MOEX МОСКОВСКАЯ БИРЖА

Клиентский программный интерфейс

Москва 2025

Содержание

1. Быстрое знакомство 4
1.1. Установка и подготовка к использованию 4
1.1.1. Состав и установка ПО в среде Linux 5
1.1.1.1. Установка из zip-архива 5
1.1.1.2. Установка из deb-пакета или rpm-пакета 6
1.2. Основные объекты 7
1.3. Общие сведения по работе с основными объектами Cgate 8
1.4. Работа с соединением 9
1.5. Получение потоков репликации 10
1.6. Работа со схемами данных 13
1.6.1. Политика использования схем данных 20
1.7. Отправка транзакций и получение ответов 22
1.8. Сценарий работы с p2sys 24
1.9. Объекты протокола изменения пароля 24
1.9.1. Публикатор p2mqpwd 24
1.9.2. Подписчик p2mqpwdreply 24
2. Описание API 26
2.1. Общие соглашения 26
2.2. Жизненный цикл объектов 26
2.3. Использование в многопоточном окружении 27
2.4. Запуск и остановка окружения 27
2.4.1. Настройка логов p2 28
2.5. Соединение 29
2.5.1. cg_conn_new 29
2.5.2. cg_conn_open 30
2.5.3. cg_conn_close 31
2.5.4. cg_conn_destroy 32
2.5.5. cg_conn_process 32
2.5.6. cg_conn_getstate 33
2.6. Подписчик 33
2.6.1. cg_lsn_new 34
2.6.2. cg_lsn_open 36
2.6.3. cg_lsn_close 38
2.6.4. cg_lsn_destroy 38
2.6.5. cg_lsn_getstate 39
2.6.6. cg_lsn_getscheme 39
2.7. Публикатор 40
2.7.1. cg_pub_new 40
2.7.2. cg_pub_open 41
2.7.3. cg_pub_close 41
2.7.4. cg_pub_destroy 42
2.7.5. cg_pub_getstate 42
2.7.6. cg_pub_getscheme 43
2.7.7. cg_pub_msgnew 43
2.7.8. cg_pub_post 45
2.7.9. cg_pub_msgfree 45
2.8. Логирование 46
2.8.1. cg_log_trace 46
2.8.2. cg_log_debug 46
2.8.3. cg_log_info 46
2.8.4. cg_log_error 47
2.8.5. cg_log_tracestr 47
2.8.6. cg_log_debugstr 47
2.8.7. cg_log_infostr 48
2.8.8. cg_log_errorstr 48
2.9. Объекты протокола p2sys 48
2.9.1. Соединение p2sys 48
2.9.2. Подписчик p2sys 49
2.9.3. Публикатор p2sys 49
2.10. Вспомогательные функции 49
2.10.1. cg_bcd_get 49
2.10.2. cg_getstr 50
2.10.3. cg_msg_dump 51
3. Описание инструментария 52
3.1. Утилита schemetool 52
3.1.1. makesrc - генерация структур 52
3.2. Утилита change_password 54
4. Описание API для Java, .NET 55
4.1. Описание 55

4.1.1. API CGate для Java 55
4.1.2. API CGate для .NET 55
4.2. Объект Cgate 56
4.3. Объект Connection 56
4.3.1. Конструктор Connection 56
4.3.2. Метод Connection.dispose 56
4.3.3. Метод Connection.open 56
4.3.4. Метод Connection.close 57
4.3.5. Метод Connection.process 57
4.3.6. Свойство Connection.state 57
4.4. Объект Listener 58
4.4.1. Конструктор Listener 58
4.4.2. Метод Listener.dispose 58
4.4.3. Метод Listener.open 59
4.4.4. Метод Listener.close 59
4.4.5. Свойство Listener.State 59
4.4.6. Свойство Listener.Scheme 59
4.4.7. Свойство Listener.Handler 60
4.5. Объект Publisher 60
4.5.1. Конструктор Publisher 61
4.5.2. Метод Publisher.dispose 61
4.5.3. Метод Publisher.open 61
4.5.4. Метод Publisher.close 61
4.5.5. Свойство Publisher.State 61
4.5.6. Свойство Publisher.Scheme 62
4.5.7. Метод Publisher.newMessage 62
4.5.8. Метод Publisher.post 63
4.6. Объект Message 63
4.6.1. Метод Message.dispose 63
4.6.2. Свойство Message.Type 63
4.6.3. Свойство Message.Data 64
4.6.4. Метод Message.toString 64
4.6.5. Типы сообщений 64
4.6.5.1. Объект OpenMessage 64
4.6.5.2. Объект CloseMessage 64
4.6.5.3. Объект DataMessage 64
4.6.5.4. Объект StreamDataMessage 65
4.6.5.5. Объект TnBeginMessage 65
4.6.5.6. Объект TnCommitMessage 65
4.6.5.7. Объект P2MQTimeoutMessage 65
4.6.5.8. Объект P2ReplLifeNumMessage 65
4.6.5.9. Объект P2ReplClearDeletedMessage 65
4.6.5.10. Объект P2ReplOnlineMessage 66
4.6.5.11. Объект P2ReplStateMessage 66

1. Быстрое знакомство

1.1. Установка и подготовка к использованию

Библиотека P2 CGate представляет собой набор следующих компонент:

- системные библиотеки Plaza-2
- маршрутизатор сообщений P2MQRouter
- шлюзовая библиотека cgate
- заголовочный файл с описанием API - cgate.h

Все эти компоненты необходимы для разработки с использованием библиотеки P2 CGate.

Для того, чтобы начать разработку, требуется установить компоненты инсталлятором, соответствующим используемой Вами операционной системе. В зависимости от операционной системы библиотеки и заголовочный файл будут установлены либо в стандартные предназначенные для этого места, либо в место, указанное при установке. В ходе дальнейших инструкций каталог установки будет обозначаться как C:\Moscow Exchange\SpectraCGate (В ОС семейства Windows при инсталляции будет создана системная переменная окружения с именем CGATE_HOME, значением которой является путь установки CGate).

Важно

Для работы с библиотекой необходимо наличие логина в систему Plaza-2 и ключа приложения. Для разработки используются логины в тестовую систему Plaza-2 и тестовые ключи, которые могут быть использованы всеми разработчиками произвольно. Для production используются production-логины и ключи. Production-ключи могут быть получены путём прохождения процедуры сертификации.

Для проверки корректности установки и готовности к разработке можно выполнить тестовую сборку примеров и их исполнение. Для этого надо выполнить следующие шаги:

1. Настройка маршрутизатора Plaza-2 в соответствии с имеющимся логином (в случае использования интерактивного инсталлятора данное действие выполняется автоматически)

   Необходимо открыть файл настройки роутера P2MQRouter, который, как правило, называется client_router.ini и в секции [AS:NS] заполнить логин и пароль:

   ```
   [AS:NS]
   USERNAME=<your login>
   PASSWORD=<your password>
   ```

2. Сборка примеров

   Примеры располагаются в каталоге C:\Moscow Exchange\SpectraCGate\SDK\samples для платформы Windows или в каталоге /usr/share/doc/cgate-examples (/opt/moex/cgate/samples) для Linux. Сборка примеров выполняется запуском сборочных скриптов, которые различаются в зависимости от используемой платформы и языка программирования. Для ОС Linux рекомендуется сделать копию примеров в своём домашнем каталоге и собирать их оттуда.

3. Исполнение примеров

   Для исполнения примеров необходимо убедиться, что роутер P2MQRouter запущен и соединен с сетью Plaza-2 (анализом сообщений роутера), в доступности библиотек Plaza-2 для запускаемого файла примера (возможно потребуется добавление каталога SpectraCGate\bin в переменную окружения PATH или указание каталога SpectraCGate\bin в Вашей среде разработки), а также в доступности конфигурационных файлов.

4. Описание примеров

   a. Пример aggrspy

      aggrspy - пример построения агрегированного стакана на покупку и продажу по фиксированному инструменту по потоку FORTS_AGGR50_REPL. При нажатии Enter в outfile выводится срез стакана.

      Команда для запуска:

      ```
      aggrspy ISIN_ID depth outfile [r]
      ```

      Входные аргументы:

      - isin_id - id инструмента;
      - depth - глубина выводимого стакана в файл (не больше 50);
      - outfile - файл для печати стакана;

- r - включить обратное направление сортировки (параметр используется для инструментов с обратной сортировкой).

b. Пример repl

repl - получение реплики данных по потоку. Пример печатает все получаемые сообщения в log. При разрыве соединения реплика начинается сначала. Входных аргументов нет.

c. Пример repl_resume

repl_resume - пример идентичен repl. Отличие заключается в том, что после разрыва соединения repl_resume продолжает реплику с последнего сообщения TN_COMMIT. Входных аргументов нет.

d. Пример send

send - выставляет заявку в SPECTRA. Выводит в лог ответ торговой системы. Входных аргументов нет.

e. Пример orderbook

orderbook - пример построения агрегированного стакана на покупку и продажу по фиксированному инструменту по online потоку FORTS_ORDLOG_REPL и снепшот потоку FORTS_USERORDERBOOK_REPL. При нажатии Enter в outfile выводится срез стакана.

Команда для запуска:

```
orderbook ISIN_ID depth outfile [r]
```

Входные аргументы:

- isin_id - id инструмента;
- depth - глубина выводимого стакана в файл (не больше 50);
- outfile - файл для печати стакана;
- r - включить обратное направление сортировки (параметр используется для инструментов с обратной сортировкой).

f. Пример p2sys

p2sys - пример авторизации роутера из cgate. Повторяет в цикле следующие действия:

i. Посылает ошибочный набор (login, pwd), в ответ получает сообщение logon failed;

ii. После этого посылает правильный набор (login, pwd);

iii. На сообщение об успешной авторизации посылается запрос на logout;

iv. Возврат к пункту 1.

g. Пример send_mt

send_mt - многопоточный пример посылки заявки. (Примечание: компилируется только под компиляторами, поддерживающими C++11.). В треде 1 посылаются заявки. В треде 2 обрабатываются reply на посылаемые заявки.

h. Пример instruments_state

instruments_state - пример, который демонстрирует самостоятельный расчет статусов инструментов по потоку FORTS_SECURITYGROUPSTATE_REPL (подробнее см. https://ftp.moex.com/pub/ClientsAPI/Spectra/CGate/prod/docs/p2gate_ru.pdf).

1.1.1. Состав и установка ПО в среде Linux

1.1.1.1. Установка из zip-архива

Дистрибутив Cgate состоит из инсталляционого скрипта и архива, в котором находятся загружаемые модули проекта cgate и проекта cgate_java, файлы include, файлы документации и файлы примеров. Дистрибутив доступен для скачивания по адресу: https://ftp.moex.com/pub/ClientsAPI/Spectra/CGate/.

Порядок установки:

1. Выполните команду (она добавит необходимые разрешения в атрибуты файла):

```
chmod 755 ./install.sh
```

2. Выполните команду:

```
./install.sh ./cgate_linux_amd64-7.12.0.103.zip
```

Примечание

Внимание! Имя архива привязано к версии ПО и может отличаться от имени, которое указано в примере выше.

3. В ответ на запрос: "Please, enter cgate install path:" укажите полный путь к каталогу, в который вы хотите распаковать cgate;
4. В ответ на запрос: "Please, enter P2 login:" укажите логин пользователя;
5. В ответ на запрос: "Please, enter P2 password:" укажите пароль пользователя.

Примечание

Внимание! Дальнейшие шаги установки различаются в зависимости от версии ОС Linux, установленной на компьютере.

Debian 6:

1. Установить пакет liblzo2-2(запуск роутера)
2. Установить пакет ant
3. Установить пакет openjdk-6-jdk (компиляция примеров java)
4. Установить пакет g++ (компиляция примеров C++).

CentOS 6:

1. Установить пакет lzo (запуск роутера)
2. Установить пакет gcc
3. Установить пакет gcc-c++ (компиляция примеров C++)
4. Установить пакет ant (компиляция примеров java).

Для компиляции примеров java необходимо выполнить команду:

```
make_java_samples.sh
```

Для запуска примеров java необходимо выполнить следующие действия:

1. Запустить роутер командой

   ```
   router.sh
   ```

2. Зайти в каталог examples/java
3. Запустить скрипт runjava.sh с необходимыми параметрами.

1.1.1.2. Установка из deb-пакета или rpm-пакета

Порядок установки:

1. Скачайте и установите пакет шлюза с помощью команды:

   ```
   dpkg -i cgate_<version>_amd64.deb
   ```

   в случае установки из deb-пакета, или команды:

   ```
   rpm -U cgate-<version>.x86_64.rpm
   ```

   в случае установки из rpm-пакета.

   <version> - номер версии дистрибутива

   Таблица раскладки компонентов:

Куда ставится	Описание
/opt/moex/cgate	Бинарники, схемы, документация по шлюзу
/etc/opt/moex/cgate	Настроечные файлы, auth.ini - файлы
/var/opt/moex/cgate	Каталог логирования, трейс файлы
/usr/share/doc	Copyright, документация по установке

2. Зайдите в каталог /etc/opt/moex/cgate/auth и в соответствующем ini-файле укажите логин/пароль для подключения:
   - prod.ini - для подключения к production системе

- t1.ini - для подключения к полигону T1
- t0.ini - для подключения к полигону T0
- game.ini - для подключения к игровому полигону

Примечание

Обратите внимание, что в случае переустановки пакета каталог \auth и файлы с настройками подключения не удаляются, поэтому повторно настраивать логин/пароль не потребуется.

3. Запустите сервис (роутер) с помощью команды:

```
systemctl start cgate@<profile>
```

<profile> - вариант подключения. В качестве профиля можно указать:

- prod - подключение к Spectra боевой контур
- rezerv - подключение к Spectra резервный контур
- t1 - подключение к полигону T1
- t0 - подключение к полигону T0
- game - подключение к игровому полигону
- rfs.prod - подключение к Spectra боевой контур с дополнительным доступом к RFS
- rfs.rezerv - подключение к Spectra резервный контур с дополнительным доступом к RFS
- rfs.t1 - подключение к полигону T1 с дополнительным доступом к RFS
- rfs.t0 - подключение к полигону T0 с дополнительным доступом к RFS

Примечание

Обратите внимание, что при апгрейде пакета запущенный сервис останавливается, поэтому после обновления сервис нужно запустить заново.

Код примеров C++ находится в каталоге /opt/moex/cgate/samples/c. Для сборки нужно запустить скрипт:

```
/opt/moex/cgate/samples/c/build_c_samples.sh
```

Результаты сборки будут в текущем каталоге.

Код примеров Java находится в каталоге /opt/moex/cgate/samples/java/basic. Скрипт сборки:

```
/opt/moex/cgate/samples/java/basic/build_samples.sh
```

Результаты сборки будут в текущем каталоге.

Для запуска какого-либо примера нужно из каталога сборки вызвать команду:

```
/opt/moex/cgate/samples/java/basic/run.sh <sample> <args>
```

Например:

```
/opt/moex/cgate/samples/java/basic/run.sh run.repl -Dindex=0
```

Возможные параметры run.sh:

```
run.aggrSpy -DisinId= -Ddepth= -DfileName=
run.orderBook -DisinId= -Ddepth= -DfileName=
run.repl -Dindex=
run.repl_resume -Dindex=
run.p2sys -Dlogin= -Dpwd=
run.send
run.send_mt
run.instruments_state
```

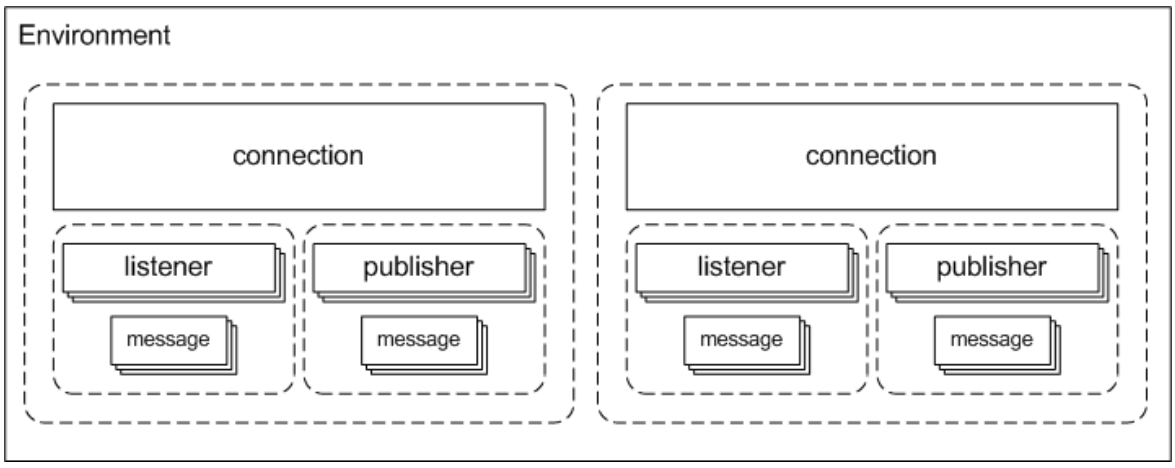
1.2. Основные объекты

Библиотека вводит набор объектов, с помощью которых осуществляется доступ к различным функциям системы. Основными объектами являются:

Окружение (Environment)	Описывает рабочее окружение библиотеки. Этот объект существует в единственном экземпляре. Его предназначение состоит в проведении инициализации и деинициализации подсистем, ведении журналов работы и управлении памятью.
Соединение (Connection)	Обеспечивает доступ к соединению с роутером Plaza-2
Сообщение (Message)	Описывает какое либо сообщение. В виде сообщений представляется вся информацию, отправляемая и получаемая пользователем - уведомления об обновлении данных, приказы в торговую систему, уведомления об исполнении приказов, уведомления об открытии и закрытии потоков данных.
Подписчик (Listener)	Предоставляет доступ к получению сообщений. Этот интерфейс используется для получения всех сообщений - обновлений потоков данных, ответов об исполнении приказов - если Вы получаете какое-либо сообщение, то Вы делаете это с помощью объекта Listener.
Публикатор (Publisher)	Предоставляет доступ к отправке сообщений. Всё, что Ваш код отправляет, он отправляет с помощью одного из объектов Publisher.

Объекты Listener и Publisher существуют в привязке к какому-либо соединению. Вы можете использовать много соединений, много подписчиков и публикаторов в зависимости от архитектуры Вашего приложения; как правило, соединения для получения обновлений рыночной информации отделяются от соединений, предназначенных для отправки приказов.

Общая схема объектов библиотеки в составе клиентского ПО выглядит следующим образом:



В общем окружении может существовать несколько соединений, каждое из которых содержит произвольное количество подписчиков и публикаторов, каждый из которых владеет некоторым количеством сообщений. При практическом использовании, как правило, назначение каждого соединения и соединенных с ним подписчиков и публикаторов продумывается исходя из фактических потребностей приложения.

1.3. Общие сведения по работе с основными объектами Cgate

Объекты Cgate: connection, listener, publisher в настройках url имеют параметр name, который отвечает за именование объекта в системе. Имя объекта должно быть уникальным. В случае попытки создать объект с занятым именем возвращается ошибка CG_ERR_INVALIDARGUMENT. Параметр name не обязателен для задания, по умолчанию Cgate присваивает объектам имена "noname_%d", где %d - некоторое число. Пользователям рекомендуется задавать параметр name для упрощения разборов логов Cgate.

