrade;        // номер последней сделки в периоде
   mtValue vSum;           // суммарный объём (в деньгах)
   mtQuantity qSum;        // суммарное количество (в штуках)
   long nSum;              // число сделок
   mtPrice pVwap;          // последняя рассчитанная средневзвешенная
};
```

Участок постоянства кусочно-постоянной функции времени. Используется для хранения спрэдов. Вычисленные по ходу торгов спрэды считываются функциями mtSessionSpreadSize и mtSessionGetSpread во время торгов или после.

```
struct mtConstancyInterval {
   mtTime Time;            // время начала участка постоянства
   double Value;           // значение функции на участке постоянства
};
```

Статистика сессии: набор признаков, характеризующих сессию на заданном интервале времени. Выдаётся функцией mtSessionGetStatistics как после, так и во время торгов.

```
struct mtStatistics {
   mtTime tBegin;          // начало интервала времени
   mtTime tEnd;            // конец интервала времени
   mtTime tSpread;         // время существования спрэда
   // текущие объёмные параметры
   mtValue vTurnover;      // объём в деньгах
   mtQuantity qTurnover;   // количество (объём в штуках)
   // текущие ценовые параметры
   mtPrice pMin;           // минимальная цена сделки
   mtPrice pMax;           // максимальная цена сделки
```

```
   mtPrice pLast;           // цена последней сделки
   mtPrice pMidpoint;       // средневзвешенная по времени середина спрэда
   mtPrice pSpread;         // средневзвешенный по времени спрэд
   mtPrice pVolatility;     // средневзв. по времени квадрат откл. середины спрэда
   // число событий
   long nTrades;            // число сделок
   long nBidOrders;         // число заявок покупателей
   long nAskOrders;         // число заявок продавцов
   long nBidWithdraw;       // число снятий заявок покупателями
   long nAskWithdraw;       // число снятий заявок продавцами
   // спрэд непосредственно до текущего события
   mtPrice pBidBefore;      // лучший бид перед вызовом внешней AfterEvent
   mtPrice pAskBefore;      // лучший аск перед вызовом внешней AfterEvent
   mtTime tBidBefore;       // время установления лучшего бида
   mtTime tAskBefore;       // время установления лучшего аска
   // изменчивость спрэда
   mtPrice pBidChange;      // среднее изменение лучшего бида
   mtPrice pAskChange;      // среднее изменение лучшего аска
   long nBidChange;         // число изменений лучшего бида
   long nAskChange;         // число изменений лучшего аска
   long nBestBidWithdraw;   // число бидов, снимаемых по цене лучшей котировки
   long nBestAskWithdraw;   // число асков, снимаемых по цене лучшей котировки
   // ликвидность (сумма объёмов заявок в очереди)
   mtQuantity qBidLiq;      // текущая ликвидность покупки
   mtQuantity qAskLiq;      // текущая ликвидность продажи
   mtQuantity qBidLiqAvr;   // средневзвешенная по времени ликвидность покупки
   mtQuantity qAskLiqAvr;   // средневзвешенная по времени ликвидность продажи
   // ликвидность лучшей цены (сумма объёмов заявок, стоящих в очереди по лучшей цене)
   mtQuantity qBestBidLiq;      // текущая ликвидность покупки
   mtQuantity qBestAskLiq;      // текущая ликвидность продажи
   mtQuantity qBestBidLiqAvr;   // средневзвешенная по времени ликвидность покупки
   mtQuantity qBestAskLiqAvr;   // средневзвешенная по времени ликвидность продажи };
```

Краткая статистика по интервалу времени. Используется для хранения предыстории сессии. Предыстория выдается функциями mtSessionGetHistoryTotal, mtSessionGetHistorySize, mtSessionGetHistory.

```
struct mtShortStat {
   mtTime tBegin;           // начало интервала времени, содержит дату и время
   mtPrice pOpen;           // цена открытия
   mtPrice pClose;          // цена закрытия
   mtPrice pMin;            // минимальная цена
   mtPrice pMax;            // максимальная цена
   mtValue vTotal;          // суммарный объём в рублях
   mtQuantity qTotal;       // суммарный объём в штуках
   long nTotal;             // число сделок
};
```

Входящие позиции участника. Задаются функцией mtSessionSetPartInitial при инициализации состояния участника перед торгами.

```
struct mtPartInitial {
   mtQuantity qInitDepo;    // входящая позиция по инструменту
```

```
   mtValue vInitCash;       // входящая позиция по деньгам
   mtBool CheckLimits;      // проверять лимиты при постановке заявок?
};
```

Досье участника. Формируется по ходу торгов и может быть получено с помощью функции mtSessionGetPartDossier как после, так и во время торгов.

```
struct mtPartDossier {
      // поддержка текущих позиций
   mtQuantity qDepo;        // текущая позиция по инструменту
   mtValue vCash;           // текущая позиция по деньгам
   mtQuantity qTotalAsk;    // текущее количество бумаг в очереди на продажу, шт.
   mtValue vTotalAsk;       // текущее количество денег в очереди на продажу, руб.
   mtQuantity qTotalBid;    // текущее количество бумаг в очереди на покупку, шт.
   mtValue vTotalBid;       // текущее количество денег в очереди на покупку, руб.
                                    // вычисляемые признаки участника
   mtQuantity qMaxDepo;     // максимальный расход бумаг = MAX (qInitDepo - qDepo + qTotalAsk)
   mtValue vMaxCash;        // максимальный расход денег = MAX (vInitCash - vCash + vTotalBid)
   mtQuantity qBuy;         // купленное количество
   mtQuantity qSell;        // проданное количество
   mtValue vBuy;            // купленный объём
   mtValue vSell;           // проданный объём
   long nTrades;            // число сделок
   long nBuy;               // число покупок
   long nSell;              // число продаж
   long nBidOrders;         // число заявок покупки
   long nAskOrders;         // число заявок продажи
   long nMarketBidOrders;   // число рыночных заявок покупки
   long nMarketAskOrders;   // число рыночных заявок продажи
   long nActiveBidOrders;   // число активных заявок покупки
   long nActiveAskOrders;   // число активных заявок продажи
   long nBidWithdraw;       // число снятий заявок покупки
   long nAskWithdraw;       // число снятий заявок продажи
   mtValue vProfit;         // прибыль с оценкой остатка по цене закрытия
   mtValue vTax;            // уплаченные комиссионные от сделок
   mtValue vSpreadLoss;     // суммарная потеря на спрэде
   mtValue vSpreadWin;      // суммарный выигрыш на спрэде
};
```

Решение модельного участника. Используется для передачи торговых решений в функциях модельного участника торгов TuneDecision и PlayDecision.

```
struct mtDecision {
      // поля, заполняемые моделью участника
   mtPrice Price;           // цена подаваемой или снимаемой заявки
   mtQuantity Quantity;     // объём подаваемой или снимаемой заявки
   mtEventType Mask;        // флаги заявки и тип решения
   mtTime EnterTime;        // время подачи снимаемой заявки
   mtModelTime NextTime;    // время следующего принятия решения
      // поля, заполняемые моделью торгов
   long PartNo;             // номер участника, принимающего решение
   mtModelTime PlayTime;    // время реализации решения
   mtDecision* Next;        // следующее решение в очереди
```

```
};
```

Описание формата trd-файла. Содержит номера информационных полей в записях, нумерация полей начинается с нуля. Номера необязательных полей могут быть отрицательны, тогда это поле не считывается. Используется при загрузке и сохранении протоколов в функциях mtSessionLoad, mtSessionSave.