В Cgate за время жизни объекта отвечает пользователь. Методы _new создают объекты. Методы _destroy разрушают объекты. Для избежания утечек памяти и системных ресурсов для каждого созданного объекта должен вызываться парный разрушающий метод.

В Cgate из callback lsn нельзя вызывать следующие функции:

- cg_lsn_destroy
- cg_conn_destroy
- cg_pub_destroy
- cg_pub_close
- cg_env_open
- cg_env_close

Вызов lsn_close, conn_close из callback lsn допускается после получения сообщения CG_MSG_OPEN.

Порядок открытия подписчиков и публикаторов важен, если в callback слушателя используется публикатор mq reply. Публикатор доложен быть открыт в первую очередь, т.к. в противном случае при открытии подписчика мы можем получить сообщение mq и не будем иметь активного публикатора на этот момент времени.

В разных версиях Cgate объекты при разрыве соединения с вышестоящим роутером ведут себя по-разному. Так, в версиях Cgate до 1.3.10 при разрыве одного из соединений с вышестоящими роутерами соединение переходило в состояние opening, при этом все объекты, связанные с этим соединением, закрывались. Уведомление о невалидности соединения приходило после определённого таймаута (3 минуты для сервисов репликации).

В Cgate версии 1.3.10 объекты ведут себя иначе. В Plaza 2 версии 202 и выше реализован механизм, когда сервисы (например, P2Proxy) уведомляют объекты Cgate о своём закрытии. При этом связанный с соединением объект закрывается немедленно, а объект listener закрывается через 3 репликационных таймаута (30 секунд в текущей реализации). Все остальные объекты остаются открытыми и продолжают работать.

1.4. Работа с соединением

Объект "Соединение" обеспечивает взаимодействие с роутером Plaza-2 для отправки и получения сообщений. Эти объекты могут создаваться в произвольном количестве в любое время работы ПО при инициализированном окружении; тем не менее, рекомендуется создавать соединения при старте ПО, а уничтожать - непосредственно перед выходом.

Создание соединения выполняется вызовом cg_conn_new, например так:

```
cg_conn_t* conn;
result = cg_conn_new("p2tcp://127.0.0.1:4001;app_name=test", &conn);
```

В этом примере создаётся соединение по протоколу TCP/IP с роутером Plaza-2 на порту 4001, запущенным на той же машине и именем приложения test. Вызов этой функции инициализирует объект соединения, но не приводит к фактической установке соединения.

Установка соединения производится посредством вызова функции conn_open:

```
result = cg_conn_open(conn, 0);
```

, где conn - объект, инициализированный вызовом функции cg_conn_new, а 0 в качестве второго параметра означает отсутствие параметров вызова открытия соединения.

Закрытие соединения выполняется вызовом conn_close:

```
result = cg_conn_close(conn);
```

При этом связь с роутером Plaza-2 разрывается, но объект остаётся инициализированным и может быть открыт повторно.

Уничтожение объекта выполняется с помощью функции conn_destroy:

```
result = cg_conn_destroy(conn);
```

Инициализация соединения может провалиться в случае нарушения целостности установки или неправильной конфигурации, например, были переданы некорректные параметры. В этом случае правильным действием будет остановка ПО и анализ конфигурации.

Открытие соединения может завершаться с ошибкой в силу разных причин, например, неготовности роутера Plaza-2 обслуживать входящие соединения, сбой в канале связи и прочее. Открытие соединения нужно выполнять циклично, так как следующая попытка открытия может оказаться удачной.

Пример описанного поведения:

```
cg_conn_t* conn;
result = cg_conn_new("p2tcp://127.0.0.1:4001;app_name=test", &conn);
if (result != CG_ERR_OK)
{
    // инициализация соединения провалилась
    // дальнейшая работа невозможна
    // сообщить об ошибке и выйти из программы
    return;
}

// в этом месте существует инициализированный объект conn,
// с которым можно работать - получать статус, открывать, закрывать

while (haveToExit()) // основной цикл программы
{
    uint32_t state;
    result = cg_conn_getstate(conn, &state); // получить статус соединения
    if (result != CG_ERR_OK) // произошла ошибка получения статуса соединения
    {
        // сообщить об ошибке и выйти из программы
        return;
    }
    switch (state)
```

```
    {
    case CG_STATE_CLOSED: // соединение закрыто, значит пробуем открыть
        result = cg_conn_open(conn, 0);
        // сообщить в случае ошибки
        break;
    case CG_STATE_ERROR: // соединение в состоянии ошибки, значит надо закрыть
        result = cg_conn_close(conn);
        // сообщить в случае ошибки
        break;
    case CG_STATE_ACTIVE: // соединение активно, с ним можно работать
        ...
    }
    ...
}
```

Подобный цикл реализует правильную работу с соединением: если соединение закрыто, то будет предпринята попытка его открыть; если соединение перешло в состояние ошибки, то выполняется его закрытие; работа с соединением производится в то время, когда оно активно.

В этом примере используется вызов функции cg_conn_getstate:

```
uint32_t state;
result = cg_conn_getstate(conn, &state);
```

Эта функция возвращает состояние инициализированного объекта "Соединение". Отправку и получение сообщений можно выполнять только в том случае, когда соответствующее соединение находится в состоянии "Активно" (CG_STATE_ACTIVE).

Соединение, находящееся в активном состоянии, нуждается в периодическом вызове функции обработки событий conn_process, в ходе которой выполняется вызов пользовательских функций обратного вызова, а также внутренняя обработка:

```
case CG_ACTIVE:
{
    result=cg_conn_process(conn, 0, 0);
    if (result != CG_ERR_OK && result != CG_ERR_TIMEOUT)
    {
        // работа соединения нарушена
        result = cg_conn_close(conn);
    }
    ...
    break;
}
```

Вызов conn_process принимает в качестве второго параметра интервал времени в миллисекундах, в течение которого происходит ожидание появление нового события внутри соединения. При этом, во время ожидания вызов conn_process блокируется. В случае, если в течение указанного времени не было произведено обработки ни одного сообщения, функция вернёт значение CG_ERR_TIMEOUT - это значение не является в данном случае индикатором ошибки и может быть использовано, например, для индикации того, что входящие сообщения отсутствуют и логика ПО может перейти к следующей задаче. Третий параметр зарезервирован.

Примечание! Если задать 0 в качестве значения второго параметра, то блокировка не происходит, при этом может достигаться 100% загрузка ядра CPU.

1.5. Получение потоков репликации

Получение потоков репликации выполняется с помощью объектов "Подписчик". Объект подписчик создаётся в привязке к соединению вызовом функции cg_lsn_new, например, так:

```
result = cg_lsn_new(conn, "p2repl://FORTS_TRADE_REPL", dataCB, user_data, &lsn);
```

В этом примере lsn инициализируется объектом "Подписчик", настроенным на получение потока репликации FORTS_TRADE_REPL через соединение conn. Сообщения об обновлениях данных, а также других событиях жизненного цикла потока будут приходить в заданную пользователем функцию обратного вызова dataCB. При создании подписки возможно задание различных параметров, в том числе клиентской схемы репликации; в этом случае инициализация объекта будет происходить так:

```
result = cg_lsn_new(conn,
               "p2repl://FORTS_TRADE_REPL;scheme=|FILE|ini/trades.ini|TRADES",
               dataCB, user_data, &lsn);
```

, где путь к файлу описания схемы и название секции соответствующего ini-файла задаются в параметре scheme строкой специального формата.

В случае успешного вызова функции cg_lsn_new объект находится в инициализированном, но не активном состоянии. Фактически открытие потока происходит посредством вызова функции cg_lsn_open:

```
result = cg_lsn_open(lsn, 0);
```

В этом примере поток репликации открыт без указания параметров, что означает, что он будет открыт с параметрами по умолчанию:

- номер жизни схемы данных не установлен (равен 0)
- ревизии всех таблиц равны 0, что означает их получение с нуля
- режим репликации выбран, как snapshot+online, что приводит к получению среза таблиц (или полной их истории), а затем переходу к получению данных в режиме он-лайн

Параметры задаются в виде строки:

```
result = cg_lsn_open(lsn, "mode=online");
```

При этом поток будет открыт в режиме online, что исключает фазу получения начального слепка данных. В режиме online, при разрывах соединения, не гарантируется непрерывность потока данных. Подробное описание возможных параметров см. в описании функции cg_lsn_open.

Функция cg_lsn_open может возвращать код ошибки в разных случаях: временная недоступность потока, нарушение работы канала. Для правильной работы следует обеспечить циклические попытки открытия потоков.

Поток закрывается вызовом функции cg_lsn_close:

```
result = cg_lsn_close(lsn);
```

При этом происходит отключение подписчика от получения данных и сообщения по обновлению данного потока прекращают идти через соединение; сам объект остаётся в инициализированном состоянии и может быть открыт повторно, в том числе и с другими параметрами.

Уничтожение объекта происходит посредством вызова cg_lsn_destroy:

```
result = cg_lsn_destroy(lsn);
```

После этого объект lsn освобождается и дальнейшая работа с ним невозможна.

Для корректного получения обновлений данных объектом "Подписчик" необходимо вызывать функцию conn_process для соединения, к которому привязан объект. Частота получения данных не превышает частоту вызовов conn_process, поэтому для максимальной скорости получения данных нужно обеспечить максимально возможную частоту вызова conn_process для интересующих соединений. При отсутствии вызова conn_process для соединения в течение timeout = 30 секундам происходит отключение подписчика. Минимальная рекомендованная частота вызова conn_process - не меньше одного раза в 10 секунд.

При получении данных, а также в моменты возникновения других событий в жизненном цикле потока репликации происходит вызов задаваемой пользователем в lsn_new функции обратного вызова, имеющей следующий вид:

```
typedef CG_RESULT (*CG_LISTENER_CB)(cg_conn_t* conn,
                                    cg_listener_t* listener,
                                    struct cg_msg_t* msg,
                                    void* data);
```

В функцию обратного вызова передаются:

- conn - соединение, к которому привязана подписка
- listener - объект "Подписчик"
- msg - пришедшее сообщение
- data - пользовательские данные, переданные в момент вызова функции lsn_new

Сообщение msg, которое приходит в пользовательскую функцию, в общем случае описывается следующей структурой:

```
struct cg_msg_t
{
    uint32_t type;     // Тип сообщения
    size_t data_size;  // Размер данных
    void* data;        // Указатель на данные
};
```

Любое сообщение, приходящее в пользовательскую функцию, гарантированно имеет указанные поля.

Идентификация конкретного вида сообщения выполняется с помощью анализа поля type. При получении потоков репликации используются следующие типы сообщений:

CG_MSG_OPEN

Сообщение приходит в момент активации потока данных. Это событие гарантированно возникает до прихода каких либо данных по данной подписке. Для потоков репликации приход сообщения означает, что схема данных согласована и готова для использования (Подробнее см. Схемы данных) Данное сообщение не содержит дополнительных данных и его поля data и data_size не используются.

	Примечание: Методы cg_pub_getscheme, cg_lsn_getscheme можно вызывать только после прихода сообщения CG_MSG_OPEN, до этого момента схема не определена.
CG_MSG_CLOSE	Сообщение приходит в момент закрытия потока данных. Приход сообщения означает, что поток был закрыт пользователем или системой. В поле data содержится указатель на int, по указанному адресу хранится информация о причине закрытия подписчика. Возможны следующие причины: • CG_REASON_UNDEFINED - не определена; • CG_REASON_USER - пользователь вернул ошибку в callback подписчика; • CG_REASON_ERROR - внутренняя ошибка; • CG_REASON_DONE - вызван метод cg_lsn_destroy; • CG_REASON_SNAPSHOT_DONE - снэпшот получен.
CG_MSG_TN_BEGIN	Означает момент начала получения очередного блока данных. В паре со следующим сообщением может быть использовано логикой ПО для контроля целостности данных. Данное сообщение не содержит дополнительных данных и его поля data и data_size не используются.
CG_MSG_TN_COMMIT	Означает момент завершения получения очередного блока данных. К моменту прихода этого сообщения можно считать, что данные полученные по данной подписке, находятся в непротиворечивом состоянии и отражают таблицы в синхронизированном между собой состоянии. Данное сообщение не содержит дополнительных данных и его поля data и data_size не используются.
CG_MSG_STREAM_DATA	Сообщение прихода потоковых данных. Поле data_size содержит размер полученных данных, data указывает на сами данные. Само сообщение содержит дополнительные поля, которые описываются структурой cg_msg_streamdata_t. Подробнее о получении данных будет рассказано ниже в данном разделе.
CG_MSG_P2REPL_ONLINE	Переход потока в состояние online - это означает, что получение начального среза было завершено и следующие сообщения CG_MSG_STREAM_DATA будут нести данные онлайн. Данное сообщение не содержит дополнительных данных и его поля data и data_size не используются.
CG_MSG_P2REPL_LIFENUM	Изменен номер жизни схемы. Такое сообщение означает, что предыдущие данные, полученные по потоку, не актуальны и должны быть очищены. При этом произойдёт повторная трансляция данных по новому номеру жизни схемы данных. Поле data сообщения указывает на целочисленное значение, содержащее новый номер жизни схемы; поле data_size содержит размер целочисленного типа. Подробнее про обработку номера жизни схемы в конце данного раздела.
CG_MSG_P2REPL_CLEARDELETED	Произошла операция массового удаления устаревших данных. Поле data сообщения указывает на структуру cg_data_cleardeleted_t, в которой указан номер таблицы и номер ревизии, до которой данные в указанной таблице считаются удаленными. Если ревизия в cg_data_cleardeleted_t == CG_MAX_REVISON, то последующие ревизии продолжатся с 1.
CG_MSG_P2REPL_REPLSTATE	Сообщение содержит состояние потока данных; присылается перед закрытием потока. Поле data сообщения указывает на строку, которая в закодированном виде содержит состояние потока данных на момент прихода сообщения - сохраняются схема данных, номера ревизий таблиц и номер жизни схемы на момент последнего CG_MSG_TN_COMMIT (Внимание: при переоткрытии потока с replstate ревизии, полученные после последнего CG_MSG_TN_COMMIT, будут присланы повторно!) Эта строка может быть передана в вызов cg_lsn_open в качестве параметра "replstate" по этому же потоку в следующий раз, что обеспечит продолжение получения данных с момента остановки потока.

При приходе события CG_MSG_STREAM_DATA параметр msg пользовательской функции обратного вызова содержит указатель на расширенную структуру данных:

```
struct cg_msg_streamdata_t
{
    uint32_t type;     /// Message type = CG_MSG_STREAM_DATA
    size_t data_size;  /// Data size
    void* data;        /// Data pointer
    int64_t owner_id;  /// Message owner id
    size_t msg_index;  /// Message number in active scheme
    uint32_t msg_id;   /// Unique message ID (if applicable)
    const char* msg_name; /// Message name in active scheme
    int64_t rev;          /// Message sequence number
    size_t num_nulls;     /// Size of presence map
```

```
    uint8_t* nulls;       /// Presence map. Contains 1 for NULL fields
    uint64_t user_id;     /// User ID message is intended for
};
```

Доступ к расширенной структуре осуществляется следующим способом:

```
CG_RESULT dataCallback(cg_conn_t* conn,
                       cg_listener_t* listener,
                       struct cg_msg_t* msg,
                       void* data)
{
    switch(msg->type)
    {
        case CG_MSG_STREAM_DATA:
        {
             // приведение указателя к расширенной структуре
            cg_msg_streamdata_t* replmsg = (cg_msg_streamdata_t*)msg;
            // здесь можно использовать расширенную структуру
            ...
        }
        ...
    }
}
```

С помощью данной структуры можно узнать номер таблицы и её имя в схеме данных - эта информация доступна в полях msg_index и msg_name структуры. Для реплики Plaza-2 поле msg_id не используется и его значение равно 0. Поле rev содержит ревизию (номер обновления) записи в таблице, а поле nulls может содержать указатель на массив байт, значения которых определяют, имеется ли конкретное поле в записи или отсутствует.

Данные, на которые ссылается указатель data сообщения, структурированы в соответствие со схемой данных, используемых в данной подписке. О том, что такое схемы данных и каким образом можно получить доступ к интересующим полям записи, можно узнать в следующем разделе.

Номер жизни - это атрибут схемы данных, который служит для определения актуальности полученных в потоке данных. Номер жизни схемы у источника реплики (сервера) и клиента должны совпадать. Сравнение производится при открытии потока репликации. Если номера жизни не совпадаю, это означает, что предыдущие данные, полученные в потоке, не актуальны и должны быть очищены. Сигналом о таком событии служит специальная нотификация CG_MSG_P2REPL_LIFENUM, содержащая новый номер жизни схемы.

Клиентский номер жизни схемы задается в параметрах функции cg_lsn_open:

- cg_lsn_open(lsn, "lifenum=%d"). Допускается задавать "lifenum=0", строка "lifenum=" также является допустимым вариантом и эквивалентна заданию "lifenum=0".
- cg_lsn_open(lsn, "replstate=%s") - восстановление из строки состояния потока реплики. Строка состояния - это строка, которая в закодированном виде содержит состояние потока данных на момент его предыдущего закрытия. Под состоянием потока здесь понимается: схема данных, номера ревизий таблиц и номер жизни схемы. Данная строка приходит в нотификации CG_MSG_P2REPL_REPLSTATE перед закрытием потока данных.

Номер жизни не может быть задан одновременно в 'lifenum' и 'replstate'.

Номер жизни схемы всегда десятичное число. Если значение номера жизни не будет десятичным числом, открытие подписки произойдет с ошибкой.

После передачи клиентского номера жизни запускается процесс его верификации. Возможны 3 ситуации:

- переданный clientLifeNumber < serverLifeNumber. В этом случае подписка открывается, клиентский код перед CG_MSG_OPEN получает сообщение CG_MSG_P2REPL_LIFENUM, в поле данных которого указан текущий серверный номер жизни. При приходе данного сообщения необходимо очистить все старые данные и получить их заново.
- переданный clientLifeNumber = serverLifeNumber. Объект "Подписчик" готов к работе сразу, без уведомлений и очистки данных.
- переданный clientLifeNumber > serverLifeNumber. Объект "Подписчик" переходит в состояние CG_STATE_ERROR.

Допускается вообще не передавать номер жизни при открытии подписки. В этом случае объект "Подписчик" начнет работу по серверному номеру жизни, получая нотификацию CG_MSG_P2REPL_LIFENUM.

1.6. Работа со схемами данных

Любые данные, которые принимаются или отправляются в процессе взаимодействия клиентского ПО с торговой системой, специальным образом структурированы. Для описания структуры конкретных сообщений применяются схемы данных.

Схема данных описывает множество возможных сообщений для выбранного канала данных (подписки или публикации), поля и типы этих сообщений, а также задает правила доступа к этим данным. Схема данных описывается следующей структурой:

```
struct cg_scheme_desc_t {
  // Тип схемы
  uint32_t scheme_type;

  // свойства схемы
  uint32_t features;

  // Количество сообщений в схеме
  size_t num_messages;

  // Указатель на список описаний сообщений
  struct cg_message_desc_t* messages;

  /// Свойства схемы
  struct cg_value_pair_t* hints;

};
```

В настоящее время поддерживается единственный тип схемы, которому соответствует идентификатор 1 - данные хранятся в бинарном виде с выравниванием 4 байта без поддержки опциональных полей.

Поле features описывает доступную информацию в схеме - по этому полю можно узнать, заданы ли значения по умолчанию для полей в данной схеме, имеют ли поля или сообщения описания и т.п. За это отвечают константы CG_SCHEME_BIN_*.

Поле num_messages задает количество сообщений в схеме, а поле messages указывает на первое из них. Сообщения являются основным объектом, описывающим конкретные структуры данных и используются во всех видах подписки и публикации; например, для реплики Plaza-2 сообщения описывают события обновления данных в таблицах.

Каждое сообщение описывается структурой:

```
struct cg_message_desc_t {
    /// указатель на следующее сообщение
    struct cg_message_desc_t* next;

    /// размер блока сообщения
    size_t size;

    /// Количество полей в сообщении
    size_t num_fields;

    /// Указатель на массив описаний полей
    struct cg_field_desc_t* fields;

    /// Идентификатор сообщения
    /// Может быть 0, если идентификатор у сообщения отсутствует
    uint32_t id;

    /// Указатель на имя сообщения
    /// Может быть NULL - в этом случае у сообщения отсутствует имя
    char *name;

    /// Указатель на описание сообщения
    /// Может быть NULL - в этом случае у сообщения отсутствует описание
    char *desc;

    /// Свойства сообщения
    struct cg_value_pair_t* hints;

    /// количество индексов сообщения
    size_t num_indices;

    /// Указатель на первый индекс
    struct cg_index_desc_t* indices;

    ///Поле предназначено для внутреннего пользования
    int64_t owner_id

   /// Величина выравнивания
    size_t align;

};
```

Поле next указывает на следующее сообщение в схеме или содержит значение NULL, что означает, что данное сообщение - последнее. Таким образом, сообщения упорядочены в связанном списке и доступ к ним можно получить с помощью цикла вида:

```
cg_scheme_desc_t* schemedesc; // инициализированный указатель на схемы данных
for (cg_message_desc_t* msgdesc = schemedesc->messages;
     msgdesc; msgdesc = msgdesc->next)
{
    // здесь можно работать с описанием сообщения,
    // которое содержится в msgdesc
    ...
}
```

Поле size структуры описания сообщения задает размер блока в байтах, который требуется для хранения данных сообщения целиком. Поле num_fields содержит количество полей в сообщении, а fields указывает на первое поле сообщения.

Поля id, name и desc содержат идентификатор сообщения, его имя и описание. Идентификатор, имя или описание могут отсутствовать, в случае, если конкретная схема не описывает эти значения для сообщений.

Поле hints содержит указатель на структуру, которая может быть использована пользователем для автоматической настройки своей программы на определенный вид или способ обновления данных:

```
struct cg_value_pair_t {
                /// Pointer to the next list entry
                struct cg_value_pair_t *next;
                /// Key, required
                char * key;
                /// Value, may be null
                char * value;
};
```

, где:

- key – ключ хинта;
- value – значение хинта;
- next – указатель на следующий хинт;
- next = 0 – конец списка.

Для хинтов request и reply, подсказывающих, какие сообщения нужно отправлять, а какие приходят в ответ, теперь предусмотрены следующие параметры:

- request=1 - значение передаётся в поле value;
- reply=1 - значение передаётся в поле value;

Для параметра replies предусмотрено значение 129,99,100 (передаётся в поле value).

Поле num_indices содержит количество индексов, а поле indices указывает на первый индекс. Первый индекс в списке всегда является уникальным primary ключом.

Индексы описываются следующей структурой:

```
struct cg_index_desc_t {
    /// указатель на следующий индекс
    struct cg_index_desc_t * next;

    /// количество полей в ключе
    size_t num_fields;

    /// указатель на описание первого поля в ключе
    struct cg_indexfield_desc_t* fields;

    /// имя ключа
    char* name;

    /// описание ключа
    char* desc;

    /// Свойства индекса
    struct cg_value_pair_t* hints;
};
```

Поле next указывает на следующий индекс в схеме или содержит значение NULL, что означает, что данный индекс - последний.

Поле num_fields указывает на количество полей в индексе.

Поле fields указывает на первое поле в индексе.

Поля name и desc содержат название индекса и его описание.

Поле hints содержит хинты для индекса, например, "unique".

Поля индекса описываются структурой:

```
struct cg_indexfield_desc_t {
    /// указатель на следующее описание поля ключа
    struct cg_indexfield_desc_t* next;

    /// указатель на поле
    struct cg_field_desc_t* field;

    /// порядок сортировки
    uint32_t sort_order;
};
```

Поле next указывает на следующее поле в индексе или содержит значение NULL, что означает, что данное поле - последнее.

Поле field указывает на структуру - описатель поля схемы.

Поле sort_order задает порядок сортировки 0 - по возрастанию, 1 - по убыванию.

Поля сообщения описываются следующей структурой:

```
/// Описание поля сообщения
struct cg_field_desc_t {
    /// указатель на следующее поле
    struct cg_field_desc_t* next;

    /// Идентификатор поля
    /// Может быть 0, если идентификатор поля отсутствует
    uint32_t id;

    /// Имя поля
    /// Может быть NULL - в этом случае у поля отсутствует имя
    char* name;

    /// Описание поля
    /// Может быть NULL - в этом случае у поля отсутствует описание
    char* desc;

    /// Тип поля
    char*  type;

    /// Длина значения данного поля
    size_t size;

    /// Смещение относительно начала сообщения
    size_t offset;

    /// Указатель на значение поля по умолчанию.
    /// Указывает на буфер размером size, в котором хранятся данные в формате type
    /// Если null, то значение по-умолчанию отсутствует
    void* def_value;

    /// Указатель на список значений, принимаемых полями
    struct cg_field_value_desc_t* values;

    /// Указатель на структуру со свойствами сообщения
    struct cg_value_pair_t* hints;

    /// Количество значений, которое содержится в поле values
    size_t num_values;

    /// Максимальное кол-во полей. Значение по умолчанию "1"
    size_t max_count;

    /// Для внутреннего использования
    struct cg_field_desc_t * count_field;

    /// Указатель на описание сообщений с типом полей "m"
    struct cg_message_desc_t * type_msg
```

```
};
```

Поле next указывает на описание следующего поля сообщения или содержит NULL в случае последнего поля. Поля id, name и desc задают идентификатор, имя и описание поля; для различных схем сообщений эти поля могут содержать пустые значения. Поле type содержит название типа поля, по которому можно определить способы работы с этим полем. Наиболее часто используемыми типами полей являются следующие типы:

i1, i2, i4, i8	Целочисленные знаковые значения размером 1, 2, 4 и 8 байт соответственно
u1, u2, u4, u8	Целочисленные беззнаковые значения размером 1, 2, 4 и 8 байт соответственно
a	Строка символов размером 1 байт
cNN	Строка с максимальной длиной NN (завершаемая байтом со значением 0)
dMM.NN	Число в двоично-десятичном формате с общим количеством знаков MM и количеством знаков после запятой NN
bNN	Блок с неформатированными двоичными данными размера NN
t	Структура, описывающая дату и время
f	Число с плавающей точкой двойной точности размером 8 байт

Поле size содержит размер значения поля, а поле offset - смещение данного поля в байтах от начала блока данных. Эта информация позволяет однозначно идентифицировать расположение и размер интересующего поля в блоке данных сообщения.

Поле def_value содержит указатель на значение по умолчанию. Тип и размер значения полностью совпадают с типом и размером поля, таким образом, инициализация поля значением по-умолчанию может быть выполнена простым копированием. Значение NULL поля def_value соответствует отсутствию значения по умолчанию.

Поле values содержит указатель на первое значение списка допустимых значений. Значение NULL поля values означает, что поле может принимать любое значение из области определения типа.

```
struct cg_field_value_desc_t {
    /// указатель на следующее значение
    struct cg_field_value_desc_t* next;

    /// название значения
    char* name;

    /// описание значения
    char* desc;

    /// указатель на допустимое значение
    void* value;

    /// для полей типа integer (i[1-8], u[1-8]) маска,
    /// определяющая диапазон занимаемых значением бит
    void* mask;
};
```

Поле next указывает на следующее значение списка допустимых значение поля или содержит значение NULL, что означает, что данное значение - последнее.

Поля name и desc содержат наименование и описание значения.

Поле value содержит указатель на значение поля, при этом размер и тип значения совпадают с размером и типом самого поля.

Поле mask используется для группировки взаимоисключающих значений, при этом значения с разными масками могут комбинироваться.

Предположим, что мы имеем дело с подпиской на получение потока репликации Plaza-2 со следующей схемой данных:

```
[dbscheme:FutTrade]
table=orders_log
table=heartbeat

[table:FutTrade:orders_log]
field=replID,i8
field=replRev,i8
field=replAct,i8
field=id_ord,i8
field=sess_id,i4

[table:FutTrade:heartbeat]
```

```
field=replID,i8
field=replRev,i8
field=replAct,i8
field=server_time,t
```

Подобный формат описания схем в виде ini-файлов принят в системе Plaza-2.

Эта схема описывает две таблицы (два сообщения) с некоторым набором полей в каждой из них. Предположим, что нас интересует получение номеров заявок из таблицы orders_log и событий синхронизации серверного времени из таблицы heartbeat - эти значения содержатся в поле id_ord сообщения orders_log и поле server_time сообщения heartbeat, соответственно.

Существуют два способа разбора получаемых данных - статический, с использованием заранее заготовленных структур данных и динамический, с вычислением смещений интересующих полей в момент получения схемы.

Статический подход состоит в том, что для интересующих потоков на этапе разработки фиксируются схемы данных, которые будут использованы в дальнейшем; затем по схемам данных вручную или автоматически, например, с помощью утилиты schemetool, генерируются описания структур языка C, соответствующие форматам бинарных блоков данных для каждого из принимаемых сообщений (для языков типа Java или .NET вместо структур генерируется код, который разбирает бинарные блоки сообщений). В процессе работы данные пришедшего сообщения отображаются на структуру, соответствующую типу сообщения и осуществляется требуемая обработка данных.

Такой подход позволяет упростить разработку с одной стороны, с другой - фиксирует определённый формат схем данных, что потребует повторной подготовки структур данных или кода разбора бинарных блоков, в случае изменения схем данных - старые структуры могут перестать отображаться на новые форматы сообщений, что в некоторых случаях может привести к сложнообнаруживаемым ошибкам.

Важно

В случае использования заранее заготовленных структур для отображения данных, следует использовать клиентскую схему при инициализации потока. При изменении схемы получаемых данных, структуры следует сгенерировать заново.

Заранее заготовленные структуры данных или код для разбора бинарных блоков можно сгенерировать с использованием утилиты schemetool.

Выглядеть это может следующим образом:

```
/// Описание структур, полученное с использованием
/// утилиты schemetool

#pragma pack(push, 4)
/// Scheme "FutTrade" description

    struct orders_log
    {
        signed long long replID;
        signed long long replRev;
        signed long long replAct;
        signed long long id_ord;
        signed int sess_id;

    };
    const int orders_log_index = 0;

    struct heartbeat
    {
        signed long long replID;
        signed long long replRev;
        signed long long replAct;
        struct cg_time_t server_time;

    };
    const int heartbeat_index = 1;


#pragma pack(pop)

/// в обработчике подписки

case CG_MSG_STREAM_DATA:
{
     cg_msg_streamdata_t* replmsg = (cg_msg_streamdata_t*)msg;
     if (replmsg->msg_index == orders_log_index)
     {
```

```
        orders_log* ordlog = (orders_log *)replmsg->data;
        printf ("Order ID = %lld\n", ordlog->id_ord);
    }
    else
    if (replmsg->msg_index == heartbeat_index)
    {
        heartbeat* hb = (heartbeat *)replmsg->data;
        printf ("Server time = %d:%d:%d.%d\n",
                hb->server_time.hour, hb->server_time.min,
                hb->server_time.sec, hb->server_time.ms);
    }
}
```

Динамический подход предполагает отсутствие явно зафиксированной схемы данных, напротив - схема данных каждый раз получается из источника схемы (например, с сервера репликации), а код пользователя анализирует её и осуществляет поиск интересующих сообщений и полей в них.

Подобный подход позволяет создать более универсальную систему, которая сможет переживать не критичные изменения схем данных; с другой стороны динамический анализ схемы является более сложным в реализации.

Первым шагом такого подхода является подготовка информации об интересующих полях - нужно проанализировать используемую схему потока данных и запомнить номера интересующих сообщений и смещения интересующих полей:

```
/// переменные, которые будут содержать нужную для анализа
/// поступающих данных информацию
size_t index_orders_log; /// индекс сообщения orders_log в схеме
size_t offset_id_ord; /// смещение поля id_ord в блоке

size_t index_hearbeart; /// индекс сообщения heartbeat в схеме
size_t offset_server_time; /// смещение поля server_time в блоке
```

Этой информации достаточно, для того, чтобы в момент прихода данных идентифицировать тип сообщения и найти нужное поле в бинарном блоке. Заполнение этих полей выполняется следующим образом:

```
cg_scheme_desc_t* scheme; // инициализированное описание схемы данных

size_t msgidx = 0;
for (cg_message_desc_t* msgdesc = schemedesc->messages;
     msgdesc; msgdesc = msgdesc->next, msgidx ++)
{
    size_t fieldindex = 0;
    if (strcmp(msgdesc->name, "orders_log") == 0)
    {
        index_orders_log = msgidx;
        for (cg_field_desc_t* fielddesc = msgdesc->fields;
             fielddesc; fielddesc = fielddesc->next, fieldidx ++)
            if (strcmp(fielddesc->name, "id_ord") == 0 &&
                strcmp(fielddesc->type, "i8") == 0)
                offset_id_ord = fieldidx;
    }
    if (strcmp(msgdesc->name, "heartbeat") == 0)
    {
        index_heartbeat = msgidx;
        for (cg_field_desc_t* fielddesc = msgdesc->fields;
             fielddesc; fielddesc = fielddesc->next, fieldidx ++)
            if (strcmp(fielddesc->name, "server_time") == 0 &&
                strcmp(fielddesc->type, "t") == 0)
                offset_server_time = fieldidx;
    }
}
```

В приведенном коде осуществляется последовательный перебор всех сообщений схемы и поиск нужных полей для интересующих сообщений. При этом осуществляется проверка типов полей в соответствие с ожиданиями.

Обработка получаемых данных выполняется следующим образом:

```
/// в обработчике подписки

case CG_MSG_STREAM_DATA:
{
    cg_msg_streamdata_t* replmsg = (cg_msg_streamdata_t*)msg;

    /// приведение к char*, чтобы затем правильно прибавлять offset в байтах
    char* data = (char*)replmsg->data;
```

```
    if (replmsg->msg_index == index_orders_log)
    {
        int64_t id_ord = *((int64_t*)(data + offset_id_ord));
        printf ("Order ID = %lld\n", id_ord);
    }
    else
    if (replmsg->msg_index == index_heartbeat)
    {
        cg_time_t *srvtime = (cg_time_t*)(data + offset_server_time);
        printf ("Server time = %d:%d:%d.%d\n",
                srvtime->hour, srvtime->min, srvtime->sec, srvtime->ms);
    }
}
```

Этот пример будет выводить на экран идентификатор заявки в момент прихода данных по изменению состояния заявки и серверное время в момент прихода соответствующего сообщения.

Показанный пример демонстрирует следующие полезные практики при создании кода:

- Контроль типов данных при анализе схемы - обеспечивает корректную диагностику ошибок при изменении схем
- Использование численных идентификаторов сообщений вместо строк - положительно влияет на производительность, так вместо более дорогой операции сравнения строк можно обойтись сравнением двух чисел
- Отсутствие копирования данных - не нужно обращаться к каждому полю вызовом специальной функции; данные доступны непосредственно в буфере сообщения
- Поддержка эволюции схем данных - код, анализирующий схему при открытии потока, сможет работать с разными вариантами схем данных, без необходимости изменения зашитых идентификаторов и перекомпиляции

1.6.1. Политика использования схем данных

При обновлении версии ТКС Спектра, как правило, происходят изменения схем публичных потоков. В схемы могут добавиться новые потоки, а в уже существующих потоках могут быть добавлены или удалены какие-то поля. Эти изменения затрагивают и дистрибутив Cgate, в котором будут присутствовать обновлённые схемы данных. В дистрибутиве Cgate схемы хранятся в каталогах, соответствующих номерам версий ТКС Спектра: SPECTRA53, SPECTRA56. Каталоги со схемами SPECTRA53, SPECTRA56 не удаляются; это сделано для того, чтобы при обновлении шлюзового ПО клиенту не пришлось перенастраивать своё ПО.

Существуют два способа работы с сообщениями:

- Динамический способ. Клиент при получении сообщения CG_MSG_OPEN получает схему из lsn. В полученной схеме пользователь находит интересующие его поля и запоминает их смещения в байтах относительно начала сообщения. При получении сообщения CG_MSG_STREAM_DATA пользователь, учитывая величину смещения в байтах, извлекает значения для требуемых полей.
- Статический способ. Клиент при помощи утилиты schemetool или каким-либо другим способом создаёт структуру или класс, предоставляющий доступ к полям сообщения. При получении сообщения CG_MSG_STREAM_DATA пользователь при помощи соответствующих методов или полей класса/структуры получает значения для требуемых полей.

... и четыре способа задания схемы данных:

- Схема данных задается в ini-файле на диске;
- Схема данных задается из встроенной в код строки вида:

```
|STRING|base64encodedscheme
```

- Используется серверная схема данных;
- Используется схема из репозитория схем данных (только для протокола mq).

Ниже описаны проблемы, возникающие при "неправильном" использовании схем данных:

Способ задания схемы	Способ работы с сообщениями	Преимущества	Недостатки	Комментарий	Оценка
ini-файл на диске	Динамический	Нет необходимости перекомпилировать исходный код при изменении схем (при наличии всех требуемых полей)	Выяснить, что ПО использует поле, которое удалено из шлюза ТКС Спектра, можно только в runtime.	Рекомендуем проверять ПО во время обновлений на тестовом публичном полигоне ТКС Спектра.	Можно использовать.
	Статический	Простота кода, не нужно запоминать величину смещения	Структура и код на диске могут оказаться несогласованны-		Не рекомендуется к использованию.

Способ задания схемы	Способ работы с сообщениями	Преимущества	Недостатки	Комментарий	Оценка
		для требуемых полей.	ми, в результате в ПО могут попасть невалидные данные.		
Встроенная в код строка	Динамический	Нет необходимости перекомпилировать исходный код при изменении схем (при наличии всех требуемых полей)	Выяснить, что ПО использует поле, которое удалено из шлюза ТКС Спектра, можно только в runtime.	Схема поменяться не может. Рекомендуем проверять ПО во время обновлений на тестовом публичном полигоне ТКС Спектра.	Можно использовать.
	Статический	Простота кода, не нужно запоминать величину смещения для требуемых полей.	Выяснить, что ПО использует поле, которое удалено из шлюза ТКС Спектра, можно только в runtime.	Схема поменяться не может. Рекомендуем проверять ПО во время обновлений на тестовом публичном полигоне ТКС Спектра.	Можно использовать.
Серверная схема	Динамический	Нет необходимости перекомпилировать исходный код при изменении схем (при наличии всех требуемых полей)	Выяснить, что ПО использует поле, которое удалено из шлюза ТКС Спектра, можно только в runtime.	Большая вероятность того, что схема на сервере изменится.	Можно использовать.
	Статический		Структура и схема на сервере могут оказаться несогласованными, в результате в ПО могут попасть невалидные данные.	Большая вероятность того, что схема на сервере изменится.	Не рекомендуется к использованию.
Репозиторий схем (только для протокола mq)	Динамический	Нет необходимости перекомпилировать исходный код при изменении схем (при наличии всех требуемых полей)	Выяснить, что ПО использует поле, которое удалено из шлюза ТКС Спектра, можно только в runtime.	Большая вероятность того, что схема на сервере изменится.	Можно использовать.
	Статический		Структура и схема в репозитории схем могут оказаться несогласованными, в результате в ПО могут попасть невалидные данные.	Большая вероятность того, что схема на сервере изменится.	Не рекомендуется к использованию.