```
const int SECID_LEN = 32;    // длина строки для хранения кода фин.инструмента
struct mtTrdFormat {
   long Type;              // тип записи: H-истор I-инстр B-бид S-аск T-сдел W-снят
                           // D-данные O-откр N-норм C-закр E-конец
   // общие поля для всех строк кроме D
   long Time;              // время события
   long SecID;             // идентификатор финансового инструмента
   // поля в описании инструмента (I)
   long SecPriceTick;      // шаг цены
   long SecDecimals;       // число десятичных знаков цены
   long SecTax;            // комиссионные в процентах
   long SecOpenBegin;      // начало периода открытия
   long SecOpenEnd;        // конец периода открытия
   long SecNormalBegin;    // начало нормального периода
   long SecNormalEnd;      // конец нормального периода
   long SecCloseBegin;     // начало периода закрытия
   long SecCloseEnd;       // конец периода закрытия
   // поля в строке истории (H)
   long HistPeriod;        // длина интервала в истории
   long HistOpen;          // цена открытия
   long HistClose;         // цена закрытия
   long HistMin;           // минимальная цена
   long HistMax;           // максимальная цена
   long HistValue;         // суммарный объём в рублях
   long HistQty;           // суммарный объём в штуках
   long HistCount;         // число сделок
   // поля в строке заявки (события B и S)
   long OrderID;           // идентификатор заявки
   long OrderPartID;       // идентификатор участника, подавшего заявку
   long OrderPrice;        // цена заявки
   long OrderQty;          // объём заявки
   long OrderQtyHidden;    // скрытый объём заявки
   long OrderFlags;        // флаги: M-рыночная F-снять-остаток N-всё-или-ничего
   // поля в строке снятия заявки W
   long WithdrawOrderID;   // идентификатор снимаемой заявки
   long WithdrawPartID;    // идентификатор участника, снимающего заявку (необяз)
   long WithdrawPrice;     // цена снимаемой заявки (необязательное поле)
   long WithdrawQty;       // объём снимаемой заявки (необязательное поле)
   // поля в строке сделки T
   long TradeID;           // идентификатор сделки (необязательное поле)
   long TradeAskID;        // идентификатор аска
   long TradeBidID;        // идентификатор бида
   long TradeAskerID;      // идентификатор участника, подавшего аск (необяз)
   long TradeBiderID;      // идентификатор участника, подавшего бид (необяз)
```

```
   long TradePrice;         // цена сделки
   long TradeQty;           // объём сделки
   // дополнительная информация и опции
   long VerifyTrades;       // производить ли верификацию сделок
   long AuxDate;            // дата генерации trd-файла
   long AuxDescription;     // описание trd-файла
   char SecFix [SECID_LEN]; // бумага, по которой отбираются события (если SecID>=0)
};
```

Функции библиотеки MoTor

Группы функций. Функции библиотеки поделены на следующие группы:

mtSession{…}	функции высокого уровня для обработки торговых сессий
mtEngine{…}	функции низкого уровня для работы с торговым автоматом
mtModel{…}	функции модели торгов
mtModelPart{…}	функции модельного участника торгов
mtMarket{…}	функции рынка
mtRegister{…}	функции реестра моделей участника торгов
mtFormat{…}	функции установки формата входных данных
mtError{…}	функции диагностики и обработки ошибок
mt{…}	вспомогательные функции

Совместимость объектных типов. Модель торгов mtModel включает в себя торговую сессию mtSession. Поэтому, если параметр функции имеет тип mtSession, то вместо него можно передавать также mtModel. Остальные типы не совместимы.

Обмен строками. При передаче строк через функции библиотеки необходимо соблюдать два основных правила.

- Если строка передаётся в функцию библиотеки в качестве входного аргумента как mtString, то эта строка формируется и освобождается вне библиотеки, и функции библиотеки не имеют права её модифицировать.
- Если указатель на строку передаётся в функцию библиотеки в качестве выходного аргумента как mtString*, то эта строка формируется и освобождается внутри библиотеки, и пользовательское приложение не имеет права её модифицировать. При необходимости сохранить или модифицировать полученную строку следует воспользоваться операцией копирования, так как следующий вызов любой библиотечной функции может изменить расположенную по этому указателю строку.

Возвращается код ошибки. Все функции библиотеки возвращают целое число. Если оно отрицательно, то это код ошибки. Неотрицательные значения свидетельствуют о корректном выполнении функции.

Торговые сессии

Все функции, принимающие объект mtSession в качестве первого аргумента, возвращают код ошибки mtER_SESSION, если объект не существует или не является сессией.

Сессию можно воспроизводить произвольное число раз как по шагам, используя функции торгового автомата, так и целиком с помощью функции mtSessionReply.

В процессе воспроизведения по исходной последовательности событий вычисляются текущие очереди, досье участников, статистика сессии. Если требуется, могут также вычисляться скользящие средние и спрэды.

`int mtSessionCreate (mtSession*)`

Функция создаёт сессию и записывает в mtSession указатель на созданную сессию. В случае успешного создания объект mtSession можно передавать другим функциям для дальнейшей работы с сессией. Созданная сессия не имеет очередей и таблиц данных. По окончании работы сессия должна быть удалена функцией mtSessionDelete.

Возвращает код ошибки mtER_CREATE, если не удалось создать сессию.

`int mtSessionDelete (mtSession)`

Удаляет сессию, ранее созданную mtSessionCreate.

Возвращает код ошибки mtER_DELETE, если сессия не может быть удалена (невозможно удалить сессию, входящую в серию имитаций).

`int mtSessionLoad (mtSession, mtString filename, mtTrdFormat*)`

Функция загружает сессию из текстового файла формата TRD с именем mtString, содержащего протокол событий одного торгового дня. Порядок полей в записях файла задаётся структурой mtTrdFormat. Эта структура должна быть инициализирована функцией mtFormatTrdInit или загружена из ini-файла функцией mtFormatTrdLoad. В процессе загрузки работает процесс воспроизведения торгов. Если в файл записаны также и сделки, то производится верификация — сравнение сгенерированной последовательности сделок с исходной.

Функция возвращает код ошибки:
mtER_FILELOAD — файл не существует;
mtER_FILEFORMAT — файл имеет неверный формат;
mtER_VERIFY — при верификации обнаружены некорректные сделки.

`int mtSessionsLoadAll (mtSession* list,`
`int size, mtString filename, mtTrdFormat*)`

Загрузка не более size сессий из текстового файла-протокола формата TRD с именем filename. Все size сессий из списка сессий list должны быть созданы заранее. Впоследствии их необходимо будет удалить функцией mtSessionDelete. Возвращает число реально загруженных сессий. Если это число меньше size, то остальные сессии можно удалить. По окончании загрузки сессии упорядочиваются по числу сделок. Порядок полей в записях файла задаётся структурой mtTrdFormat. Эта структура должна быть инициализирована функцией mtFormatTrdInit или загружена из ini-файла функцией mtFormatTrdLoad. В процессе загрузки работает процесс воспроизведения торгов. Если в файл записаны также и сделки, то производится верификация — сравнение сгенерированной последовательности сделок с исходной.

Функция возвращает код ошибки:
mtER_FILELOAD — файл не существует;
mtER_FILEFORMAT — файл имеет неверный формат;
mtER_VERIFY — при верификации обнаружены некорректные сделки.

`int mtSessionSave (mtSession, mtString filename, mtTrdFormat*)`

Функция сохраняет исходную последовательность событий сессии в текстовом файле filename формата TRD. Порядок полей в записях файла задаётся структурой mtTrdFormat. Сделки и другие вычисляемые данные не сохраняются.

Функция возвращает код ошибки:
mtER_FILESAVE — невозможно создать выходной файл;
mtER_FILEFORMAT — файл имеет неверный формат;
mtER_NOEVENTS — сессия не содержит исходных событий.

`int mtSessionReplay (mtSession)`

Функция воспроизводит сессию, используя имеющуюся в ней последовательность событий. Вычисляются все таблицы данных.

Функция возвращает код ошибки mtER_NOEVENTS, если сессия не содержит исходную последовательность событий.

`int mtSessionSetSecurity (mtSession, mtSecurity*)`

Функция устанавливает свойства торгуемого финансового инструмента для данной сессии: название, шаг цены, время начала и конца торгов. Структура mtSecurity должна быть предварительно корректно заполнена.

Возвращает код ошибки mtER_SECURIY при обнаружении некорректных данных в структуре mtSecurity.

`int mtSessionGetSecurity (mtSession, mtSecurity*)`

Функция записывает свойства финансового инструмента в структуру mtSecurity.

Возвращает код ошибки mtER_SECURIY при обнаружении некорректных данных в структуре mtSecurity.

`int mtSessionGetStatistics (mtSession, mtStatistics*)`

Функция записывает статистику сессии в структуру mtStatistics. Статистика вычисляется во время работы торгового автомата. В частности статистика становится доступной после корректной отработки функций mtSessionLoad, mtSessionReplay, mtModelTune и mtModelPlay.

`int mtSessionGetHistorySize (mtSession)`

Функция возвращает длину массива предыстории сессии.

`int mtSessionGetHistory (mtSession, long i, mtShortStat *hist)`

Функция переписывает i-ую запись из массива предыстории сессии в структуру mtShortStat. Размер массива выдаёт функция mtSessionGetHistorySize. Предыстория должна быть загружена из исходного файла протокола.

Возвращает код ошибки mtER_HIST, если i выходит за размер массива.

`int mtSessionGetHistoryTotal (mtSession, mtShortStat *hist)`

Функция записывает итоговые данные по всей предыстории сессии в структуру mtShortStat. Предыстория должна быть загружена из исходного файла протокола.

Возвращает код ошибки mtER_HIST, если i выходит за границы массива.

int mtSessionSetMovingAverage (mtSession,
mtTime length, double accuracy)

Функция создаёт скользящую среднюю цены с периодом length секунд. С целью экономии ресурсов функция может вернуть ранее созданную скользящую среднюю, если её период отличается от length не более чем на accuracy процентов. Скользящая средняя вычисляется во время воспроизведения торгов. Узнать текущее значение скользящей средней можно с помощью функции mtSessionGetMovingAverage.

Возвращает номер скользящей средней, который в дальнейшем может быть передан функции mtSessionGetMovingAverage.

int mtSessionGetMovingAverage (mtSession,
long movno, mtMovingAverage*)

Функция записывает скользящую среднюю с порядковым номером movno в структуру mtMovingAverage.

Возвращает код ошибки mtER_MOVAVR, если скользящей средней с таким номером не существует.

int mtSessionSetSpread (mtSession, mtQuantity qLimit, mtBool isBid)

Функция создаёт протокол спрэда с объёмом qLimit. Протокол спрэда представляет собой кусочно-постоянную функцию цены от времени и хранится в виде массива записей, имеющих тип mtConstancyInterval:

```
struct mtConstancyInterval {
   mtTime Time;            // время начала участка постоянства
   double Value;           // значение функции на участке постоянства
};
```

Каждая запись содержит время и цену. Цена определяется следующим условием. При isBid=1 заявки на покупку суммарным объёмом qLimit стоят в очереди не ниже этой цены. При isBid=0 заявки на продажу суммарным объёмом qLimit стоят в очереди не выше этой цены.

Протокол спрэда наращивается в ходе воспроизведения сессии. Новая запись добавляется всякий раз, когда одновременно меняется и цена, и время. Если изменяется только цена, но не время, то вместо создания новой записи модифицируется последняя запись протокола. Тем самым гарантируется, что последовательные интервалы имеют строго возрастающие значения поля Time.

Функция возвращает идентификатор спрэда, используемый в функциях mtSessionSpreadSize, mtSessionGetSpread.

int mtSessionSpreadSize (mtSession, long spreadno)

Функция возвращает число записей в спрэде с идентификатором spreadno.

Возвращает mtER_SPREADNO, если протокол спрэда с указанным идентификатором не существует.

int mtSessionGetSpread (mtSession,
long spreadno, long i, mtConstancyInterval*)

Функция записывает i-ую запись спрэда spreadno в структуру mtConstancyInterval.

Возвращает mtER_SPREADNO, если протокол спрэда с указанным идентификатором не существует, или число записей в нём равно нулю, или номер записи i находится за пределами массива.

int mtSessionParts (mtSession)

Функция выдаёт число участников сессии. Участники создаются неявно при загрузке сессии mtSessionLoad и при подаче заявок mtEngineOrderEnter. Участника можно также создать явно с помощью функции mtSessionAddPart.

int mtSessionPartID (mtSession, long part, mtString*)

Функция записывает в mtString указатель на идентификатор part-го участника сессии. Участники нумеруются от нуля до mtSessionParts-1.

Возвращает код ошибки mtER_PART, если нет участника с таким номером.

int mtSessionFindPart (mtSession, mtString)

Функция ищет в сессии участника с идентификатором mtString.

Возвращает номер участника или код ошибки mtER_PART, если участник не найден.

int mtSessionAddPart (mtSession, mtString)

Функция создаёт нового участника с заданным именем и возвращает его номер. Новый участник создаётся всегда, даже если другой участник с таким же именем уже существует. В этом случае дальнейшие операции с участником (например, подача заявок) должны производиться по номеру, а не по имени.

int mtSessionSetPartInitial (mtSession,
long partno, const mtPartInitial*)

Функция задаёт входящие позиции partno-го участника. Функция может вызываться как до начала торгов, так и во время торгов. Булевское поле CheckLimits в структуре mtPartInitial позволяет установить, следует ли для данного участника контролировать лимиты по позициям при постановке заявок.

Возвращает код ошибки mtER_PART, если участник не найден.

int mtSessionGetPartDossier (session, long partno, mtPartDossier*)

Функция записывает досье partno-го участника в структуру mtPartDossier.

Возвращает код ошибки mtER_PART, если участник не найден.

```
int mtSessionSetEventFunc (mtSession,
      mtEventFunc on_before,
      mtEventFunc on_after,
      mtEventTypeFunc on_control);
```

Функция устанавливает внешние обработчики торговых событий (ввода заявки, снятия заявки или сделки).

Функция предобработки on_before вызывается непосредственно перед каждым торговым событием.

Функция постобработки on_after вызывается непосредственно после каждого торгового события.

В качестве аргумента функциям on_before и on_after передаётся ссылка на запись mtEvent о данном событии в протоколе сессии:

```
typedef int (*mtEventFunc) (mtEvent &event);
```

Функция обработки управляющего события on_control вызывается при появлении управляющего события. В качестве аргумента ей передаётся тип события mtEventType:

```
typedef int (*mtEventTypeFunc) (mtEventType &etype);
```

Управляющие события кодируются константами типа mtEventType.

Торговый автомат

Функции торгового автомата возвращают коды ошибок:

mtER_SESSION — первый аргумент mtSession не существует или не является сессией;

mtER_ENGINE — торговый автомат не был инициализирован для данной сессии функцией mtEngineStart.

`int mtEngineStart (mtSession)`

Функция переводит торговый автомат в исходное состояние для воспроизведения сессии. Старая последовательность событий стирается, очереди и таблицы данных очищаются. Новая последовательность заявок будет формироваться с помощью функций mtEngineOrderEnter и mtEngineOrderWithdraw.

`int mtEngineStop (mtSession)`

Функция останавливает формирование сессии. После этого все функции торгового автомата, кроме mtEngineStart, будут возвращать код ошибки mtER_ENGINE.

`int mtEngineOrderEnter (mtSession, mtOrder*)`

Функция подаёт заявку. В структуре mtOrder* необходимо заполнить поля Time, Price, Volume, PartID или PartNo, Mask, необязательно OrderNo. Если заявка рыночная, то она должна иметь флаг mtET_MARKET и/или нулевую цену.

В таблицы событий добавляется строка, соответствующая поданной заявке.

Участник может быть задан либо символьным идентификаторов PartID (тогда PartNo должен быть отрицательным), либо порядковым номером PartNo в слова-

ре участников сессии (тогда PartID должен быть нулевым). В первом случае новый участник автоматически добавляется в словарь.

Функция возвращает коды ошибок:
mtER_ORDER, если в заявке неправильно указана цена или объём;
mtER_PART, если в заявке неправильно задан участник;
mtER_NOCASH, если участнику не хватает денежной позиции;
mtER_NODEPO, если участнику не хватает депозитарной позиции;
mtER_NOTRADING, если торги уже закончились или ещё не начались.

`int mtEngineOrderWithdraw (mtSession, mtOrder*, mtTime now)`

Функция снимает заявку, на которую указывает mtOrder. Эта заявка должна стоять в очереди, то есть должна быть получена с помощью функции mtEngineGetOrderBook или mtEngineGetOrderPart. Аргумент now должен содержать время снятия заявки. В таблицы событий добавляется строка, соответствующая снятию заявки.

Функция возвращает код ошибки:
mtER_ORDER, если заявка не стоит в очереди.

`int mtEngineGetOrderBook (mtSession,`
`mtOrder** bestbid, mtOrder** bestask)`

Функция записывает указатели на текущие лучшие бид и аск в переменные *bestbid и *bestask соответственно. Если очередь пуста, записывается 0. С помощью этих указателей, следуя по ссылкам Next и Prev, можно получить весь список заявок, стоящих в очередях на покупку и продажу.

`int mtEngineGetOrderPart (mtSession,`
`long partno, mtOrder** bid, mtOrder** ask)`

Функция записывает в переменные *bid и *ask указатели на последние поданные заявки участника с номером partno. Порядковый номер участника можно узнать по его символьному идентификатору с помощью функции mtSessionFindPart. Если в очереди нет заявок данного участника, в соответствующую переменную bid и/или ask записывается 0. Полный список заявок данного участника, стоящих в очереди, можно получить, следуя по указателям PartNext и PartPrev.

Функция возвращает код ошибки mtER_PART, если участник с данным номером не существует в сессии.

`int mtEngineGetEvent (mtSession, int eventno, mtEvent*)`

Функция копирует запись с порядковым номером eventno из протокола событий в структуру mtEvent.

Возвращает код ошибки mtER_EVENT, если события с номером eventno нет в протоколе.

`int mtEngineGetEventsCount (mtSession session)`

Функция выдаёт число записей в протоколе событий.

Модель торгов

Модель торгов mtModel представляет собой торговую сессию, обладающую дополнительной возможностью автоматической генерации потока событий. Объект типа mtModel можно передавать функциям библиотеки везде, где требуется объект типа mtSession.

Чтобы модель могла генерировать поток событий, её необходимо предварительно настроить функцией mtModelTune на заданной исходной торговой сессии.

Модельные торги можно генерировать в любом заданном темпе и начиная с любой заданной точки исходной сессии. При этом в модели возникают две разных шкалы времени: *модельное время*, отмеряемое по событиям исходной сессии, и реальное *астрономическое время*, отмеряемое по системным часам. Функция mtModelSynchronize позволяет синхронизировать обе шкалы времени и задать ускорение или замедление модельного потока событий по отношению к исходной сессии.

В процессе генерации модельных торгов все операции со временем производятся в модельном времени. Это означает, что в подаваемых заявках, стоящих в очереди заявках, и в событиях, заносимых в протокол, всегда записывается модельное время, независимо от показаний системных часов. Функция mtModelGetTime выдаёт текущее время также в шкале модельного времени. Для перевода модельного времени в астрономическое и обратно используются функции преобразования mtModelProtoToAstroTime и mtModelAstroToProtoTime соответственно.

Модельные торги с одним и тем же набором участников можно имитировать любое число раз. Каждая последующая имитация затирает данные, полученные в ходе предыдущей.

Функции, принимающие объект mtModel в качестве первого аргумента, возвращают код ошибки mtER_MODEL, если объект mtModel не существует или не является моделью торгов;

`int mtModelCreate (mtModel*)`

Функция создаёт модель торгов и записывает в mtModel указатель на созданную модель. В случае успешного создания объект mtModel можно передавать другим функциям для дальнейшей работы с моделью. Созданная модель должна быть впоследствии удалена с помощью функции mtModelDelete.

Возвращает код ошибки mtER_MODEL, если создать модель не удалось.

`int mtModelDelete (mtModel)`

Функция удаляет модель торгов, созданную mtModelCreate.

`int mtModelSynchronize (mtModel, mtTime, double delay, mtVoidFunc)`

Функция синхронизирует заданный момент времени исходной сессии mtTime с текущим системным (астрономическим) временем. Функцию можно вызвать как до начала торгов, так и во время торгов.

Время mtTime отмечает момент начала синхронизации (в секундах с полуночи) в исходной сессии. До этого момента торги будут генерироваться с максимальной скоростью.

Коэффициент задержки delay указывает, во сколько раз надо замедлить течение времени после начала синхронизации. Если delay = 0, торги генерируются с максимальной скоростью. Если delay < 1, торги идут в ускоренном темпе. Если delay = 1, торги идут с естественная скорость. Если delay > 1, торги идут в замедленном темпе.

Функция mtVoidFunc задаёт обработчик холостых ожиданий в ходе модельных торгов. Этот обработчик вызывается в цикле ожидания следующего события и только в том случае, если задан режим синхронизации с задержками (delay > 0).

`int mtModelGetTime (mtModel, mtTime*)`

Функция записывает текущее модельное время в mtTime. Если торги ещё не начались или уже остановлены, возвращается текущее системное время.

`mtTime mtModelProtoToAstroTime (mtModel, mtTime prototime)`

Функция возвращает результат преобразования протокольного времени prototime в астрономическое. Если синхронизация mtModelSynchronize не установлена, функция возвращает исходное значение prototime.

`mtTime mtModelAstroToProtoTime (mtModel, mtTime astrotime)`

Функция возвращает результат преобразования астрономического времени astrotime в протокольное. Если синхронизация mtModelSynchronize не установлена, функция возвращает исходное значение astrotime.

`int mtModelSave (mtModel, mtString filename)`

Функция сохраняет модель торгов в файле формата IMT с заданным именем. Сохраняются все параметры каждого модельного участника.

Возвращает код ошибки mtER_FILESAVE, если невозможно создать файл.

`int mtModelLoad (mtModel, mtString filename)`

Функция загружает модель из ранее сохранённого файла формата IMT.

Для успешной загрузки необходимо, чтобы все используемые в данном файле модели участников были заранее зарегистрированы в реестре моделей с помощью функции mtRegisterPartModel.

Возвращает код ошибки mtER_FILELOAD, если невозможно считать файл.

`int mtModelTune (mtModel model)`

Функция настраивает модель на воспроизведение записанного в ней протокола событий.

Возвращает коды ошибки:
mtER_MODELTUNE — нет протокола событий;
mtER_VERIFY — в процессе воспроизведения обнаружены ошибки в протоколе.

`int mtModelPlayStep (mtModel, int wait)`

Функция генерирует одно модельное событие и передаёт его торговому автомату. Генерацию модельных событий можно чередовать с подачей и снятием заявок, применяя функции mtEngineOrder{…} непосредственно к модели mtModel.

Если wait=1, функция ожидает наступления модельного события, реализуя задержку такой длительности, чтобы модельные торги шли в темпе, заданном параметром delay функции mtModelSynchronize.

Если wait=0, функция проверяет, пора ли сгенерировать очередное модельное событие. Если пора, событие немедленно генерируется и функция возвращает код mtER_OK. Если нет, функция возвращает код 3. Функция завершается сразу без ожидания модельного события.

Функция может также возвращать следующие значения, которые не являются признаком ошибочного завершения:

- 1 — все модельные участники отказались от дальнейшего участия в игре;
- 2 — торговая сессия завершена согласно регламенту;
- 3 — wait=0 и время очередного модельного события ещё не наступило.

`int mtModelPlay (mtModel)`

Функция генерирует модельные торги, имитируя полный торговый день. Функция эквивалентна следующей последовательности вызовов:

```
mtModel Model;
mtEngineStart (Model);
while (mtModelPlayStep(Model,1) == mtER_OK);
mtEngineStop (Model);
```

Возвращает результат последнего обращения к mtModelPlayStep.

Модель участника торгов

Для доступа к отдельным участникам торгов используются их порядковые номера в списке участников модели. Для получения порядкового номера по символьному идентификатору следует воспользоваться функцией mtSessionFindPart. Обратная операция — получение символьного идентификатора по порядковому номеру — осуществляется функцией mtSessionPartID. Функция mtSessionParts возвращает число участников в сессии. Участники могут создаваться явным образом с помощью функции mtSessionAddPart.

`int mtModelPartCreate (mtModel, long partno, mtString model_name)`

Функция задаёт тип модели для partno-го участника. Имя модели model_name должно быть предварительно зарегистрировано в реестре моделей.

Возвращает коды ошибки:
mtER_NOREGISTER — модель с указанным именем не зарегистрирована;
mtER_PART — в торговой сессии нет участника с номером partno.

`int mtModelPartCreate (mtModel, long partno, mtString model_name)`

Функция задаёт тип модели для partno-го участника. Имя модели model_name должно быть предварительно зарегистрировано в реестре моделей.

Возвращает коды ошибки:
mtER_NOREGISTER — модель с указанным именем не зарегистрирована;
mtER_PART — в торговой сессии нет участника с номером partno.

`int mtModelPartSetParam (mtModel, long partno, mtString, double)`

Функция задаёт значение double параметра с именем mtString для partno-го участника в модели торгов mtModel. Участники нумеруются, начиная с 0.

У всех модельных участников независимо от типа модели есть параметр “enable”, который принимает два значения: 1 если участник включён, 0 если участник выключен из игры. По умолчанию параметр равен 1.

Возвращает код ошибки:
mtER_PART — нет участника с таким номером;
mtER_MODELPART — участник не имеет модели;
mtER_PARAM — параметр не существует в модели данного типа.

`int mtModelPartGetParam (mtModel, long partno, mtString, double*)`

Функция записывает значение параметра с именем mtString для partno-го участника в модели торгов mtModel в переменную с адресом double. Участники нумеруются, начиная с 0.

Возвращает код ошибки:
mtER_PART — нет участника с таким номером или идентификатором;
mtER_MODELPART — участник не имеет модели;
mtER_PARAM — параметр не существует в модели данного типа.

Модель рынка

Модель рынка представляет собой совокупность модельных торговых сессий по различным финансовым инструментам за один и тот же торговый день. Модель рынка можно целиком загружать из файлов формата TRD или IMT, а также записывать в файлы указанных форматов.

Формат TRD содержит в текстовом виде исходный протокол торгов за один торговый день по одному или нескольким финансовым инструментам. Формат IMT содержит в закодированном бинарном виде полное описание модели рынка.

Все функции, принимающие объект mtMarket в качестве первого аргумента, возвращают код ошибки mtER_MARKET, если объект не существует или не является моделью рынка.

`int mtMarketCreate (mtMarket*)`

Функция создаёт модель рынка и записывает в mtMarket указатель на созданную модель. В случае успешного создания объект mtMarket можно передавать другим функциям для дальнейшей работы с моделью. Созданная модель должна быть впоследствии удалена с помощью функции mtMarketDelete.

Возвращает код ошибки mtER_MARKET, если создать модель рынка не удалось.

`int mtMarketDelete (mtMarket)`

Функция удаляет модель рынка, созданную mtMarketCreate.

`int mtMarketLoadTRD (mtMarket, mtString, mtTrdFormat*)`

Функция загружает сессии из текстового файла-протокола формата TRD с именем mtString, содержащего последовательность событий одного торгового дня. Порядок полей в записях файла задаётся структурой mtTrdFormat. Эта структура должна быть инициализирована функцией mtFormatTrdInit или загружена из ini-файла функцией mtFormatTrdLoad.

Функция создаёт модели торгов для всех финансовых инструментов, по которым в протоколе имеется хотя бы одно событие. В процессе загрузки для каждого инструмента включается отдельный процесс воспроизведения торгов. Если в файл записаны также и сделки, то производится верификация — сравнение сгенерированной последовательности сделок с исходной.

Функция возвращает код ошибки:
mtER_FILELOAD — файл не существует;
mtER_FILEFORMAT — файл имеет неверный формат;
mtER_VERIFY — при верификации обнаружены некорректные сделки.

`int mtMarketSaveTRD (mtMarket, mtString, mtTrdFormat*)`

Функция сохраняет протокол торгов по всем финансовым инструментам рынка в текстовом файле filename формата TRD. При записи события упорядочиваются сначала по инструментам, затем для каждого инструмента — по времени. Порядок полей в записях файла задаётся структурой mtTrdFormat. Сделки и другие вычисляемые данные не сохраняются.

Функция возвращает код ошибки:
mtER_FILESAVE — невозможно создать выходной файл;
mtER_FILEFORMAT — файл имеет неверный формат;
mtER_NOEVENTS — сессия не содержит исходных событий.

`int mtMarketLoadIMT (mtMarket, mtString)`

Функция загружает настроенную модель рынка из ранее сохранённого файла формата IMT с именем mtString. При этом автоматически создаются все торговые сессии.

Возвращает код ошибки mtER_FILELOAD, если невозможно считать файл.

`int mtMarketSaveIMT (mtMarket, mtString, long encrypt_names)`

Функция сохраняет модель рынка в файле формата IMT с именем mtString. Сохраняются все параметры всех модельных участников во всех торговых сессиях. Если параметр encrypt_names имеет ненулевое значение, то имена участников будут шифроваться при записи в файл (при последующей загрузке файла функцией mtMarketLoadIMT участники получат зашифрованные имена, а настоящие имена будут потеряны навсегда).

Возвращает код ошибки mtER_FILESAVE, если невозможно создать файл.

`int mtMarketModels (mtMarket)`

Функция возвращает количество отдельных торговых сессий (финансовых инструментов) в модели рынка.

`int mtMarketGetModel (mtMarket, long modelno, mtModel*)`

Функция возвращает в mtModel указатель на модель торговой сессии по отдельному финансовому инструменту. Порядковый номер сессии (финансового инструмента) для данного рынка задается числом modelno. Сессии нумеруются, начиная с нуля. Максимальный номер сессии равен mtMarketModels–1.

Полученную сессию mtModel не следует удалять функцией mtModelDelete.

Обработка ошибок

`int mtErrorSetLog (mtString filename)`

Функция включает вывод отладочной информации в протокол — текстовый файл с заданным именем. По умолчанию отладочная информация не выводится.

Возвращает код ошибки mtER_FILESAVE, если невозможно создать файл.

`int mtErrorLog (mtString message)`

Функция выводит сообщение message в текущий протокол.

Вспомогательные функции

`int mtFormatTrdInit (mtTrdFormat* format)`

Функция устанавливает стандартный формат записей в trd-файле протокола сессии. Информация о формате заносится в структуру format.

Файл протокола сессии представляет собой текстовый файл, в котором каждая строка соответствует одному событию. Тип события записывается прописной буквой в нулевой позиции строки. Разделителем полей служит символ табуляции.

Стандартный формат записей в файле протокола:

0	1	2	3	4	5	6	7	8	9	10
D	Date	Remark								
I	Security	FullName								
H	Security	Time	Period	Open	Close	Min	Max	Value	Qty	Count
B S	Security	Time	OrderID	Name	Price	Qty	Flags			
W	Security	Time	OrderID	Name*	Price*	Qty*				
T	Security	Time	AskID	BidID	Seller*	Buyer*	Price	Qty		

Звёздочкой отмечены необязательные поля, которые, в случае их наличия, могут использоваться при верификации торгового автомата.

Необязательная информационная строка D записывается в файл один раз перед началом сессии и содержит информацию о генераторе файла-протокола.

Необязательная информационная строка I может содержать следующую информацию о финансовом инструменте:

Security	идентификатор финансового инструмента;
FullName	полное название инструмента;
PriceTick	величина тика цены;

Decimals	число десятичных знаков в цене;
Tax	размер комиссионных в процентах;
OpenBegin	начало периода открытия;
OpenEnd	конец периода открытия;