Примечание

Как задать схему из строки:

1. В утилите schemetool передать параметр:

   ```
   -Dgen-scheme-string=1
   ```

   (в версии ТКС Спектра 5.6 параметр включен по умолчанию);

2. Сгенерировать url lsn:

   ```
   p2repl://STREAM_NAME;scheme=SCHEME_STRING_FROM_SCHEMETOOL
   ```

Как задать серверную схему:

1. Не задавать параметр scheme, тогда будет использоваться серверная схема

   ```
   p2repl://STREAM_NAME;
   ```

Как задать схему из репозитория (только для протокола mq):

1. Не задавать параметр scheme, тогда будет использоваться последняя схема из репозитория

```
p2mq://FORTS_SRV;category=FORTS_MSG
```

Для выбора определённой версии схемы необходимо задать номер версии в параметре version, например

```
p2mq://FORTS_SRV;category=FORTS_MSG;version=5.8
```

1.7. Отправка транзакций и получение ответов

Отправка транзакций FORTS и получение ответов об их исполнении выполняется с помощью объектов «Публикатор», «Подписчик». Объект публикатор создаётся в привязке к соединению вызовом функции cg_pub_new, например, так:

```
result = cg_pub_new(conn,
        "p2mq://FORTS_SRV;category=FORTS_MSG;"
        "name=PUB;scheme=|FILE|ini/forts_messages.ini|message ",
        &pub);
```

В этом примере pub инициализируется объектом "Публикатор", настроенным на отправку транзакций FORTS по схеме, хранящейся в подкаталоге ini с именем файла forts_messages.ini и именем схемы “message” через соединение conn. Публикатору присвоено имя “PUB”, на которое будет ссылаться подписчик.

В случае успешного вызова функции cg_pub_new объект находится в инициализированном, но не активном состоянии. Дальнейшая работа с публикатором возможна после вызова функции cg_pub_open:

```
result = cg_pub_open(pub, 0);
```

Параметры открытия для объекта публикатор в настоящее время не предусмотрены, поэтому в качестве второго параметра передаётся пустой указатель.

После того, как публикатор создан и открыт, можно создавать и отправлять транзакции. Для создания транзакции можно воспользоваться функцией cg_pub_msgnew.

```
result = cg_pub_msgnew(pub, CG_KEY_NAME, “AddOrder”, &msgptr);
```

В данном случае будет создано сообщение для постановки заявки SPECTRA (транзакция AddOrder) по имени, и указатель на него будет записан в переменную msgptr. При помощи функции cg_pub_msgnew можно так же создавать сообщения по его номеру в активной схеме и идентификатору.

Сообщение представляет собой указатель на структуру cg_msg_data_t:

```
struct cg_msg_data_t
{
  uint32_t type;    // Тип сообщения = CG_MSG_P2REPL_DATA
  size_t data_size; // Размер данных
  void* data;       // Указатель на данные

  size_t msg_index; // Номер сообщения в активной схеме
  uint32_t msg_id;  // Уникальный идентификатор сообщения
  const char* msg_name; // Имя сообщения в схеме

  uint32_t user_id; // Пользовательский номер сообщения
  const char* addr; // Адрес получателя
  struct cg_msg_data_t* ref_msg; // Связанное сообщение (сейчас не используется)
};
```

Поле data структуры указывает на буфер в памяти соответствующего размера, который необходимо заполнить согласно активной схеме. Проще всего это сделать приведя этот указатель к правильной структуре. Например, так:

```
ord = (struct AddOrder*)msgptr->data;
strcpy(ord->broker_code, "HB00");
```

Создать описание структуры из схемы можно при помощи утилиты schemetool.

После того, как сообщение создано и заполнено, его нужно отправить при помощи функции cg_pub_post:

```
 result = cg_pub_post(pub, msgptr, CG_PUB_NEEDREPLY);
```

Флаг CG_PUB_NEEDREPLY означает, что мы хотим получать ответы в соответствующий подписчик lsnreply.

После того, как сообщение отправлено, его можно уничтожить при помощи функции cg_pub_msgfree:

```
result = cg_pub_msgfree(pub, msgptr);
```

При отсылке однотипных заявок, более эффективно использовать созданное сообщение повторно.

Публикатор закрывается вызовом функции cg_pub_close:

```
result = cg_pub_close(pub);
```

При этом происходит отключение публикатора от объекта соединение; сам объект остаётся в инициализированном состоянии и может быть открыт повторно. Уничтожение объекта происходит посредством вызова cg_pub_destroy:

```
result = cg_pub_destroy(pub);
```

После этого объект pub освобождается и дальнейшая работа с ним невозможна.

Подписчик для получения ответов на команды создаётся следующим образом:

```
result = cg_lsn_new(conn, “p2mqreply://;ref=PUB”, replyCB, user_data, &lsnreply);
```

Этот вызов инициализирует переменную lsnreply специальным объектом-подписчиком для получения ответов на отправленные публикатором сообщения. Связь между подписчиком и публикатором осуществляется по имени, в данном случае это имя “PUB”, параметр “ref=PUB” строки инициализации устанавливает эту связь. С одним публикатором можно сопоставить один подписчик. Имена соответствующих пар должны быть уникальны. Сообщения, содержащие ответы на транзакции, а также других событиях публикатора будут приходить в функцию replyCB. Жизненный цикл данного объекта «подписчик» ничем не отличается от жизненного цикла подписчика реплики, рассмотренного в соответствущем разделе, за исключением того, что в replyCB приходят не сообщения системы репликации, а простые единичные сообщения MQ, описываемые структурой cg_msg_data_t, которые ссылаются на данные, описываемые схемой соответствующего публикатора, а так же нотификация CG_MSG_P2MQ_TIMEOUT, в случае, если был превышен интервал ожидания ответа на сообщение. При закрытии соединения, на все отправленные mq запросы, на которые не был получен ответ приходят сообщения с нотификацией CG_MSG_P2MQ_TIMEOUT. При возврате ошибки из callback пользователя подписчик закрывается. Подписчик p2mqreply должен создаваться после создания публикатора mq, в противном случае ссылка на объект ref=PUB будет не валидной. Порядок разрушения этих связанных объектов не важен.

Пользовательский обработчик ответов может выглядеть следующим образом:

```
CG_RESULT ClientMessageCallback(cg_conn_t* conn, cg_listener_t* listener, struct cg_msg_t* msg, void* data)
{
  switch (msg->type)
  {
    case CG_MSG_DATA:
    {
    uint32_t* data = msg->data;
    printf("Client received reply [id:%d, data: %d, user-id: %d, name: %s]\n",
      ((struct cg_msg_data_t*)msg)->msg_id,
      *((uint32_t*)msg->data),
      ((struct cg_msg_data_t*)msg)->user_id,
      ((struct cg_msg_data_t*)msg)->msg_name);

    {
      struct scheme_desc_t* scheme;
      size_t bufSize;

      if (cg_lsn_getscheme(listener, &scheme) != CG_ERR_OK)
        scheme = 0;

      if (cg_msg_dump(msg, scheme, 0, &bufSize) == CG_ERR_BUFFERTOOSMALL)
      {
        char* buffer = malloc(bufSize+1);

        bufSize++;
        if (cg_msg_dump(msg, scheme, buffer, &bufSize) == CG_ERR_OK)
          printf("client dump: %s\n", buffer);
        free(buffer);

      }
    }
    break;
    }
  case CG_MSG_P2MQ_TIMEOUT:
  {
    printf("Client reply TIMEOUT\n");
    break;
  }
  default:
    printf("Message 0x%X\n", msg->type);
  }
  return CG_ERR_OK;
}
```

Этот пользовательский обработчик либо выводит дамп сообщений с помощью вспомогательной функции cg_msg_dump, либо, в случае превышения ожидания ответа, отслеживает эту ситуацию и выводит на экран соответствующие сообщения.

Для осуществления связи между отправленными сообщениями и ответами на них, следует использовать поле user_id структуры cg_msg_data_t: задание user_id у отправляемого сообщения обеспечивает получение ответного сообщения с тем же user_id.

1.8. Сценарий работы с p2sys

При установке соединения p2sys подписчик получает 2 сообщения типа CG_MSG_DATA:

- **ConnectionConnected** (msgid = 3). Поле данных содержит переменную state типа int32 со статусом операции, равным 2;
- **RouterDisconnected** (msgid = 2). Поле данных также содержит переменную state типа int32 со значением 1.

В случае успеха пользователь может послать исходящий запрос аутентификации. Для этого он должен создать сообщение **RouterLogin** (msgid = 1) типа CG_MSG_DATA. В поле данных этого сообщения должна находиться строка формата "USERNAME=%имя_пользователя%;PASSWORD=%пароль%".

Если логин и пароль являются верными, подписчик примет сообщение **RouterConnected** (msgid = 1). В поле данных содержится пользовательский логин в виде строки. Если же аутентификация отклонена, подписчик примет сообщение **LogonFailed** (msgid = 5). В поле данных также содержится статус операции равный 1.

Для отключения от системы Plaza-II необходимо послать сообщение **RouterLogout** (msgid = 2). Поле данных в этом сообщении игнорируется. В ответ подписчик должен получить оповещение **RouterDisconnected** со статусом, равным 1.

По окончании работы необходимо закрыть объекты стандартным способом: сначала публикатор и подписчик, затем соединение.

1.9. Объекты протокола изменения пароля

Для изменения пароля пользователя предусмотрен специальный протокол p2mqpwd. Протокол обеспечивает защищённую передачу данных: пароль и логин пользователя не передаются по сети в открытом виде. API протокола реализовано объектами публикатора и подписчика.

1.9.1. Публикатор p2mqpwd

Публиктор p2mqpwd предназначен для отправки команды смены пароля в торговую систему. Перед использованием публикатора необходимо добавить подсистему mqpwd в окружение:

```
result = cg_env_open("subsystems=mq,replclient,mqpwd;ini=ini/settings.ini;key=72395823576;");
```

Соединение для публикатора стандартное, и может быть любого типа, как p2tcp, так и p2lrpcq:

```
cg_conn_t* conn;
result = cg_conn_new("p2tcp://127.0.0.1:4001;app_name=pwd_changer", &conn);
result = cg_conn_open(conn, 0);
```

Схема публикатора фиксирована и находится в составе дистрибутива CGate по адресу: *<каталог инсталляции>\SpectraCGate\SDK\scheme\SPECTRAXX\change_password_messages.ini.*

Задать схему можно как строкой, так и указанием пути к файлу. Остальные параметры публикатора, такие как категория сообщений и имя сервиса, аналогичны стандартному публикатору p2mq, используемому для посылки команд в торговую систему.

```
publisher_t* pub = 0;
result = pub_new(conn,
"p2mqpwd://FORTS_SRV;name=pwd_changer;category=FORTS_MSG;
scheme=|FILE|C:\Moscow Exchange\SpectraCGate\SDK\scheme\SPECTRAXX\change_password_messages.ini|
change_password_message",
&pub);
result = pub_open(pub, 0);
```

Сообщение для смены пароля называется ChangePassword и создаётся стандартным способом с помощью публикатора mqpwd:

```
cg_msg_data* msg;
result = cg_pub_msgnew(pub, CG_KEY_NAME, “ChangePassword", &msg);
```

Для смены пароля необходимо заполнить поля сообщения ChangePassword (указать старый и новый пароль) и послать сообщение в торговую систему аналогично тому, как посылаются другие команды:

```
result = cg_pub_post(pub, msg, CG_PUB_NEEDREPLY);
```

1.9.2. Подписчик p2mqpwdreply

Подписчик mqpwd предназначен для получения ответа от торговой системы. Для создания подписчика необходимо указать протокол p2mqpwd:

```
listener_t* lsn;
result = cg_lsn_new(conn, "p2mqpwdreply://;ref=pwd_changer;", &clientMessageCallback, NULL, &lsn);
result = cg_lsn_open(lsn, "");
```

В остальном использование подписчика p2mqpwdrely аналогично использованию стандартного подписчика p2mqreply.

Ответ от торговой системы может быть как положительным, так и содержать описание и код ошибки. Возникновение ошибки возможно из-за неправильных пользовательских данных или из-за ошибок системного уровня. При возникновении ошибки пароль пользователя в системе не меняется.

Получение ответа об успешном выполнении означает изменение пароля пользователя в торговой системе, при этом авторизация текущего соединения роутера не меняется. Для авторизации роутера с новым паролем необходимо изменить ини-файл роутера и перезапустить его или использовать протокол p2sys для авторизации из CGate.

2. Описание API

2.1. Общие соглашения

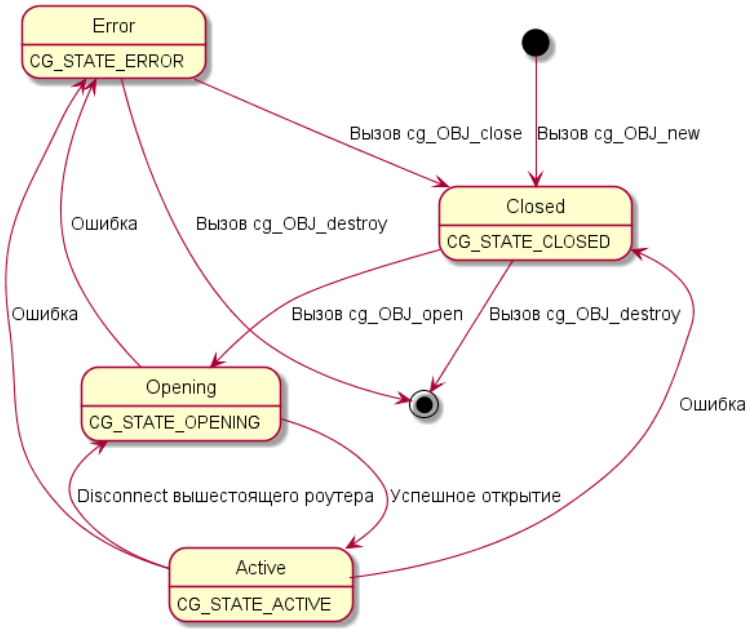
Программный интерфейс библиотеки построен с учетом ряда соглашений:

- Каждая функция API возвращает код ошибки
- Выходные параметры задаются в качестве указателей на переменные, куда следует поместить возвращаемое функцией значение и располагаются в конце списка параметров
- Функции имеют префиксы, как правило, состоящие из двух частей, первая - "cg_" означает принадлежность функции библиотеке Client Gate, вторая идентифицирует класс объектов, с которым работает та или иная функция
  - env_ - функции работы с общим окружением работы системы
  - conn_ - функции работы с соединением
  - lsn_ - функции работы с подписками
  - pub_ - функции работы с отправкой сообщений
  - log_ - функции работы с журналом работы

  , при этом существует несколько функций, обладающих только префиксом "cg_" - это вспомогательные и сервисные функции, которые не относятся к какой либо конкретной группе.
- Функции вида lsn_new, pub_new и т.п. создают и инициализируют объекты, которые затем должны быть освобождены соответствующими вызовами lsn_destroy, pub_destroy и т.п. В случае, если объекты не будут явно уничтожены, возникнут утечки памяти.

2.2. Жизненный цикл объектов

Объекты, доступ к которым предоставляется библиотекой, имеют жизненный цикл, описываемый следующей схемой:



Объекты в течение своего жизненного цикла существуют в следующих состояниях:

- CG_STATE_CLOSED

  Закрытое состояние. В этом состоянии объект создаётся (после вызова cg_OBJ_new) или переходит в него после вызова cg_OBJ_close. Объект может перейти в состояние CG_STATE_CLOSED в случае ошибки.
- CG_STATE_OPENING

  Состояние перехода из закрытого в активное состояние. В этом состоянии объект существует после вызова cg_OBJ_open и до перехода в состояние CG_STATE_ACTIVE или, в случае возникновения ошибки открытия объекта, в состояние CG_STATE_ERROR.
- CG_STATE_ACTIVE

  Активное состояние - основное рабочее состояние объекта. В этом состоянии возможна работа с объектом - обработка событий соединения, отправка или получение сообщений. В это состояние объект попадает после успешного завершения процесса

открытия из состояния CG_STATE_OPENING. Из этого состояние объект может перейти либо в состояние CG_STATE_CLOSED посредством вызова функции cg_OBJ_close, либо в состояние CG_STATE_ERROR в случае возникновения ошибки. Connection может перейти из CG_STATE_ACTIVE в CG_STATE_OPENING при разрыве соединения между своим и вышестоящим роутером.

Примечание: объект connection при закрытии закрывает все связанные с ним объекты (подписчики и публикаторы).

- CG_STATE_ERROR

  Состояние ошибки. В нём объект оказывается, если в процессе его открытия или работы произошла ошибка. Из этого состояния объект можно перевести в закрытое состояние вызовом cg_OBJ_close или уничтожить объект вызовом cg_OBJ_destroy, если дальнейшая работа с ним не требуется.

Такая схема состояний используется для следующих объектов:

- Соединения cg_conn_t
- Подписчики cg_listener_t
- Публикаторы cg_publisher_t

2.3. Использование в многопоточном окружении

Библиотека CGate может быть использована в многопоточном окружении, но *не является потокобезопасной*. Это означает, что для корректной работы с библиотекой из нескольких потоков необходимо соблюдать специальные правила:

- Работа с объектом "Соединение" в каждый момент времени должна вестить только из одного потока.

  При этом корректным является создание соединения из одного потока, а работа с ним из другого - главное, чтобы несколько потоков не выполняли действия с соединением в одно и то же время. Если существует необходимость разделять соединение между несколькими потоками одновременно, следует воспользоваться примитивами синхронизации ОС для синхронизации доступа к объекту "Соединение".

- Работа с объектами "Подписчик" и "Публикатор" в каждый момент времени должна вестись только из одного потока, аналогично объекту "Соединение"

- Объект "Подписчик" привязан к конкретному соединению(тому, которое было задано при их создании) и работа с ним должна вестись из того же потока, из которого ведется работа с соединением. Объект "Публикатор" также привязан к конкретному соединению(тому, которое было задано при их создании), но работа с ним допускается из другого потока.

2.4. Запуск и остановка окружения

Для начала работы с библиотекой необходимо выполнить инициализацию окружения. Инициализация выполняется с помощью функции env_open:

```
CG_RESULT cg_env_open(const char* settings);
```

Функция принимает на вход строку, описывающую параметры системы. Строка представляет собой набор пар вида "КЛЮЧ=ЗНАЧЕНИЕ", разделённых точкой с запятой. Поддерживаются следующие параметры:

ini	Путь к файлу инициализации. В этом файле описывается конфигурацию библиотеки - режим журналирования и т.п. Задание параметра может выглядеть, например, так: "ini=conf/settings.ini", в этом случае библиотека будет загружать конфигурацию из файла conf/settings.ini Секция [p2syslog] задает настройку логирования p2. (см. "Настройка логов p2")
subsystems	Инициализируемые подсистемы p2, через запятую. mq - mq протокол; replclient - клиент p2 реплики; mqpwd - протокол смены пароля.
log	Способ логирования. log = std - выводить логи cgate в stdout; log = null - выводить в /dev/null; log = p2; log = p2:p2syslog - где p2syslog название секции с настройками логирования.
minloglevel	Минимальный уровень логов для вывода (по умолчанию debug).

Принимаемые значения: trace, debug, info, notice, warning, error, critical.

key — Идентификатор клиентского ПО. Должен быть указан для работы с библиотекой. Ключ используется для получения доступа в систему Plaza-2 - для тестовой системы существует набор предопределенных ключей; для production - ключ получается в результате прохождения процедуры сертификации ПО.

Ошибка инициализации может свидетельствовать об ошибке конфигурации: отсутствует конфигурационный файл, нарушена целостность установки и т.п. В случае такой ошибки нет смысла пытаться повторно инициализировать библиотеки; вместо этого следует остановить ваше ПО и проверить конфигурацию.

Если инициализация провалилась или не была выполнена, работа с другими функциями библиотеки невозможна.

Код инициализация системы может выглядеть следующим образом:

```
result = cg_env_open("ini=ini/settings.ini;key=72395823576");
if (result != CG_ERR_OK)
{
    // вывести сообщение об ошибке и произвести выход из программы
    ...
    return;
}
```

Что означает выполнение инициализации библиотеки с конфигурационным файлом ini/settings.ini и ключом приложения "72395823576". Файл ini/settings.ini должен быть доступен по указанному пути, относительно текущего рабочего каталога в момент запуска ПО.

Деинициализация выполняется перед выходом из программы вызовом функции env_close:

```
CG_RESULT cg_env_close(void);
```

Функция проводит деинициализацию подсистем и закрытие журнала работы. Следует всегда вызывать данную функцию в конце работы ПО.

2.4.1. Настройка логов p2

Параметры работы с логами задаются в секции [p2syslog] соответствующего ini-файла. В этой секции могут быть заданы следующие параметры (опции):

- **logfileperday** — задает порционность лог-файлов. Может принимать следующие значения:
  - 0 — новый лог-файл формируется каждый раз при запуске модуля (открытии лога). Старый лог-файл переименовывается путем добавления к его базовому имени "1". При этом в зависимости от значения параметра "logfilenametype" номер вставляется в середину названия файла (перед точкой), либо в конец названия (после расширения). При формировании очередного лог-файла номера предыдущих увеличиваются на единицу. Количество хранимых лог-файлов задается в параметре "logfiledepth".
  - 1 — новый лог-файл создается на каждый день, если в этот день была хотя бы одна запись в лог. Если при открытии лога необходимый лог-файл уже существует, то новые сообщения дописываются в конец этого файла. При первом открытии лога к базовому имени лог-файла добавляется дата его создания в формате YYYYMMDD. При этом в зависимости от значения параметра "logfilenametype" дата вставляется в середину названия файла (перед точкой), либо в конец названия (после расширения). Количество хранимых лог-файлов задается в параметре "logfiledepth".
  - 2 (значение по умолчанию) — новый лог-файл создается каждый час. Если при открытии лога необходимый лог-файл уже существует, то новые сообщения дописываются в конец этого файла. При первом открытии лога к базовому имени лог-файла добавляется дата и время его создания в формате YYYYMMDDHH. При этом в зависимости от значения параметра "logfilenametype" дата вставляется в середину названия файла (перед точкой), либо в конец названия (после расширения). При переходе через границу часа создается новый лог-файл с соответствующим именем. Количество хранимых лог-файлов задается в параметре "logfiledepth".
- **logfile** — базовое имя лога, используется для формирования имени лог-файла. Если параметр не задан или задано пустое имя, то используется имя файла исполняемого модуля без расширения. Представляет собой левую часть имени лог-файла, может быть задан полный или относительный путь. Если задан относительный путь, то полный путь расположения лог-файлов вычисляется относительно текущего каталога в момент открытия лога. Если задать строку logfile=nul, то лог-файл сформирован не будет.
- **logfilenametype** — способ формирования имени лог-файла. Может принимать следующие значения:
  - 0 — дополнительные параметры имени (номер или дата) добавляются в конец названия файла.
  - 1 (значение по умолчанию) — дополнительные параметры имени вставляются в середину названия файла.
- **logfiledepth** — количество хранимых лог-файлов, включая текущий. Значение должно быть больше нуля, значение по умолчанию: 168.
- **logtime** — определяет формат вывода времени. При наличии соответствующей поддержки в ОС вывод миллисекунд (3 знака) заменяется на вывод микросекунд (6 знаков). Сейчас эта возможность реализована только под Windows (функция API представлена в Win8+/Win Server 2012). Может принимать следующие значения:

- 0 (значение по умолчанию при logfileperday=0) — время не выводится.
- 1 — время с точностью до секунд выводится на отдельной строке перед соответствующим сообщением, если с момента предыдущего сообщения прошло не менее секунды. Формат даты: YYYY-MM-DD HH:MM:SS.
- 2 — время с точностью до тысячных долей секунды выводится в начале каждой строки с сообщением. Формат даты: YYYY-MM-DD HH:MM:SS.mmm.
- 3 (значение по умолчанию при logfileperday=1 или logfileperday=2) — время с точностью до тысячных долей секунды выводится в начале каждой строки с сообщением, но при этом календарная дата не выводится. Формат даты: HH:MM:SS.mmm. Опция может быть указана только для режимов "logfileperday=1" и "logfileperday=2".
- 4 — в начале каждой строки с сообщением выводится количество микросекунд, прошедших с момента инициализации библиотеки P2SysLog.

  ### Предупреждение

  Внимание! Сейчас это значение соответствует формату вывода со значением "5".

- 5 — в начале каждой строки с сообщением выводится количество микросекунд, прошедших с момента старта компьютера.
- 6 — время выводится как объединение режимов (3) и (5). Например: 13:13:58.627|00010475078718. Опция может быть указана только для режимов "logfileperday=1" и "logfileperday=2".

- **logtoconsole** — разрешает копирование всех лог-сообщений на консоль. Может принимать следующие значения:
  - 0 (default) — копирование запрещено.
  - 1 — копирование разрешено.
- **traceini** — задает имя ini-файла, куда необходимо заносить все трейсы. Если параметр traceini не задан, то все трейсы попадают в основной ini-файл приложения. Отдельный файл с трейсами позволяет присваивать основному ini-файлу свойство Read only, что невозможно в случае, когда трейсы заносятся в него же. Если в качестве имени ini-файла указан nul, то отдельный ini-файл с трейсами не формируется. В этом случае трейсы никуда не записываются и попадают в лог, только если выводятся посредством не P2TRACE*_0 макросов.
- **addthreadid** — при включении данной настройки к каждой строке лог-файла будет добавляться информация о потоке, который это выводит (ThreadID). Настройка актуальна для многопоточных приложений. Может принимать следующие значения:
  - 0 — выключено.
  - 1 (default) — включено.
- **logfilecache** — размер (в байтах) буфера в памяти процесса, в который кэшируется лог. Значение должно быть в диапазоне 64-32767, при выходе за пределы этого диапазона возвращается ошибка. Значение по умолчанию определяется установленной операционной системой. Опция используется для управления кэшированием в памяти операций записи в лог-файл. При задании данного параметра (использовании кэша) следует помнить, что файловая операция записи в лог-файл не активизируется пока не заполнится весь буфер. Поэтому слишком большой размер буфера может привести к ситуации, когда лог-файл будет пустым. В асинхронном режиме логирования буфер файла будет сброшен при отсутствии вызовов подсистемы логирования в течение 300 миллисекунд. При установке значения в 0 при включенном режиме асинхронного логирования выводится предупреждение и используется значение по умолчанию.
- **logasync** — при включении настройки запись в лог-файл будет производиться в отдельном потоке. Может принимать следующие значения:
  - 1 (default) — запись в лог-файл производится в отдельном потоке.
  - 0 — запись в лог-файл производится в общем потоке.
- **logthread_affinity_ex** — маска процессоров для потока асинхронных логов. Маска представляет собой строку чисел, разделенных запятой. Каждое число задает номер ядра, на котором может выполняться поток логов. Используется только при асинхронном логировании (logasync=1).

2.5. Соединение

Объект "Соединение" обеспечивает взаимодействие с роутером Plaza-2 для отправки и получения сообщений. Эти объекты могут создаваться в произвольном количестве в любое время работы ПО при инициализированном окружении; тем не менее, рекомендуется создавать соединения при старте ПО, а уничтожать - непосредственно перед выходом.

2.5.1. cg_conn_new

Создание соединения выполняется вызовом:

```
CG_RESULTcg_conn_new(const char* settings, cg_conn_t** connptr);
```

Параметрами являются строка инициализации соединения и указатель, в который будет занесен указатель на созданное соединение. Строка создания соединения задаётся в формате URL следующего вида: `"TYPE://HOST:PORT;param1=value1;param2=value;...;paramN=valueN"`, где

TYPE Тип соединения. В настоящее время поддерживаются три вида соединений:

- p2tcp — Соединение с роутером Plaza-2 посредством протокола TCP/IP. Медленнее, удобно для отладки, может связываться с роутером, установленным на другой машине
- p2lrpcq — Соединение с роутером Plaza-2 посредством разделяемой памяти. Быстрее, оптимально для production, работает исключительно в рамках одной машины.
- p2sys — Специальное соединение для управление роутером.

Примечание: В случае p2lrpcq соединение должно всегда закрываться корректно, т.к. в противном случае в роутере соединение не деинициализируется, поэтому повторное соединение к роутеру под этим же именем будет не возможно без перезапуска роутера.

HOST Адрес, с которым устанавливается соединение. В случае типа соединения p2tcp - это адрес машины, на которой запущен интересующий процесс P2MQRouter, в случае p2lrpcq - 127.0.0.1.

PORT Номер порта, по которому производится соединение. Должен быть указан как для p2tcp, так и для p2lrpcq; в последнем случае порт будет использован в качестве управляющего канала для установки соединения через разделяемую память.

Параметры допустимые при задании соединений p2tcp и p2lrpcq:

app_name — Имя приложения Plaza-2. В пределах одного роутера Plaza-2 каждое соединение с ним должно обладать уникальным именем. Этот идентификатор используется для маршрутизации сообщений в соответствующие обработчики.

local_pass — Пароль для соединения с роутером Plaza-2, если роутер сконфигурирован на проверку открываемых соединений паролем. При подключении к удалённому роутеру использование local_pass является обязательным.

timeout — Время в миллисекундах, в течение которого ожидается установка соединения с роутером в процессе вызова conn_open(...). В случае превышения времени ожидания соединения вызов conn_open(...) вернёт ошибку.

local_timeout — Время в миллисекундах, в течение которого ожидается ответ от роутера Plaza-2 при использовании соединения p2lrpcq.

name — имя объекта в cgate;

lrpcq_buf — Размер буфера lrpcq в байтах.

Пример вызова функции:

```
const char* conn_str = "p2lrpcq://127.0.0.1:4001;app_name=myapp";
cg_conn_t* conn;

result = cg_conn_new(conn_str, *conn);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to initialize connection: 0x%X\n", result);
    return;
}
```

Примеры строки соединения:

```
p2lrpcq://127.0.0.1:4001;app_name=example;timeout=2000;local_timeout=500;lrpcq_buf=0;name=p2lrpcq_example;
p2tcp://192.168.1.1:4003;app_name=example2;timeout=2000;local_pass=123;name=p2tcp_example;
p2sys://127.0.0.1:4001;app_name=example3;timeout=2000;name=p2sys_example;
```

Возвращаемые значения:

CG_ERR_OK — Успешное выполнение.

CG_ERR_INVALIDARGUMENT — Функции были переданы некорректные аргументы.

CG_ERR_INTERNAL — Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

2.5.2. cg_conn_open

Открытие соединения выполняется вызовом:

```
CG_RESULT cg_conn_open(cg_conn_t* conn, const char* settings);
```

Параметрами являются указатель на объект соединения и строка открытия соединения. Строка открытия соединения в настоящее время не используется и должна быть либо пустой, либо NULL.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENT Функции были переданы некорректные аргументы.

CG_ERR_INCORRECTSTATE Предпринята попытка открыть соединение в то время, когда оно не может быть открыто, т.к. либо уже активно, либо находится в состоянии ошибки

CG_ERR_UNSUPPORTED Возвращается при невозможности проверки пользовательского ключа, при этом в лог выводится сообщение "Certificate check failed".

CG_ERR_INTERNAL Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Важно

Возврат функцией значения CG_ERR_OK не означает, что соединение было успешно открыто - об этом факте можно судить только по изменению статуса соединения (cg_conn_getstate). Успешное исполнение данной функции означает, что процесс открытия соединения был начат успешно и через некоторое время соединение может перейти в состояние CG_STATE_ACTIVE в случае успеха или в состояние CG_STATE_ERROR в случае неудачи открытия соединения.

Рекомендуется делать паузу между попытками открытия соединения, например 1 секунду, чтобы уменьшить нагрузку на логовую систему и клиентский роутер при невозможности открытия соединения или ошибках при его открытии.

Передаваемые значения:

settings В настоящее время не используется. Зарезервировано для будущего использования.

Пример вызова функции:

```
cg_conn_t* conn; // указатель на инициализированный вызовом conn_new объект

result = cg_conn_open(conn, NULL);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to open connection: 0x%X\n", result);
    // Надо предпринять попытку повторного открытия соединения
}
```

2.5.3. cg_conn_close

Закрытие соединения выполняется вызовом:

```
CG_RESULT cg_conn_close(cg_conn_t* conn);
```

Параметром является указатель на объект соединения.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENT Функции были переданы некорректные аргументы.

CG_ERR_INCORRECTSTATE Предпринята попытка закрыть соединение в то время, когда оно закрыто.

CG_ERR_INTERNAL Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

После закрытия соединения, оно может быть повторно открыто вызовом cg_conn_open.

Пример вызова функции:

```
cg_conn_t* conn; // указатель на инициализированный вызовом conn_new объект

result = cg_conn_close(conn);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to close connection: 0x%X\n", result);
    return;
}
```

2.5.4. cg_conn_destroy

Уничтожение соединения выполняется вызовом:

```
CG_RESULT cg_conn_destroy(cg_conn_t* conn);
```

Параметром является указатель на объект соединения.

Возвращаемые значения:

CG_ERR_OK	Успешное выполнение.
CG_ERR_INVALIDARGUMENT	Функции были переданы некорректные аргументы.
CG_ERR_INCORRECTSTATE	Предпринята попытка уничтожить соединение в то время, когда оно не было корректно закрыто.
CG_ERR_INTERNAL	Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Данный вызов уничтожает объект, на который указывает параметр conn и освобождает все связанные с ним ресурсы. После вызова этой функции объект больше не может быть использован. Эта функция должна быть вызвана для каждого объекта, созданного вызовом cg_conn_new, вне зависимости от того, выполнялась работа (открытие, получение данных, отправка сообщений) с данным объектом или нет. При вызове cg_conn_destroy для активного conn внутри cgate происходит закрытие соединения, а потом его разрушение.

Пример вызова функции:

```
cg_conn_t* conn; // указатель на объект, который был закрыт вызовом conn_close

result = cg_conn_destroy(conn);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to destroy connection: 0x%X\n", result);
    return;
}
```

2.5.5. cg_conn_process

Обработка сообщений соединения выполняется вызовом:

```
CG_RESULT cg_conn_process(cg_conn_t* conn, uint32_t timeout, void* reserved);
```

Параметрами является указатель на объект соединения и время ожидания событий. Последний параметр reserved в настоящее время не используется и должен быть равен NULL.

Возвращаемые значения:

CG_ERR_OK	Успешное выполнение.
CG_ERR_TIMEOUT	За указанное время не было обработано ни одного события системы
CG_ERR_INVALIDARGUMENT	Функции были переданы некорректные аргументы.
CG_ERR_INTERNAL	Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Данный вызов осуществляет итерацию работы с соединением, которая включает в себя опрос очереди входящих сообщений, анализ пришедших данных, вызов пользовательских функций обратного вызова. Эта функция должна вызываться из кода пользователя с частотой, соответствующей максимальной желаемой частоте получения данных.

> **Важно**
>
> Пользовательские функции обратного вызова для подписчиков, привязанных к данному соединению будут вызваны в процессе работы этой функции из того же потока исполнения.

В случае, если значения параметра timeout отличается от 0, вызов будет заблокирован на timeout миллисекунд в ожидании событий. Если в момент вызова функции очередь входящих сообщений не пуста, функция не будет ничего ожидать и сразу же перейдёт к обработке входящих сообщений. Если функцию не вызывать дольше трёх репликационных пингов (3*repl_ping (значение repl_ping по умолчанию = 10 секунд)), то возникает timeout подписчика, а потом timeout соединения, после чего соединения и подписчики закрываются.

Пример вызова функции:

```
cg_conn_t* conn; // указатель на соединение в активном состоянии
```

```
// разбираем входящие сообщения в цикле, но не более 100 за итерацию
for (int callidx = 0; callidx < 100; ++ callidx)
{
    result = cg_conn_process(conn, 0, NULL);

    if (result == CG_ERR_TIMEOUT) // сообщений нет
        break; // перейти к дальнейшей логике работы ПО
    else
    if (result != CG_ERR_OK)
    {
        // попытка обработки соединения завершилась ошибкой
        // вывести сообщение и закрыть соединение
        fprintf(stderr, "Failed to process connection: 0x%X\n", result);
        result = cg_conn_close(conn); //
        if (result != CG_ERR_OK)
        {
            // закрытие соединения провалилось, выйти из программы
            fprintf(stderr, "Failed to close connection: 0x%X\n", result);
            return;
        }
        break;
    }
}
```

2.5.6. cg_conn_getstate

Получение статуса соединения выполняется вызовом:

```
CG_RESULT cg_conn_getstate(cg_conn_t* conn, uint32_t* state);
```

Параметрами является указатель на объект соединения и указатель на значение размером 4 байта, куда будет записан текущий статус соединения.

Возвращаемые значения:

CG_ERR_OK	Успешное выполнение.
CG_ERR_INVALIDARGUMENT	Функции были переданы некорректные аргументы.
CG_ERR_INTERNAL	Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Данный вызов должен быть использован для периодического получения статуса соединения, для того, чтобы можно было выполнить действия, связанные с переходом объекта подписки в разные состояния - например, выполнить закрытие соединения, если он перешла в состояние ошибки. Более подробно статусы объектов описаны в разделе Жизненный цикл объектов.

Эта функция доступна для вызова в любое время между вызовами cg_conn_new и cg_conn_destroy.

Пример вызова функции:

```
cg_connection_t* conn; // указатель на объект соединение
uint32_t state; // Сюда будет записан статус

result = cg_conn_getstate(conn, &state);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to query connection state: 0x%X\n", result);
    return;
}

switch (state)
{
    case CG_STATE_ERROR: /* ... */
    case CG_STATE_CLOSED: /* ... */
}
```

2.6. Подписчик

Объект "Подписчик" обеспечивает получение сообщений через указанное соединение. Правила, по которым получаются сообщения зависят от типа подписчика - это могут быть как сообщения peer-to-peer, так и сообщения publish-subscribe, такие как реплика.