NormBegin	начало нормального периода;
NormEnd	конец нормального периода;
CloseBegin	начало периода закрытия;
CloseEnd	конец периода закрытия.

Необязательная информационная строка H содержит информацию об интервале времени в предыстории торгов по данному финансовому инструменту. Предполагается, что каждый интервал продолжается до начала следующего. Длительность каждого интервала и количество интервалов могут быть произвольными. Однако для построения графика предыстории длительности интервалов рекомендуется делать одинаковыми.

Security	идентификатор финансового инструмента;
Time	время начала интервала;
Period	длительность интервала (в секундах);
Open	цена первой сделки на интервале;
Close	цена последней сделки на интервале;
Min	минимальная цена на интервале;
Max	максимальная цена на интервале;
Value	объём торгов в рублях на интервале;
Qty	объём торгов в штуках инструмента на интервале;
Count	число сделок на интервале.

Строки событий B и S содержат информацию о подаче заявки на покупку и продажу соответственно.

Security	идентификатор финансового инструмента;
Time	время подачи заявки;
OrderID	идентификационный номер заявки в торговой системе;
Name	имя участника;
Price	цена заявки;
Qty	объём заявки;
QtyHidden	скрытый объём заявки;
Flags	строка статусов заявки.

Поле заявки Flags представляет собой строку, которая может содержать следующие символы:

M	рыночная заявка
F	снять-остаток
N	немедленно или отклонить
O	период открытия
C	период закрытия
A	аукцион

Строка события W содержит информацию о снятии заявки. Необязательные поля, если присутствуют, используются только с целью верификации.

Security	идентификатор финансового инструмента;
Time	время снятия заявки;

OrderID	идентификационный номер снимаемой заявки;
Name	имя участника (необязательно);
Price	цена заявки (необязательно);
Qty	объём заявки (необязательно).

Строка события T содержит информацию о сделке. Строки сделок необязательны и используются только для верификации торгового автомата.

Security	идентификатор финансового инструмента;
Time	время сделки;
TradeID	идентификационный номер сделки;
AskID	идентификационный номер OrderID заявки продавца;
BidID	идентификационный номер OrderID заявки покупателя;
Price	цена сделки;
Qty	объём сделки;
Seller	имя Name продавца (необязательно);
Buyer	имя Name покупателя (необязательно).

`int mtFormatTrdLoad (mtString filename, mtTrdFormat* format)`

Загрузить спецификацию trd-файла из ini-файла filename в структуру format.

`mtTime mtStrToTime (mtString str)`

Преобразовать строку во время.

`mtString mtTimeToStr (mtTime t)`

Преобразовать время в строку. Полученную строку рекомендуется скопировать, так как функция каждый раз использует один и тот же буфер, и повторный вызов затирает предыдущий результат.

`mtEventType mtStrToMask (mtString str)`

Преобразовать строковое представление типов и статусов события в битовую маску mtEventType.

`mtString mtMaskToStr (mtEventType mask)`

Преобразовать битовую маску mask в строковое представление типов и статусов события. Возвращаемый указатель на строку не следует освобождать. Полученную строку рекомендуется скопировать, так как функция каждый раз использует один и тот же буфер, и повторный вызов затирает предыдущий результат.

Создание моделей участников торгов

Библиотека MoTor позволяет разрабатывать и включать в торговые сессии собственные модели участников торгов.

Модель участника представляет собой объект, выведенный путем наследования из интерфейсного (чисто виртуального) базового класса IPartModel. Реализуя этот объект, разработчик модели должен наделить его следующими функциональными возможностями:

- принятие торговых решений (функции PlayStart, PlayDecision, PlayStop);
- хранение *внутренних параметров* модели, необходимых для принятия торговых решений;
- настройка внутренних параметров модели на воспроизведение заданной последовательности торговых решений (функции TuneStart, TuneDecision, TuneStop);
- запись всех внутренних параметров модели в файл и считывание их из файла (функции Save и Load);
- предоставление пользователю доступа к *внешним параметрам* модели, которые не определяются в процессе настройки и должны устанавливаться извне, например, при создании модели участника (функция Param).

Модель торгов должна дважды пройти по торговой сессии. На первом проходе производится настройка (режим tune) всех модельных участников, на втором проходе настроенная модель имитирует биржевую игру (режим play).

Настроенную модель можно сохранить в IMT-файл и позже загрузить из IMT-файла с помощью функций mtMarketSaveIMT и mtMarketLoadIMT соответственно. Загруженная из IMT-файла модель уже не нуждается в настройке, её можно сразу запускать в режиме имитации биржевой игры.

Режим настройки запускается функцией mtModelTune. Сначала она инициализирует модели всех участников, вызывая для каждого функцию TuneStart. Затем торги воспроизводятся, и перед каждой подачей или снятием заявки вызывается функция TuneDecision для модельного участника, инициирующего данную торговую операцию. Режим настройки завершается вызовом функции TuneStop для всех модельных участников.

Режим имитации запускается функцией mtModelPlay или mtEngineStart. При этом для каждого модельного участника вызывается функция PlayStart, которая должна инициализировать модель участника и вернуть время первого принятия решения. Эти времена формируют очередь отложенных решений. Функционирование модели торгов заключается в том, что на каждом шаге из очереди вынимается самая ранняя запись, и для соответствующего модельного участника вызывается функция PlayDecision. Эта функция вычисляет и возвращает все параметры торгового решения. Если тип решения отличается от mtET_DO_NOTHING, то принятое решение передаётся механизму торгов. Иначе решение игнорируется. Но в обоих случаях функция PlayDecision должна сообщить время следующего принятия решения через поле NextTime записи mtDecision. Если это время равно нулю, то модельный участник выводится из игры. В противном случае время NextTime за-

носится в очередь отложенных решений. Таким образом, очередь отложенных решений постоянно обновляется в ходе имитации торгов. Длина очереди в каждый момент времени равна количеству активных модельных участников. Режим имитации завершается вызовом функции PlayStop для всех модельных участников.

Для создания модели участника разработчик должен выполнить следующие шаги:

- Реализовать модель участника в виде класса, поддерживающего интерфейс IPartModel. Поместить описание класса и его реализацию в отдельный файл с расширением cpp. Внутреннее устройство класса может быть каким угодно на усмотрение разработчика.
- Реализовать функцию порождения экземпляров класса. Поместить описание этой функции в заголовочный файл модели с расширением h.

Для включения модели участника в торговую сессию необходимо выполнить следующие шаги:

- Перед первым использованием модели зарегистрировать её класс в реестре моделей с помощью функции mtRegisterPartModel.
- Для выбранных участников задать тип модели с помощью функции mtModelPartCreate. При необходимости установить значения внешних параметров моделей с помощью функции mtModelPartSetParam.
- После этого модель участника готова к использованию. Можно настраивать модель торговой сессии (mtModelTune) и воспроизводить торги (mtModelPlay).
- По завершении работы необходимо очистить реестр моделей с помощью функции mtRegisterDelete.

Внимание! Если модель участника использовалась в настроенной модели торгов и была записана в IMT-файл, то к моменту загрузки IMT-файла данная модель уже должна быть зарегистрирована в реестре моделей. Иначе прочитать IMT-файл будет невозможно.

Интерфейс IPartModel

Интерфейс IPartModel представляет собой абстрактный базовый класс. Он описывает виртуальные функции, которые должна предоставлять модель участника, но сам этих функций не реализует. Разработчик модели участника должен вывести свой класс из IPartModel путем наследования и переопределить все его функции.

Несмотря на некоторое сходство, класс IPartModel не является интерфейсом COM. Наследник IPartModel может быть реализован только на языке C++.

Интерфейсный класс IPartModel:

```
class IPartModel {
public:
   virtual int Init (mtSession session, long partno);
   virtual int Free ();
   virtual mtString ModelName ();
   virtual double* Param (int id);
   // основные функции модели
```

```
   virtual int TuneDecision (const mtDecision& decision);
   virtual int PlayDecision (mtDecision& decision);
   // инициализация и завершение
   virtual int TuneStart ();
   virtual int TuneStop ();
   virtual int PlayStart (mtModelTime& first_decision_time);
   virtual int PlayStop ();
   // файловые операции
   virtual int Save (FILE* f);
   virtual int Load (FILE* f);
};
```

Далее перечисляются функции, которые разработчик модели должен реализовать в своём классе-наследнике интерфейса IPartModel.

int Init (mtSession session, long partno)

Функция инициализирует модель. Выполняется один раз в момент создания модели. Модель должна запомнить идентификатор торговой сессии session и номер моделируемого участника в этой сессии partno. Эта информация необходима для обращения к функциям библиотеки MoTor в процессе настройки модели и принятия торговых решений.

int Free ()

Функция удаляет модель. Она освобождает всю память, выделенную для хранения модели, после чего вызывает свой собственный деструктор:

```
delete this;
```

mtString ModelName ()

Функция возвращает имя модели. Под этим именем модель регистрируется в реестре моделей функцией mtRegisterPartModel. По этому имени создаются экземпляры модели функцией mtModelPartCreate.

double* Param (int id)

Функция возвращает указатель на ячейку, хранящую внешний параметр модели с заданным порядковым номером. Если параметра с указанным номером не существует, выдается нулевой указатель.

Номерам внешних параметров рекомендуется дать символьные идентификаторы с помощью макроопределений #define, либо с помощью константных определений, и вынести их в заголовочный файл модели.

Параметр с нулевым номером зарезервирован библиотекой MoTor:

```
const int MP_ENABLE = 0;
```

Если значение *Param(MP_ENABLE) больше нуля, то модельный участник включён в торги, иначе он выключен из торгов. Обработка этого параметра производится самой библиотекой MoTor, разработчику модели обрабатывать его не нужно.

`int TuneDecision (const mtDecision& decision)`

Функция принимает в качестве параметра описание очередного торгового решения данного участника в текущей сессии. Это решение может быть использовано моделью как угодно: она может сразу обновить накапливаемые статистики или запомнить это решение для дальнейшей обработки в функции TuneStop.

Функция может также использовать любую информацию о текущем состоянии сессии и участника торгов. С помощью указателя на сессию, переданного ранее через функцию Init, можно получить доступ к очередям, протоколу сессии, статистике сессии, досье участника, скользящим средневзвешенным и спрэдам.

Библиотека MoTor гарантирует поступление решений строго в той же последовательности, в которой соответствующий участник торгов производил свои торговые операции. Решениями считаются только подачи и снятия заявок. Сделки в функцию TuneDecision не передаются.

`int TuneStart ()`

Функция инициализирует модель участника перед началом настройки. Она вызывается для всех модельных участников перед началом воспроизведения сессии в режиме настройки (mtModelTune).

`int TuneStop ()`

Функция завершает настройку модели. Она вызывается моделью торгов для всех модельных участников по завершении воспроизведения сессии в режиме настройки (mtModelTune).

Для простых моделей, основанных на хранении протокола действий участника, функция TuneStop может ничего не делать, поскольку накопление протокола и его элементарную обработку можно целиком осуществить в функции TuneDecision. В более сложных моделях настройка может потребовать обработки всего протокола как единого целого. Тогда основная работа по настройке модели переносится в функцию TuneStop, а функция TuneDecision только собирает данные.

`int PlayDecision (mtDecision& decision)`

Функция генерирует торговое решение и записывает параметры принятого решения в запись decision. Эта запись содержит следующие поля:

- mtPrice Price — цена подаваемой или снимаемой заявки;
- mtQuantity Quantity — объем подаваемой или снимаемой заявки;
- mtEventType Mask — битовая маска, содержащая тип решения:

`mtET_DO_NOTHING`	ничего не делать;
`mtET_ORDER`	подать заявку;
`mtET_WITHDRAW`	снять заявку;
`mtET_BID`	тип подаваемой заявки — заявка на покупку, если не указано, то заявка на продажу;
`mtET_MARKET`	тип подаваемой заявки — рыночная заявка;
`mtET_FILL_AND_KILL`	тип подаваемой заявки — «снять остаток»;
`mtET_ALL_OR_NOTHING`	тип подаваемой заявки — «немедленно или отклонить».

- mtTime EnterTime — время подачи снимаемой заявки;
- mtModelTime NextTime — время следующего принятия решения;

Для подачи заявки необходимо заполнить поля Price, Quantity и установить флаг mtET_ORDER. Дополнительно могут быть установлены флаги типа подаваемой заявки. Если заявка рыночная (тип mtET_MARKET), то цена заявки не имеет значения и при попадании в торговый автомат устанавливается нулевой.

Для снятия заявки необходимо заполнить поля Price, Quantity и EnterTime. Эти параметры почти однозначно идентифицируют стоящую в очереди заявку. Наилучший способ принять решение о снятии — получить список своих заявок с помощью функции mtEngineGetOrderPart и выбрать в нём заявку, наиболее подходящую для снятия. Для этого можно использовать некоторый эвристический функционал, оценивающий целесообразность снятия. Затем у выбранной заявки считать указанные параметры и занести их в decision. Другой способ принять решение о снятии — непосредственно оценить цену и объем. В этом случае модель торгов попытается найти в очереди заявку с наиболее подходящими параметрами и снять её.

Функция PlayDecision может использовать любую информацию о текущем состоянии сессии и участника торгов. С помощью указателя на сессию, переданного ранее через функцию Init, можно получить доступ к очередям, протоколу сессии, статистике сессии, досье участника, скользящим средневзвешенным и спрэдам.

`int PlayStart (mtModelTime& first_decision_time)`

Функция инициализирует модель участника перед началом имитации. Она вызывается для всех модельных участников перед началом воспроизведения сессии в режиме имитации (mtModelPlay).

Функция обязана записать в аргумент first_decision_time время первого принятия торгового решения данным модельным участником. Если оно не будет записано, или если его значение окажется нулевым, участник будет исключен из игры.

`int PlayStop ()`

Функция завершает работу модели в режиме имитации. Она вызывается моделью торгов для всех модельных участников по завершении воспроизведения сессии в режиме имитации (mtModelPlay). Обычно эта функция ничего не делает.

`int Save (FILE* f)`

Функция записывает все внешние и внутренние параметры модели в файл f.

Функция возвращает код ошибки mtER_FILESAVE, если запись в файл невозможна.

`int Load (FILE* f)`

Функция считывает все внешние и внутренние параметры модели из файла f. Считывание данных должно производиться точно в той последовательности, в которой функция Save записывала эти данные.

Функция возвращает код ошибки mtER_FILELOAD, если чтение файла невозможно, или если в файле содержатся неверные данные.

Функция порождения экземпляров модели

Функция порождения создает экземпляр модели и возвращает указатель на созданный объект, приведенный к интерфейсному типу IPartModel. Функция может порождать модели только одного типа. Указатель на функцию порождения передаётся при регистрации модели в реестре моделей. Тип указателя на функцию порождения определен в файле mtPlugin.h следующим образом:

```
typedef IPartModel* (*mtCreatePartModelFunc) ()
```

Обычно функция порождения реализуется через оператор new. Допустим, что CMyPartModel — имя класса, выведенного из IPartModel. Тогда функция порождения могла бы быть реализована следующим образом:

```
IPartModel* CreateMyPartModel () {
   return new CMyPartModel;
}
```

Реестр моделей

Реестр моделей — это список доступных моделей участников торгов. Каждый элемент списка хранит имя модели и указатель на функцию порождения экземпляров модели.

Реестр используется при создании модели участника торгов с помощью функции mtModelPartCreate. Эта функция принимает на входе имя модели, находит модель в реестре, с помощью функции порождения создаёт экземпляр модели и приписывает его заданному участнику.

Реестр используется также при считывании модели торгов из IMT-файла. Для каждого модельного участника в файл заносится сначала имя модели, затем её внутреннее состояние с помощью функции IPartModel->Save. При загрузке модели участника из IMT-файла сначала считывается имя модели, затем по имени модели в реестре находится функция порождения, которая создает экземпляр модели. Для загрузки внутреннего состояния модели у порожденного экземпляра вызывается функция IPartModel->Load.

Реестр создается единожды перед первым использованием модели торгов (mtRegisterCreate) и удаляется также единожды по завершении работы с моделью (mtRegisterDelete). Модели добавляются в реестр с помощью функции mtRegisterPartModel. Если к моменту вызова mtRegisterPartModel реестр ещё не создан, то он создаётся автоматически.

Функции для работы с реестром mtRegisterCreate, mtRegisterDelete, mtRegisterPartModel являются частью библиотеки MoTor. Для их использования необходимо кроме заголовочного файла “mtLibrary.h” подключить также файл “mtPlugin.h”

Замечание. Реестр моделей является внутренним словарём библиотеки MoTor и не имеет никакого отношения к системному реестру Windows.

`int MT_INTERFACE mtRegisterCreate ()`

Функция создает реестр моделей и регистрирует все встроенные модели.

`int MT_INTERFACE mtRegisterDelete ()`

Функция удаляет реестр внешних моделей. Вызывать эту функцию необходимо для корректного освобождения памяти по окончании работы с моделями участников.

`int MT_INTERFACE mtRegisterPartModel (mtCreatePartModelFunc creator)`

Функция регистрирует модель в реестре. Для этого ей передается функция порождения экземпляров модели creator. После регистрации появляется возможность приписывать данную модель любому участнику в любой торговой сессии.

Чтобы узнать имя модели, функция порождает экземпляр модели с помощью creator, затем вызывает функцию порожденной модели ModelName и удаляет экземпляр. Заодно проверяется способность функции creator порождать работоспособные экземпляры модели.

Если имя регистрируемой модели совпадает с именем другой модели, зарегистрированной ранее, то функция замещает старый creator новым. Тем самым модель переопределяется, новая регистрационная запись не создается.

Функция возвращает код ошибки mtER_NOREGISTER, если не удается породить экземпляр модели или зарегистрировать модель в реестре.

Встроенные модели

`Voron.ProtoTest`

Простейшая модель, основанная на хранении протокола действий участника. Включает пересчет цен относительно противоположной очереди. Обладает нежелательным эффектом сжатия спрэда при возмущающем воздействии на модель.

`Voron.Protocol`

Базовая модель, основанная на хранении протокола действий участника. Содержит эвристические элементы, устраняющие эффект сжатия спрэда при возмущающем воздействии на модель.

Параметры модели:

`const int MP_ENABLE`	`1-модель включена, 0-выключена`
`const int MP_PROT_TYPE`	`способ вычисления базовой цены`
`const int MP_PROT_MOVLEN`	`длина скользящей средневзвешенной`

Значения параметра MP_PROT_TYPE:

`const int MC_PROTOCOL`	`без поправки цен`
`const int MC_PROTOCOL_CBO`	`лучшая цена противоположной очереди`
`const int MC_PROTOCOL_MPP`	`середина спрэда (bid+ask)/2`
`const int MC_PROTOCOL_MOV`	`скользящая средневзвешенная длины MP_PROT_MOVLEN сек.`
`const int MC_PROTOCOL_ORD`	`лучшая цена противоположной очереди для активных заявок, лучшая цена своей очереди - для пассивных и снятий`
`const int MC_PROTOCOL_LTP`	`цена последней сделки (Last Trade Price)`
`const int MC_PROTOCOL_SBO`	`лучшая цена своей очереди`

Перечень сообщений об ошибках

```
const int mtER_OK           =  0;    // Нет ошибки
const int mtER_SESSION      = -1001; // Сессия не инициализирована mtSessionCreate
const int mtER_CREATE       = -1002; // Невозможно создать сессию или модель
const int mtER_DELETE       = -1003; // Невозможно удалить занятый объект
const int mtER_SECURITY     = -1011; // Неверно задана информация о ценной бумаге
const int mtER_HIST         = -1014; // Невозможно получить данные из предыстории
const int mtER_STAT         = -1015; // Невозможно получить статистику
const int mtER_MOVAVR       = -1016; // Скользящая средняя не определена
const int mtER_SPREADNO     = -1017; // Спрэд не существует
const int mtER_FILESAVE     = -1021; // Невозможно сохранить данные в файле
const int mtER_FILELOAD     = -1022; // Невозможно открыть файл для чтения данных
const int mtER_FILEFORMAT   = -1023; // Невозможно считать данные - неверный формат
const int mtER_VERIFY       = -1024; // Обнаружены ошибки в торговом автомате
const int mtER_PART         = -1031; // Участник не найден по номеру или имени
const int mtER_NOEVENTS     = -1041; // Сессия не содержит исходных событий
const int mtER_ENGINE       = -1061; // Торги не запущены функцией mtEngineStart
const int mtER_ORDER        = -1062; // Заявка неверна, либо снятие не из очереди
const int mtER_EVENT        = -1063; // Событие не найдено в протоколе событий
const int mtER_NOCASH       = -1065; // Не достаточно денег для заявки на покупку
const int mtER_NODEPO       = -1066; // Не достаточно бумаг для заявки на продажу
const int mtER_NOTRADING    = -1067; // Торги уже закончились или ещё не начались
const int mtER_MODEL        = -1081; // Модель не инициализирована mtModelCreate
const int mtER_PARAM        = -1083; // Неверное имя параметра модельного участника
const int mtER_MODELTUNE    = -1084; // Нет событий для настройки модели
const int mtER_MODELPART    = -1086; // Участник не имеет модели
const int mtER_MARKET       = -1091; // Рынок не инициализирован mtMarketCreate
const int mtER_NOREGISTER   = -1101; // Не создан реестр внешних моделей участника
const int mtER_ERROR        = -1111; // Неизвестный код ошибки
```