Работами с объектами "Подписчик" в API производятся посредством указателя cg_listener_t*.

2.6.1. cg_lsn_new

Создание подписчика выполняется вызовом:

```
CG_RESULT cg_lsn_new(cg_conn_t* conn, const char* settings, CG_LISTENER_CB callback, void* data,
cg_listener_t** lsnptr);
```

Параметрами являются: указатель на инициализированный объект соединения, в котором создаётся подписчик, строка инициализации подписчика, указатель на функцию обратного вызова, которая будет вызываться при возникновении событий, произвольный указатель, который будет передаваться в функцию обратного вызова и указатель, в который будет занесен указатель на созданный подписчик. Строка создания соединения задаётся в формате URL следующего вида: "TYPE://[STREAM][;param1=value1[;param2=value[;...[;paramN=valueN]]]]", где

TYPE Тип подписки. Поддерживаются следующие типы подписки:

p2repl	Получение табличного потока репликации Plaza-2
p2mqreply	Получение ответов на ранее отправленные сообщения
p2ordbook	Получение активных заявок с использованием срезов стаканов для начальной синхронизации, а затем переход на он-лайн поток
p2sys	Получение статуса: соединения, роутера, авторизации.

Остальные параметры зависят от типа подписки.

Параметры, поддерживаемые типом подписки *p2repl*:

STREAM	Задаёт имя потока табличной репликации.
Параметр "scheme"	Путь к используемой схемы данных потока. См. Схемы данных.
Параметр "tables"	Список требуемых таблиц из серверной схемы, разделенных запятой. Не допускается одновременное задание параметров "tables" и "scheme"

Параметры, поддерживаемые типом подписки *p2ordbook*:

STREAM	Задаёт имя он-лайн потока с журналом действий над заявками (FORTS_TRADE_REPL)
Параметр "snapshot"	Имя потока со срезами активных заявок (FORTS_USERORDERBOOK_REPL).
Параметр "online.scheme"	Путь к используемой схемы данных он-лайн потока. Схема должна содержать таблицу orders_log.
Параметр "snapshot.scheme"	Путь к используемой схемы данных снапшот потока. Схема должна содержать таблицы orders и info.
Параметр "online.data"	Имя таблицы с заявками для online потока. Значение по умолчанию "orders_log".
Параметр "snapshot.data"	Имя таблицы с заявками для snapshot потока. Значение по умолчанию "orders".
Параметр "snapshot.bind"	Имя таблицы и поля с ревизией, по которому осуществляется сшивка потоков. Значение по умолчанию "info.logRev".

Параметры, поддерживаемые типом подписки *p2mqreply*:

Параметр "ref"	Содержит имя публикатора, который был использован для отправки сообщений, ответы на которые требуется получать в данном подписчике. На момент создания подписчика p2mqreply должен существовать и связанный с ним по ref публикатор.

Подписчик "p2mqreply" использует в качестве схемы данных схему, заданную в связанном публикаторе. Именна эта схема данных будет возвращена вызовом cg_lsn_getscheme.

Подписчик "p2ordbook" использует в качестве схемы объединенную схему срезов и он-лайн потоков. При этом данные в момент прихода среза будут соответствовать сообщениям из схемы со срезами данных, а после перехода в онлайн будут приходить сообщения из он-лайн схемы. При использовании статических структур данных для работы с сообщениями данного потока, нужно иметь описания структур как для срезов, так и для он-лайн данных, причём следует обращать внимания на индексы соответствующих таблиц. При использовании динамического подхода к работе со схемами, следует применять стандартные практики - запомнить номера интересующих сообщений и полей и использовать их во время прихода данных.

Параметр callback функции указывает на пользовательскую функцию обратного вызова, которая имеет следующий вид:

```
CG_RESULT callback(cg_conn_t* conn, cg_listener_t* listener, struct cg_msg_t* msg, void* data);
```

Эта функция вызывается в момент возникновения какого либо события по данной подписке: открытие подписки, закрытие, приход сообщения и т.п. В качестве параметров функция обратного вызова получает указатель на соединение, в котором создана подписка,

указатель на объект подписки, в котором возникло событие, указатель на сообщение и пользовательский указатель data, который был передан в вызов cg_lsn_new. Код возврата пользовательского обработчика должен быть установлен в 0 в случае успеха обработки сообщения или в другое значение в случае ошибки. В случае возврата ошибки из callback подписчик будет закрыт, и в callback придут последовательно сообщения: replstate - с состоянием подписчика на последний CG_MSG_TN_COMMIT, CG_MSG_CLOSE.

Важно

Вызов функции cg_lsn_new выполняет только инициализацию объекта подписки, но не приводит к фактическому началу получения данных; для начала получения данных необходимо перевести подписку в активное состояние вызовом cg_lsn_open.

В пользовательский callback могут приходить следующие сообщения:

Тип подписчика	Тип сообщения	Описание
p2repl, p2mqreply, p2sys	CG_MSG_OPEN	Сообщение приходит в момент активации потока данных. Это событие гарантированно возникает до прихода каких либо данных по данной подписке. Для потоков репликации приход сообщения означает, что схема данных согласована и готова для использования (Подробнее см. Схемы данных) Данное сообщение не содержит дополнительных данных и его поля data и data_size не используются.
p2repl, p2mqreply, p2sys	CG_MSG_CLOSE	Сообщение приходит в момент закрытия потока данных. Приход сообщения означает, что поток был закрыт пользователем или системой. В поле data содержится указатель на int, по указанному адресу хранится информация о причине закрытия подписчика. Возможны следующие причины: • CG_REASON_UNDEFINED - не определена. • CG_REASON_USER - пользователь вернул ошибку в callback слушателя. • CG_REASON_ERROR - внутренняя ошибка. • CG_REASON_DONE - вызван метод cg_lsn_destroy. • CG_REASON_SNAPSHOT_DONE - снэпшот получен.
p2repl	CG_MSG_TN_BEGIN	Означает момент начала получения очередного блока данных. В паре со следующим сообщением может быть использовано логикой ПО для контроля целостности данных. Данное сообщение не содержит дополнительных данных и его поля data и data_size не используются.
p2repl	CG_MSG_TN_COMMIT	Означает момент завершения получения очередного блока данных. К моменту прихода этого сообщения можно считать, что данные полученные по данной подписке, находятся в непротиворечивом состоянии и отражают таблицы в синхронизированном между собой состоянии. Данное сообщение не содержит дополнительных данных и его поля data и data_size не используются.
p2repl	CG_MSG_STREAM_DATA	Сообщение прихода потоковых данных. Поле data_size содержит размер полученных данных, data указывает на сами данные. Само сообщение содержит дополнительные поля, которые описываются структурой cg_msg_streamdata_t. Подробнее о получении данных см. раздел Получение потоков репликации
p2repl	CG_MSG_P2REPL_ONLINE	Переход потока в состояние online - это означает, что получение начального среза было завершено и следующие сообщения CG_MSG_P2REPL_DATA будут нести данные он-лайн. Данное сообщение не содержит дополнительных данных и его поля data и data_size не используются.
p2repl	CG_MSG_P2REPL_LIFENUM	Изменен номер жизни схемы. Такое сообщение означает, что предыдущие данные, полученные по потоку, не актуальны и должны быть очищены. При этом произойдёт повторная трансляция данных по новому номеру жизни схемы данных. Поле data сообщения указывает на целочисленное значение, содержащее новый номер жизни схемы; поле data_size содержит размер целочисленного типа.
p2repl	CG_MSG_P2REPL_CLEARDELETED	Произошла операция массового удаления устаревших данных. Поле data сообщения указывает на структуру cg_data_cleardeleted_t, в которой указан номер таблицы и номер ревизии, до которой данные в указанной таблице считаются удаленными. Если ревизия в cg_data_cleardeleted_t == CG_MAX_REVISON, то последующие ревизии продолжатся с 1.
p2repl	CG_MSG_P2REPL_REPLSTATE	Сообщение содержит состояние потока данных; присылается перед закрытием потока. Поле data сообщения указывает на строку, которая в закодированном виде содержит состояние потока данных на момент прихода сообщения - сохраняются схема данных, номера ревизий таблиц и номер жизни схемы. Эта строка может быть передана в вызов cg_lsn_open в качестве параметра

Тип подписчика	Тип сообщения	Описание
		"replstate" по этому же потоку в следующий раз, что обеспечит продолжение получения данных с момента остановки потока.
p2mqreply, p2sys	CG_MSG_DATA	Сообщение содержит ответ на ранее отосланное состояние. Поле data указывает на данные, а поле data_size содержит размер блока данных. Сообщение описывается структурой cg_msg_data_t и содержит дополнительные поля, позволяющие идентифицировать исходное сообщение, а также информацию о схеме данных. Подробнее см. раздел Отправка команд и получение ответов.
p2mqreply	CG_MSG_P2MQ_TIMEOUT	Сообщение приходит в том случае, если ответ на отправленное ранее сообщение не был получен в течение указанного в соответствующем publisher времени. Сообщение описывается структурой cg_msg_data_t и содержит значение user_id, задаваемое при отправке исходного сообщения.

Примеры url:

- p2sys://;name=p2sys_lsn
- p2repl://FORTS_REFDATA_REPL;name=repl_sample1
- p2repl://FORTS_ORDLOG_REPL;scheme=|FILE|./ini/ordlog_repl.ini|scheme;name=repl_sample2
- p2ordbook://FORTS_ORDLOG_REPL;snapshot=FORTS_ORDBOOK_REPL;name=ordbook_sample1
- p2ordbook://FORTS_TRADE_REPL;snapshot=FORTS_USERORDERBOOK_REPL;
  online.scheme=|FILE|ini/futtrade.ini|FutTrade;name=ordbook_sample2
- p2ordbook://FORTS_TRADE_REPL;snapshot=FORTS_USERORDERBOOK_REPL;
  online.scheme=|FILE|ini/futtrade.ini|FutTrade;snapshot.scheme=|FILE|ini/orderbook.ini|CustReplScheme;name=ordbook_sample3
- p2mqreply://;ref=pub_name;name=mqreply_sample

Пример создания подписчика на получение потока репликации:

```
cg_conn_t* conn; // указатель на инициализированный объект "Соединение"

const char* lsn_str = "p2repl://FORTS_REFDATA_REPL";
cg_listener_t* lsn;

result = cg_lsn_new(conn, lsn_str, callback, 0, *lsn);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to initialize listener: 0x%X\n", result);
    return;
}
```

Пример создания подписчика на получение ответов на отправленные команды:

```
cg_conn_t* conn; // указатель на инициализированный объект "Соединение"

// строка инициализации публикатора отправки команд
// указан параметр name=TN1
const char* pub_str = "p2mq://FORTS_SRV;category=FORTS_MSG;name=TN1";
cg_publisher_t* pub;

// строка инициализации подписчика получения ответов
// указан параметр ref=TN1, который обеспечивает связь с публикатором
const char* lsn_str = "p2mqreply://;ref=TN1";
cg_listener_t* lsn;

result = cg_lsn_new(conn, lsn_str, callback, 0, *lsn);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to initialize listener: 0x%X\n", result);
    return;
}
```

2.6.2. cg_lsn_open

Открытие подписки выполняется вызовом:

```
CG_RESULT cg_lsn_open(cg_listener_t* lsn, const char* settings);
```

Параметрами являются указатель на объект подписки и строка открытия подписки. Строка параметров открытия задаётся в формате "param1=value1;param2=value2;....;paramN=valueN", причем названия и значения параметров зависят от типа подписки. Параметр settings можно задавать только в p2repl-листенера, так как он больше нигде не используется.

Параметры открытия подписки *p2repl*:

mode — Определяет режим получения данных и может принимать следующие значения:

- snapshot — Поток открывается в режиме получения среза данных. При этом данные в режиме онлайн транслироваться не будут
- online — Поток открывается в режиме получения онлайн-данных. Срез данных получен не будет, данные будут идти с момента открытия потока
- snapshot+online — Поток открывается в режиме получения снапшота, а затем перехода в режим получения он-лайн.

replstate — Задаёт состояние потока, с которого следует произвести открытие. Значение этого параметра должно соответствовать строке, полученной в сообщении CG_MSG_P2REPL_REPLSTATE в момент предыдущего закрытия потока.

lifenum — Задаёт номер жизни схемы. Данный параметр может быть использован для подключения к потоку данных, если по каким либо причинам возможности, предоставляемые параметром "replstate" не подходят. Если задан параметр "replstate", то значение данного параметра будет проигнорировано.

rev.TABLE_NAME — Задаёт начальную ревизию таблицы TABLE_NAME. Вместо TABLE_NAME следует подставить имя интересующей таблицы. Данный параметр может быть использован для подключения к потоку данных, если по каким либо причинам возможности, предоставляемые параметром "replstate" не подходят. Если задан параметр "replstate", то значение данного параметра будет проигнорировано. Возможно указание этого параметра несколько раз для разных таблиц в потоке, например "rev.orders_log=234445;rev.deal=55". Запрещено задавать ревизию таблицы TABLE_NAME без указания lifenum=%d (ревизия без номера жизни не имеет смысла).

Параметры "replstate" и ("lifenum" + "rev.TABLE_NAME") являются взаимоисключающими. Также запрещено одновременно задавать значения параметров "replstate" и "rev.TABLE_NAME". Таким образом, возможны только следующие комбинации параметров:

- replstate
- lifenum
- lifenum + rev.TABLE_NAME

Возвращаемые значения:

CG_ERR_OK — Успешное выполнение.

CG_ERR_INVALIDARGUMENT — Функции были переданы некорректные аргументы.

CG_ERR_INCORRECTSTATE — Предпринята попытка открыть подписку в то время, когда она не может быть открыта, т.к. либо уже активна, либо находится в состоянии ошибки.

CG_ERR_INTERNAL — Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Важно

Возврат функцией значения CG_ERR_OK не означает, что подписка была успешно открыта - об этом факте можно судить только по изменению статуса подписки (cg_lsn_getstate). Успешное исполнение данной функции означает, что процесс открытия подписки был начат успешно и через некоторое время подписка может перейти в состояние CG_STATE_ACTIVE в случае успеха или в состояние CG_STATE_ERROR в случае неудачи открытия.

Рекомендуется делать паузу между попытками открытия подписки, например 1 секунду, чтобы уменьшить нагрузку на логовую систему при невозможности открытия подписки.

Пример вызова функции:

```
cg_listener_t* lsn; // указатель на инициализированный вызовом cg_lsn_new объект
const char* lsn_open_str = "mode=online";

result = cg_lsn_open(lsn, lsn_open_str);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to open listener: 0x%X\n", result);
```

```
    // Надо предпринять попытку повторного открытия подписчика
}
```

2.6.3. cg_lsn_close

Закрытие подписки выполняется вызовом:

```
CG_RESULT cg_lsn_close(cg_listener_t* lsn);
```

Параметром является указатель на объект подписчика.

Возвращаемые значения:

CG_ERR_OK	Успешное выполнение.
CG_ERR_INVALIDARGUMENT	Функции были переданы некорректные аргументы.
CG_ERR_INCORRECTSTATE	Предпринята попытка закрыть подписчика в то время, когда окружение было некорректно инициализировано.
CG_ERR_INTERNAL	Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

После закрытия подписчика, он может быть повторно открыт вызовом cg_lsn_open.

Пример вызова функции:

```
cg_listener_t* lsn; // указатель на открытую подиску

result = cg_lsn_close(lsn);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to close listener: 0x%X\n", result);
    return;
}
```

2.6.4. cg_lsn_destroy

Уничтожение подписчика выполняется вызовом:

```
CG_RESULT cg_lsn_destroy(cg_listener_t* lsn);
```

Параметром является указатель на объект подписки.

Возвращаемые значения:

CG_ERR_OK	Успешное выполнение.
CG_ERR_INVALIDARGUMENT	Функции были переданы некорректные аргументы.
CG_ERR_INCORRECTSTATE	Предпринята попытка уничтожить подписчика в то время, когда когда окружение было некорректно инициализировано.
CG_ERR_INTERNAL	Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Данный вызов уничтожает объект, на который указывает параметр lsn и освобождает все связанные с ним ресурсы. После вызова этой функции объект больше не может быть использован. Эта функция должна быть вызвана для каждого объекта, созданного вызовом cg_lsn_new, вне зависимости от того, выполнялась работа (открытие, получение данных, отправка сообщений) с данным объектом или нет.

При разрушении активного подписчика cgate закрывает подписчика, после чего производит его разрушение.

Пример вызова функции:

```
cg_listener_t* lsn; // указатель на объект, который был закрыт вызовом cg_lsn_close

result = cg_lsn_destroy(lsn);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to destroy listener: 0x%X\n", result);
    return;
```

```
}
```

2.6.5. cg_lsn_getstate

Получение статуса подписчика выполняется вызовом:

```
CG_RESULT cg_lsn_getstate(cg_listener_t* lsn, uint32_t* state);
```

Параметрами является указатель на объект подписки и указатель на значение размером 4 байта, куда будет записан текущий статус подписчика.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы.

CG_ERR_INTERNAL Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Данный вызов должен быть использован для периодического получения статуса подписчика, для того, чтобы можно было выполнить действия, связанные с переходом объекта подписки в разные состояния - например, выполнить закрытие подписки, если она перешла в состояние ошибки. Более подробно статуса объектов описаны в разделе Жизненный цикл объектов.

Пример вызова функции:

```
cg_listener_t* lsn; // указатель на объект подписки
uint32_t state; // Сюда будет записан статус

result = cg_lsn_getstate(lsn, &state);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to query listener state: 0x%X\n", result);
    return;
}

switch (state)
{
    case CG_STATE_ERROR: /* ... */
    case CG_STATE_CLOSED: /* ... */
}
```

2.6.6. cg_lsn_getscheme

Получение схемы подписчика выполняется вызовом:

```
CG_RESULT cg_lsn_getscheme(cg_listener_t* lsn, cg_scheme_desc_t** schemeptr);
```

Параметрами является указатель на объект подписки и указатель на переменную, в которую будет записан указатель на описание схемы.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы.

CG_ERR_INTERNAL Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Вызов используется для получения схемы данных объекта подписки (подробнее см. раздел Работа со схемами данных). Схема данных доступна для получения с момента прихода события OPEN для подписки. В случае, если при создании подписки схема данных не была явно задана, между двумя сессиями работы с подпиской схема может измениться, т.е. в общем случае нельзя рассчитывать на то, что в цепочке вызовов open/close, open/close схема после первого open будет аналогична схеме после вызова второго open. Это может решаться либо указанием клиентской схемы данных, в тех случаях, когда это поддерживается типом подписки, либо анализом схемы каждый раз в момент прихода события OPEN.

Пример вызова функции:

```
cg_listener_t* lsn; // указатель на объект подписки
cg_scheme_desc_t* schemedesc; // Сюда будет записан указатель на описание схемы
```

```
result = cg_lsn_getscheme(lsn, &schemedesc);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to query listener scheme: 0x%X\n", result);
    return;
}

// напечатать кол-во сообщений в схеме
printf("Number of messages: %d\n", schemedesc->num_messages);
```

2.7. Публикатор

Объект "Публикатор" обеспечивает отправку сообщений через указанное соединение. Правила, по которым отправляются сообщения зависят от типа публикатора и соединения.

Работами с объектами "Публикатор" в API производятся посредством указателя cg_publisher_t*.

2.7.1. cg_pub_new

Создание подписчика выполняется вызовом:

```
CG_RESULTcg_pub_new(cg_conn_t* conn, const char* settings, cg_publisher_t** pubptr);
```

Параметрами являются: указатель на инициализированный объект соединения, в котором создаётся публикатор, строка инициализации публикатора и указатель, в который будет занесен указатель на созданный подписчик. Строка создания соединения задаётся в формате URL следующего вида: `"TYPE://[NAME][;param1=value1[;param2=value[;...[;paramN=valueN]]]]"`, где

TYPE — Тип публикатора. Поддерживаются следующие типы:

- p2mq — Отправка произвольных сообщений Plaza-2
- p2sys — Отправка запросов аутентификации или выхода из системы. Подробнее см. "Объекты протокола p2sys"

Параметр "name" — Определяет уникальное имя публикатора. Может быть использован для связи между парными публикаторами и подписчиками (например, публикатора mq и подписчика mqreply).

Остальные параметры зависят от типа подписки.

Параметры, поддерживаемые типом *публикатора p2mq*:

NAME — Задаёт имя сервиса, на который будут отправляться сообщения через данный публикатор.

Параметр "scheme" — Путь к используемой схемы данных. См. Схемы данных. Используемая схема данных должна содержать описания запросов и ответов - связанный подписчк p2mqreply будет использовать эту схему данных при разборе сообщений.

Параметр "category" — Категория отправляемых сообщений. Для отправки команд в торговую систему FORTS данный параметр должен быть зафиксирован как "FORTS_MSG"

Параметр "timeout" — Время ожидания ответа на отправленное сообщение в миллисекундах.

Параметр "version" — Версия схемы данных. Применяется только для схем из репозитория.

.

Важно

Вызов функции cg_pub_new выполняет только инициализацию объекта публикатора, но не приводит к фактическому разрешению отправки сообщений; для начала отправки сообщений необходимо перевести публикатор в активное состояние вызовом cg_pub_open.

Пример создания подписчика на отправку данных в торговую систему:

```
cg_conn_t* conn; // указатель на инициализированный объект "Соединение"

const char* pub_str = "p2mq://FORTS_SRV;category=FORTS_MSG;name=TN1";
cg_publusher_t* pub;

result = cg_pub_new(conn, pub_str, *pub);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to initialize publisher: 0x%X\n", result);
```

```
    return;
}
```

Пример того, как можно получить ответы на отправленные через публикатор команды, можно найти в разделе описания функции cg_lsn_new.

2.7.2. cg_pub_open

Открытие публикатора выполняется вызовом:

```
CG_RESULT cg_pub_open(cg_publisher_t* pub, const char* settings);
```

Параметрами являются указатель на объект публикатора и строка открытия. В настоящий момент публикаторы не требуют задания строки параметров и этот параметр должен быть NULL или пустой строкой.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы.

CG_ERR_INCORRECTSTATE Предпринята попытка открыть публикатор в то время, когда он не может быть открыт, т.к. либо уже активен, либо находится в состоянии ошибки

CG_ERR_INTERNAL Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Важно

Возврат функцией значения CG_ERR_OK не означает, что публикатор была успешно открыт - об этом факте можно судить только по изменению статуса публикатора (cg_pub_getstate). Успешное исполнение данной функции означает, что процесс открытия публикатора был начат успешно и через некоторое время подписка может перейти в состояние CG_STATE_ACTIVE в случае успеха или в состояние CG_STATE_ERROR в случае неудачи открытия.

Рекомендуется делать паузу между попытками открытия публикатора, например 1 секунду, чтобы уменьшить нагрузку на логовую систему при невозможности открытия публикатора.

Пример вызова функции:

```
cg_publisher_t* pub; // указатель на инициализированный вызовом cg_pub_new объект

result = cg_pub_open(pub, 0);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to open publisher: 0x%X\n", result);
    // Надо предпринять попытку повторного открытия публикатора
}
```

2.7.3. cg_pub_close

Закрытие публикатора выполняется вызовом:

```
CG_RESULT cg_pub_close(cg_publisher_t* pub);
```

Параметром является указатель на объект публикатора.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы.

CG_ERR_INCORRECTSTATE Предпринята попытка закрыть публикатор в то время когда окружение некорректно инициализировано.

CG_ERR_INTERNAL Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

После закрытия подписчика, он может быть повторно открыт вызовом cg_pub_open.

Пример вызова функции:

```
```

```
cg_publisher_t* pub; // указатель на открытый публикатора

result = cg_pub_close(pub);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to close publisher: 0x%X\n", result);
    return;
}
```

2.7.4. cg_pub_destroy

Уничтожение публикатора выполняется вызовом:

```
CG_RESULT cg_pub_destroy(cg_publisher_t* pub);
```

Параметром является указатель на объект публикатор.

Возвращаемые значения:

CG_ERR_OK	Успешное выполнение.
CG_ERR_INVALIDARGUMENT	Функции были переданы некорректные аргументы.
CG_ERR_INCORRECTSTATE	Предпринята попытка уничтожить публикатор в то время, когда окружение не было корректно инициализировано.
CG_ERR_INTERNAL	Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Данный вызов уничтожает объект, на который указывает параметр pub и освобождает все связанные с ним ресурсы. После вызова этой функции объект больше не может быть использован. Эта функция должна быть вызвана для каждого объекта, созданного вызовом cg_pub_new, вне зависимости от того, выполнялась работа (открытие, отправка сообщений) с данным объектом или нет.

При вызове cg_pub_destroy для активного публикатора, сперва происходит закрытие публикатора, а потом его разрушение.

Примеры:

```
p2sys://;name=p2sys_pub
p2mq://FORTS_SRV;category=FORTS_MSG;name=srvlink;timeout=5000;scheme=|FILE|forts_messages.ini|message
```

Пример вызова функции:

```
cg_publisher_t* pub; // указатель на объект, который был закрыт вызовом cg_pub_close

result = cg_pub_destroy(pub);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to destroy publisher: 0x%X\n", result);
    return;
}
```

2.7.5. cg_pub_getstate

Получение статуса публикатора выполняется вызовом:

```
CG_RESULT cg_pub_getstate(cg_publisher_t* pub, uint32_t* state);
```

Параметрами является указатель на объект публикатора и указатель на значение размером 4 байта, куда будет записан текущий статус публикатора.

Возвращаемые значения:

CG_ERR_OK	Успешное выполнение.
CG_ERR_INVALIDARGUMENT	Функции были переданы некорректные аргументы.
CG_ERR_INTERNAL	Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Данный вызов должен быть использован для периодического получения статуса публикатора, для того, чтобы можно было выполнить действия, связанные с переходом объекта подписки в разные состояния - например, выполнить закрытие публикатора, если он перешла в состояние ошибки. Более подробно статусы объектов описаны в разделе Жизненный цикл объектов.

Эта функция доступна для вызова в любое время между вызовами cg_pub_new и cg_pub_destroy.

Пример вызова функции:

```
cg_publisher_t* pub; // указатель на объект публикатор
uint32_t state; // Сюда будет записан статус

result = cg_pub_getstate(pub, &state);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to query publisher state: 0x%X\n", result);
    return;
}

switch (state)
{
    case CG_STATE_ERROR: /* ... */
    case CG_STATE_CLOSED: /* ... */
}
```

2.7.6. cg_pub_getscheme

Получение схемы публикатора выполняется вызовом:

```
CG_RESULT cg_pub_getscheme(cg_publisher_t* pub, cg_scheme_desc_t** schemeptr);
```

Параметрами является указатель на объект публикатора и указатель на переменную, в которую будет записан указатель на описание схемы.

Возвращаемые значения:

CG_ERR_OK	Успешное выполнение.
CG_ERR_INVALIDARGUMENT	Функции были переданы некорректные аргументы.
CG_ERR_INTERNAL	Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Вызов используется для получения схемы данных объекта подписки (подробнее см. раздел Работа со схемами данных). Схема данных доступна с момента перехода публикатора в состояние ACTIVE. В случае, если при создании публикатора схема данных не была явно задана, между двумя сессиями работы с подпиской схема может измениться, т.е. в общем случае нельзя рассчитывать на то, что в цепочке вызовов open/close, open/close схема после первого open будет аналогична схеме после вызова второго open. Это может решаться либо указанием клиентской схемы данных, в тех случаях, когда это поддерживается типом подписки, либо анализом схемы каждый раз в момент прихода события OPEN.

Пример вызова функции:

```
cg_publisher_t* pub; // указатель на объект публикатор
cg_scheme_desc_t* schemedesc; // Сюда будет записан указатель на описание схемы

result = cg_pub_getscheme(pub, &schemedesc);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to query publisher scheme: 0x%X\n", result);
    return;
}

// напечатать кол-во сообщений в схеме
printf("Number of messages: %d\n", schemedesc->num_messages);
```

2.7.7. cg_pub_msgnew

Создание нового сообщения для отправки выполняется вызовом:

```
CG_RESULT cg_pub_msgnew(cg_publisher_t* pub, uint32_t id_type, const void* id, struct cg_msg_t**
msgptr);
```

Параметрами является указатель на объект публикатора, тип ключа сообщения, указатель на значения ключа сообщения и указатель на переменную, в которую будет записан указатель на созданное сообщение.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы.

CG_ERR_INTERNAL Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Данный вызов инициализирует сообщение для отправки через данный публикатор. Желаемое сообщение идентифицируется типом и значением ключа в схеме данных публикатора. Поддерживаются следующие виды ключей:

CG_KEY_INDEX Ключом является номер сообщения в схеме. Параметр id указывает на значение типа uint32_t, в котором хранится желаемый номер сообщения

CG_KEY_ID Ключом является уникальный числовой идентификатор сообщения в схеме. Параметр id указывает на значение типа uint32_t, в котором хранится желаемый идентификатор сообщения

CG_KEY_NAME Ключом является имя сообщения в схеме. Параметр id указывает на строку, в которой записано имя желаемого сообщения. Строка должна завершаться нулём.

Сообщение, создаваемое данной функцией, является сообщением типа CG_MSG_DATA и описывается расширенной структурой:

```
struct cg_msg_data_t
{
    // Тип сообщения. Всегда CG_MSG_DATA для данного сообщения
    uint32_t type;
    // Размер данных
    size_t data_size;
    // Указатель на данные
    void* data;

    // Номер описания сообщения в активной схеме
    size_t msg_index;
    // Уникальный идентификатор типа сообщения
    uint32_t msg_id;
    // Имя сообщения в активной схеме
    const char* msg_name;

    // Пользовательский номер сообщения
    uint32_t user_id;
    // Адрес противоположной стороны
    const char* addr;
    // Указатель на связанное сообщение
    struct cg_msg_data_t* ref_msg;
};
```

Поле data_size содержит размер выделенного блока памяти для запрошенного формата сообщения, а поле data указывает на этот блок памяти. Поля msg_index, msg_id и msg_name заполнены данными в соответствие с используемой схемой данных. Поле user_id может быть использовано для задания пользовательского номера сообщения - этот же user_id будет указан в ответном сообщении, что позволяет связать запрос и ответ.

Пользовательский код должен сверить размер блока в поле data_size, выделенного для сообщения со своими ожиданиями относительно размера этого блока с целью избежать ошибок с заполнением сообщения. Затем следует заполнить блок по указателю data данными. После этого сообщение готово к отправке.

Пример вызова функции:

```
cg_publisher_t* pub; // указатель на объект публикатор
cg_msg_data* msg;

result = cg_pub_msgnew(pub, CG_KEY_NAME, "DelOrder", &msg);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to allocate message: 0x%X\n", result);
}
else
{
    DelOrder* delord;
    if (msg->data_size != sizeof(*delord))
    {
        fprintf(stderr, "Block sizes do not match: %d expected, but got %d \n",
                sizeof(*delord), msg->data_size);
    }
    else
    {
```

```
        delord = (DelOrder*)msg->data;
        delord->order_id = ...; // номер удаляемой заявки

        result = cg_pub_post(pub, msg, CG_PUB_NEEDREPLY);
        if (result != CG_ERR_OK)
        {
            fprintf(stderr, "Failed to post message: 0x%X\n", result);
        }
    }
}
```

2.7.8. cg_pub_post

Отправка сообщения выполняется вызовом:

```
CG_RESULT cg_pub_post(cg_publisher_t* pub, struct cg_msg_t* msg, uint32_t flags);
```

Параметрами является указатель на объект публикатора, указатель на сообщение и флаги отправки сообщения.

Возвращаемые значения:

CG_ERR_OK	Успешное выполнение.
CG_ERR_INVALIDARGUMENT	Функции были переданы некорректные аргументы.
CG_ERR_INCORRECTSTATE	Предпринята попытка отправить сообщение в то время, как соединение не активно.
CG_ERR_INTERNAL	Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Вызов предпринимает попытку отправки сообщения. Сообщение для отправки должно быть предварительно инициализировано вызовом cg_pub_msgnew и заполнено данными пользователя. В качестве флага обязательно следует указывать значение CG_PUB_NEEDREPLY, что информирует систему о необходимости ожидания ответа на отправленное сообщение. Для сообщений, которые не подразумевают получение ответа (например, COD_HEARTBEAT), в качестве флага рекомендуется указывать значение 0.

Ответные сообщения могут быть получены с помощью подписки типа p2mqreply, подробнее см. описание функции cg_lsn_new.

Пример вызова функции:

```
cg_publisher_t* pub; // указатель на объект публикатор
cg_msg_data* msg; // указатель на инициализированное сообщение

result = cg_pub_post(pub, msg, CG_PUB_NEEDREPLY);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to post message: 0x%X\n", result);
}
cg_pub_msgfree(pub, msg);
```

2.7.9. cg_pub_msgfree

Освобождение сообщения выполняется вызовом:

```
CG_RESULT cg_pub_msgfree(cg_publisher_t* pub, struct cg_msg_t* msg);
```

Параметрами является указатель на объект публикатора и указатель на сообщение, которое требуется освободить.

Возвращаемые значения:

CG_ERR_OK	Успешное выполнение.
CG_ERR_INVALIDARGUMENT	Функции были переданы некорректные аргументы.
CG_ERR_INTERNAL	Внутренняя ошибка. Может свидетельствовать о нарушении конфигурации или среды исполнения. Для более подробной диагностики следует обратиться к анализу журналов библиотеки.

Вызов уничтожает ранее выделенное сообщение. После вызова данной функции сообщение, на которое указывает параметр msg становится недоступным для дальнейшего использования и все ресурсы, связанные с ним, освобождаются. Функция должна быть вызвана для любого сообщения, созданного функций cg_pub_msgnew после того, как сообщение было отправлено и работа с ним завершена.

Пример вызова функции:

```
cg_publisher_t* pub; // указатель на объект публикатор
cg_msg_data* msg; // указатель на инициализированное сообщение

result = cg_pub_msgfree(pub, msg);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to post message: 0x%X\n", result);
}
```

2.8. Логирование

Функции логирования предназначены для вывода в лог форматной строки или обычной строки.

2.8.1. cg_log_trace

Печатает в лог форматную строку с trace сообщением аналогично функции sprintf языка C. Выполняется вызовом:

```
CG_RESULT CG_API cg_log_trace(const char* fmt, ...);
```

где fmt - строка языка C с форматными символами, которая оканчивается нулём. Подробнее - см. информацию о функции sprintf языка C.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы (fmt = 0).

CG_ERR_INCORRECTSTATE Не был вызван cg_env_open.

Пример вызова функции:

```
int i =10;
cg_log_trace("Print test int: %d ", i);
std::string s("test str");
cg_log_trace("Print test str: %s ", s.c_str());
```

2.8.2. cg_log_debug

Печатает в лог форматную строку с debug сообщением аналогично функции sprintf языка C. Выполняется вызовом:

```
CG_RESULT CG_API cg_log_debug(const char* fmt, ...);
```

где fmt - строка языка C с форматными символами, которая оканчивается нулём. Подробнее - см. информацию о функции sprintf языка C.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы (fmt = 0).

CG_ERR_INCORRECTSTATE Не был вызван cg_env_open.

Пример вызова функции:

```
int i =10;
cg_log_debug("Print test int: %d ", i);
std::string s("test str");
cg_log_debug("Print test str: %s ", s.c_str());
```

2.8.3. cg_log_info

Печатает в лог форматную строку с информационным сообщением аналогично функции sprintf языка C. Выполняется вызовом:

```
CG_RESULT CG_API cg_log_info(const char* fmt, ...);
```

где fmt - строка языка C с форматными символами, которая оканчивается нулём. Подробнее - см. информацию о функции sprintf языка C.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы (fmt = 0).

CG_ERR_INCORRECTSTATE Не был вызван cg_env_open.

Пример вызова функции:

```
int i =10;
cg_log_info("Print test int: %d ", i);
std::string s("test str");
cg_log_info("Print test str: %s ", s.c_str());
```

2.8.4. cg_log_error

Печатает в лог форматную строку с сообщением об ошибке аналогично функции sprintf языка C. Выполняется вызовом:

```
CG_RESULT CG_APIcg_log_error(const char*fmt, ...);
```

где fmt - строка языка C с форматными символами, которая оканчивается нулём. Подробнее - см. информацию о функции sprintf языка C.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы (fmt = 0).

CG_ERR_INCORRECTSTATE Не был вызван cg_env_open.

Пример вызова функции:

```
int i =10;
cg_log_error("Print test int: %d ", i);
std::string s("test str");
cg_log_error("Print test str: %s ", s.c_str());
```

2.8.5. cg_log_tracestr

Печатает в лог строку с trace сообщением. Выполняется вызовом:

```
CG_RESULT CG_APIcg_log_tracestr(const char*str);
```

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы.

CG_ERR_INCORRECTSTATE Не был вызван cg_env_open.

Пример вызова функции:

```
int i =10;
cg_log_tracestr("Print test int);
std::string s("test str");
cg_log_tracestr("Print test str);
```

2.8.6. cg_log_debugstr

Печатает в лог строку с debug сообщением. Выполняется вызовом:

```
CG_RESULT CG_APIcg_log_debugstr(const char*str);
```

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы.

CG_ERR_INCORRECTSTATE Не был вызван cg_env_open.

Пример вызова функции:

```
int i =10;
cg_log_debugstr("Print test int);
std::string s("test str");
cg_log_debugstr("Print test str);
```

2.8.7. cg_log_infostr

Печатает в лог строку с информационным сообщением. Выполняется вызовом:

```
CG_RESULT CG_APIcg_log_infostr(const char*str);
```

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы.

CG_ERR_INCORRECTSTATE Не был вызван cg_env_open.

Пример вызова функции:

```
int i =10;
cg_log_infostr("Print test int);
std::string s("test str");
cg_log_infostr("Print test str);
```

2.8.8. cg_log_errorstr

Печатает в лог строку с сообщением об ошибке. Выполняется вызовом:

```
CG_RESULT CG_APIcg_log_errorstr(const char*str);
```

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы.

CG_ERR_INCORRECTSTATE Не был вызван cg_env_open.

Пример вызова функции:

```
int i =10;
cg_log_errorstr("Print test int);
std::string s("test str");
cg_log_errorstr("Print test str);
```

2.9. Объекты протокола p2sys

Для аутентификации пользователя в системе Plaza-II в Cgate предусмотрен специальный протокол соединения **p2sys**. Данный протокол содержит в себе три объекта: соединение, подписчик и публикатор. Рассмотрим подробнее каждый из них.

2.9.1. Соединение p2sys

Соединение **p2sys** является особым видом соединения к роутеру Plaza-II, предназначеным только для отправки аутентификационных запросов и получения ответов на них. Соединение может быть создано и установлено следующим образом:

```
cg_conn_t* conn;

result = cg_conn_new("p2sys://127.0.0.1:4001;app_name=test_p2sys", &conn);

result = cg_conn_open(conn, 0);
```

Закрытие соединения происходит стандартным способом:

```
conn_close(conn);
```

2.9.2. Подписчик p2sys

Подписчик **p2sys** отвечает за статус соединения с роутером Plaza-II и оповещает пользователя о его изменении. Каждому статусу соответствует сообщение типа CG_MSG_DATA с уникальным msgid.

Статусы соединения с роутером делятся на 2 категории:

- статус соединения: показывает состояние соединения клиента с роутером. Если соединение установлено, подписчик получает статус **ConnectionConnected** (msgid = 3), иначе - **ConnectionDisconnected** (msgid = 4);
- статус роутера: признак аутентификации пользователя в системе. Сначала подписчик получает статус **RouterDisconnected** (msgid = 2). Если клиент отправил правильные данные аутентификации, подписчик получает статус **RouterConnected** (msgid = 1), иначе - **LogonFailed** (msgid = 5). Если в ini-файле роутера указаны правильные имя пользователя и пароль, роутер сразу принимает статус **RouterConnected**.

Создание и открытие подписчика происходит следующим образом:

```
listener_t* lsn = 0;

result = lsn_new(conn, "p2sys://;name=p2sys_lsn", &MessageCallback, 0, &lsn);

result = lsn_open(lsn, 0);
```

где функция обратного вызова MessageCallback содержит обработчик приходящих сообщений.

Закрывается подписчик следующим образом:

```
lsn_close(lsn);
```

2.9.3. Публикатор p2sys

Публикатор **p2sys** предназначен для отправки запросов аутентификации или выхода из системы. Публикатор может отправлять 2 вида сообщения типа CG_MSG_DATA:

- **RouterLogin** (msgid = 1). Запрос на аутентификацию. В поле данных содержит строку в формате "USERNAME=%имя_пользователя%;PASSWORD=%пароль%"
- **RouterLogout** (msgid = 2). Запрос на отключение. Не содержит каких-либо данных.

Создание и открытие публикатора происходят следующим образом:

```
publisher_t* pub = 0;

result = pub_new(conn, "p2sys://;name=p2sys_pub", &pub);

result = pub_open(pub, 0);
```

Закрытие публикатора происходит стандартным способом:

```
pub_close(pub);
```

2.10. Вспомогательные функции

2.10.1. cg_bcd_get

Функция позволяет получить BCD-число в виде двух компонент - целой части и положения десятичной точки.

```
CG_RESULT cg_bcd_get(void* bcd, int64_t* intpart, int8_t* scale);
```

В качестве параметров функция принимает указатель на число в BCD-формате bcd, указатель на переменную, в которую будет помещено значение числа в виде целого и указатель на переменную, в которую будет помещено положение десятичной точки относительно конца числа.

Например, для исходного числа 123.45 функция запишет в переменную intpart значение 12345, а в переменную scale значение 2.

Важно

Максимальное количество знаков, представимых в виде 64-х битного целого равно 19-ти. Для получения значений BCD-чисел, по размерности превосходящих 19 знаков, следует использовать вызов cg_getstr для представления чисел в виде строк.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы.

CG_ERR_OVERFLOW Переданное число слишком велико для представления в виде 64-х битного целого.

Пример вызова функции:

```
void* bcd; // указатель на BCD-число
int64_t value; // сюда будет записано число в виде целого
int8_t scale; // сюда будет записано положение точки

result = cg_bcd_get(bcd, &value, &scale);
if (result != CG_ERR_OK)
{
    fprintf(stderr, "Failed to convert decimal: 0x%X\n", result);
}

// напечатать значение в виде числа с плавающей точкой
printf("Value is: %f\n", (double)value/pow(10.0, scale));
```

2.10.2. cg_getstr

Функция позволяет получить строковое представление произвольного типа.

```
CG_RESULT cg_getstr(char* type, void* data, char* buffer, size_t* buffer_size);
```

В качестве параметров функция принимает тип поля в формате Plaza-2 (см. раздел Работа со схемами данных) в виде строки type, указатель на место в памяти, где содержится значение data, указатель на буфер, куда будет записано строковое представление buffer и указатель на переменную, которая содержит размер буфера buffer_size.

В случае, если размер буфера слишком мал для записи строкового представления, функция вернет код ошибки CG_ERR_BUFFERTOOSMALL и запишет в buffer_size требуемый размер буфера.

При возврате CG_ERR_OK в параметр buffer_size помещается количество символов, записанных в буфер.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы.

CG_ERR_BUFFERTOOSMALLПереданный буфер слишком мал для строкового представления типа.

CG_ERR_INCORRECTSTATE Окружение не инициализировано.

Пример вызова функции:

```
void* bcd; // указатель на BCD-число

char buf[32];
size_t bufsize = sizeof(buf);

result = cg_getstr("d26.2", bcd, buf, &bufsize);
if (result == CG_ERR_BUFFERTOOSMALL)
{
    char* buf2 = new char[bufsize];
    result = cg_getstr("d26.2", bcd, buf2, &bufsize);
    if (result != CG_ERR_OK)
        fprintf(stderr, "Failed to convert value: 0x%X\n", result);
    else
        printf("Value is %s\n", buf2);
    delete[] buf2;
}
else
if (result == CG_ERR_OK)
{
    printf("Value is %s\n", buf);
}
else
```

```
{
    fprintf(stderr, "Failed to convert value: 0x%X\n", result);
}
```

2.10.3. cg_msg_dump

Функция позволяет получить текстовый дамп произвольного сообщения.

CG_RESULT **cg_msg_dump**(struct cg_msg_t* ***msg***, struct cg_scheme_desc_t* ***schemedesc***, char* ***buffer***, size_t* ***buffer_size***);

В качестве параметров функция принимает указатель на сообщение msg, указатель на описание схемы schemedesc, указатель на буфер, куда будет записан текстовый дамп buffer и указатель на переменную, которая содержит размер буфера buffer_size.

В случае, если размер буфера слишком мал для записи строкового представления, функция вернет код ошибки CG_ERR_BUFFERTOOSMALL и запишет в buffer_size требуемый размер буфера.

Возвращаемые значения:

CG_ERR_OK Успешное выполнение.

CG_ERR_INVALIDARGUMENTФункции были переданы некорректные аргументы.

CG_ERR_BUFFERTOOSMALLПереданный буфер слишком мал для дампа сообщения.

Если параметр schemedesc не равен NULL, то функция предпримет попытку разобрать сообщение с использованием переданной схемы. Если параметр schemedesc равен NULL или сообщение отсутствует в схеме или его размер не совпадает с указанным в схеме, функция выведет шестнадцатеричный дамп сообщения.

Эту функцию удобно использовать для отладки.

Пример вызова функции:

```
cg_msg_t* msg; // указатель на сообщение
size_t bufsize = 0;

result = cg_msg_dump(msg, 0, 0, &bufsize);
if (result == CG_ERR_BUFFERTOOSMALL)
{
    char* buf = new char[bufsize];
    result = cg_msg_dump(msg, 0, buf, &bufsize);
    if (result != CG_ERR_OK)
    {
        fprintf(stderr, "Failed to dump message: 0x%X\n", result);
    }
    else
    {
        printf("%s\n", buf);
    }
    delete[] buf;
}
else
    fprintf(stderr, "Failed to dump message: 0x%X\n", result);
```

3. Описание инструментария

3.1. Утилита schemetool

Утилита schemetool предназначена для работы со схемами данных.

В настоящее время поддерживается функция формирования структур данных на языках программирования, соответствующих формату сообщений потоков данных.

3.1.1. makesrc - генерация структур

Режим makesrc предназначен для формирования исходного кода с описанием структур сообщений. Полученные структуры могут быть использованы для доступа к полям сообщений.

Формирование схемы производится следующим вызовом:

```
schemetool makesrc [options] [SOURCE SCHEME]
```

, где:

SOURCE Источник схемы. Описание схемы может быть получено из INI-файла, в этом случае в качестве SOURCE следует передать путь к ini-файлу. Также схема может быть получена из потока репликации - в этом случае в качестве SCHEME_SOURCE надо передать два параметра: --conn CONN_STR --stream STREAM_NAME; при этом CONN_STR задаёт строку соединения с роутером P2MQRouter в формате URL, а STREAM_NAME задаёт имя интересующего потока репликации

SCHEME Имя интересущей схемы; должно задаваться явно.

a options - это параметры:

-o, --output

```
-o FILENAME
```

Имя выходного файла. В этот файл будут записаны результаты работы утилиты. В случае, если данный параметр не указан, будет использован стандартный вывод stdout.

-O, --output-format

```
--output-format FORMAT
```

Формат описания структур. В настоящее время поддерживаются следующие форматы:

- c - структуры на языке C
- java - классы для Java
- cs - классы для C#
- pas - структуры на Pascal
- ini - вывод

--key — Пользовательский ключ сертификации.

--verbose, -v — Выводить логи.

--stream — Название потока по которому нужно получить вывод

--lsn — URL listener, по которому нужно получить вывод.

--conn — URL соединения с роутером Plaza II.

--tables — Список таблиц, которые должны быть в output.

--load — Загрузить плагин Cgate, который реализует дополнительные lsn и pub.

--scheme_name — Задаёт название схемы для потока.

-Dgen-table-prefix=1 — Используется для формата "c". Указание данного ключа приведет к тому, что к названиям структур сообщений будут добавлены префиксы - названия схем сообщений. Этот режим может использоваться для того, чтобы избежать конфликтов имен при использовании нескольких схем в одном INI-файле в том случае, если в разных схемах существуют сообщения с одинаковыми именами.

-Dgen-namespaces=1 — Используется для форматов "c" и "cs". Указание данного ключа приведет к формированию namespace с именем схемы для каждой из схем INI-файла. Может быть использовано для

	разрешения конфликта имен как альтернатива предыдущему варианту, если для компиляции ПО применяется компилятор C++.
-Dgen-typedef=1	Генерирует typedef для каждой таблицы
-Dgen-scheme-string=1	Генерировать строковое представление схемы
-Dc-wchar-type=TYPE	Задать тип TYPE для wchar_t строк
-Djava-class-name=CLASSNAME	Используется для формата "java". Позволяет задать имя генерируемого класса Java верхнего уровня.
-Djava-user-package=PACKAGE	Используется для формата "java". Позволяет задать имя пакета Java для генерируемого класса.
-Djava-time-format=date	Используется для формата "java". Поля, содержащие значения типа дата-время, будут сконвертированы в java.util.Date (по умолчанию)
-Djava-time-format=long	Используется для формата "java". Поля, содержащие значения типа дата-время, будут сконвертированы в значение типа long, содержащее кол-во миллисекунд прошедшее с 00:00:00 1.01.1970 г.
-Djava-bcd-format=bigdecimal	Используется для формата "java". Поля, содержащие значения типа BCD, будут сконвертированы в java.math.BigDecimal (по умолчанию)
-Djava-bcd-format=long	Используется для формата "java". Поля, содержащие значения типа BCD, будут сконвертированы в long
-Dnet-user-namespace=NAMESPACE	Используется для формата "cs". Позволяет задать имя пространства имен .NET для генерируемого класса.
-Dnet-time-format=datetime	Используется для формата "cs". Поля, содержащие значения типа дата-время, будут сконвертированы в DateTime (по умолчанию)
-Dnet-time-format=long	Используется для формата "cs". Поля, содержащие значения типа дата-время, будут сконвертированы в значение типа long, содержащее кол-во миллисекунд прошедшее с 00:00:00 1.01.1970 г.
-Dnet-bcd-format=decimal	Используется для формата "cs". Поля, содержащие значения типа BCD, будут сконвертированы в decimal (по умолчанию)
-Dnet-bcd-format=long	Используется для формата "cs". Поля, содержащие значения типа BCD, будут сконвертированы в long

Примеры использования утилиты:

```
schemetool makesrc -o futinfo.h forts_scheme.ini FUTINFO
```

- этот пример формирует в файле futinfo.h описания структур на языке C схемы FUTINFO из файла forts_scheme.ini.

```
schemetool makesrc -o futinfo.pas --output-format pas forts_scheme.ini FUTINFO
```

- этот пример формирует в файле futinfo.pas описания структур на языке Pascal схемы FUTINFO из файла forts_scheme.ini.

```
schemetool makesrc -o futinfo.h --output-format c \
                   --conn p2tcp://localhost:4001;app_name=stool \
                   --stream FORTS_REFDATA_REPL \
                   --scheme_name FutInfo
```

- этот пример формирует в файле futinfo.h описания структур на языке C схемы данных потока FORTS_REFDATA_REPL, доступного через соединение с роутером Plaza-2, запущенным на этой же машине на порту 4001.

```
schemetool makesrc -o messages.h forts_messages.ini message
```

- этот пример формирует в файле messages.h описания структур сообщений торговой системы SPECTRA из файла forts_messages.ini

```
schemetool makesrc -o futinfo2.h --output-format c --conn "p2tcp://localhost:4001;
app_name=stool" --lsn p2repl://FORTS_REFDATA_REPL
```

- Этот пример формирует в файле futinfo2.h URL соединения с роутером Plaza II и URL listener.

```
schemetool makesrc --load cgate_micexd.dll -v --output-format ini --conn "tsmr://172.20.194.99;
snapshotinterval=5000;interval=10;
roundrobin=1;SERVER=INETCUR_GATEWAY;SERVICE=inetcur_gateway;USERID=XXXX;INTERFACE=IFC_Broker18;
LOGGING=4,0;SYNCTIME=0;CACHEFOLDER=.\cache" --lsn "data://;pricefieldtype=d16.6" --output res.ini
```

- Этот пример загружает plugin Cgate для реализации дополнительных lsn и pub.

3.2. Утилита change_password

Утилита change_password предназначена для изменения пароля пользователя в торговой системе. Утилита использует протокол p2mqpwd, который описан в разделе "Объекты протокола изменения пароля". Изменение пароля производится следующим вызовом:

```
change_password [options]
```

где options - это параметры:

--app_name	имя приложения. Необязательный параметр;
--local_pass	пароль для локального соединения с роутером. Необязательный параметр;
--host	ip-адрес роутера. Необязательный параметр, значение по умолчанию 127.0.0.1;
--port	порт роутера. Необязательный параметр. Значение по умолчанию 4001;
--ini	ини-файл с настройками логирования. Необязательный параметр. Если ини-файл не задан, результат операции выводится на консоль.

Утилита возвращает 0 в случае успешного выполнения команды смены пароля и 1 в случае неуспеха.

Пример запуска утилиты:

```
change_password --port=4001
```

4. Описание API для Java, .NET

4.1. Описание

В поставку P2 CGate входят следующие интерфейсные библиотеки:

- cgate_java

  Библиотека, реализующая интерфейс с платформой Java

- cgate_net

  Библиотека, реализующая интерфейс с платформой .NET

4.1.1. API CGate для Java

Поддержка CGate для Java реализуется с помощью интерфейса JNI. В поставку P2 CGate входят следующие компоненты, относящиеся к поддержке Java:

- интерфейсная библиотека cgate_jni (cgate_jni.dll для Windows, libcgate_jni.so для Linux; каталог SpectraCGate/bin)
- библиотека классов Java cgate.jar (каталог SpectraCGate/SDK/lib)
- примеры использования P2 CGate на Java (каталог SpectraCGate/SDK/samples/java)

Для использования P2 CGate из Java нужно:

- использовать библиотеку cgate.jar при компиляции проекта
- при запуске проекта
  - иметь cgate.jar в classpath
  - иметь библиотеку cgate_jni в пути, используемом для загрузки динамических библиотек (задаётся свойством java.library.path)
  - иметь набор библиотек P2 CGate доступным для загрузки (содержимое каталога SpectraCGate/bin)

Существует возможность явного указания используемой библиотеки cgate_jni и пути к ней с помощью следующих свойств:

- ru.micexrts.cgate.name

  Задаёт имя файла библиотеки

- ru.micexrts.cgate.path

  Задаёт каталог, в котором находится файл библиотеки

Например:

```
java -cp .;lib/cgate.jar -Dru.micexrts.cgate.name=libcgate_jni.so.1 -Dru.micexrts.cgate.path=. MyApp
```

В этом примере запускается приложение пользователя из класса MyApp, при этом используется библиотека cgate.jar из подкаталога lib; интерфейсная библиотека cgate_jni будет взята из файла ./libcgate_jni.so.1.

4.1.2. API CGate для .NET

Поддержка CGate для платформы .NET реализована с помощью C++/CLI. В поставку P2 CGate входят следующие компоненты, относящиеся к поддержке .NET:

- сборка cgate_net.dll (каталог SpectraCGate/bin)
- примеры использования P2 CGate на .NET (каталог SpectraCGate/SDK/samples/net)

Для использования P2 CGate из .NET нужно:

- использовать сборку cgate_net.dll при компиляции проекта
- при запуске проекта
  - иметь cgate_net.dll доступной для загрузки платформой .net
  - иметь набор библиотек P2 CGate доступным для загрузки (содержимое каталога SpectraCGate/bin)

Важно

Запуск библиотеки cgate_net под платформой Mono не поддерживается.

4.2. Объект Cgate

Объект Cgate обеспечивает инициализацию рабочего окружения (см. Запуск и остановка окружения).

Описание	Java	.NET	Функция CGate API
Открытие соединения	CGate.open(String settings)	CGate.Open(String settings)	cg_env_open
Закрытие соединения	CGate.close()	CGate.Close()	cg_env_close

4.3. Объект Connection

Объект Connection обеспечивает доступ к набору функцию соединения (см. Соединение).

Описание	Java	.NET	Функция CGate API
Создание объекта соединения	Connection(String settings)	Connection(string settings)	cg_conn_new
Уничтожение объекта соединения	void dispose()	void Dispose()	cg_conn_destroy
Открытие соединения	void open(String settings)	void Open(string settings)	cg_conn_open
Закрытие соединения	void close()	void Close()	cg_conn_close
Обработка сообщений соединения	void process(int timeout)	void Process(int timeout)	cg_conn_process
Получение состояние соединения	int getState()	State	cg_conn_getstate

Важно

После завершения работы с соединением должен быть вызван метод, который явным образом освобождает ресурсы, связанные с соединением.

4.3.1. Конструктор Connection

Инициализирует новый экземпляр класса.

Синтаксис Java:

```
public Connection(String settings) throws CGateException
```

Синтаксис C#:

```
public Connection(string settings)
```

Где:

settings Строка инициализации соединения (см. cg_conn_new)

Возможные исключения:

CGateException Ошибка создания соединения

4.3.2. Метод Connection.dispose

Очистка ресурсов соединения выполняется вызовом соответствующих методов.

Синтаксис Java:

```
public void dispose() throws CGateException
```

Синтаксис C#:

```
public void Dispose()
```

Возможные исключения:

CGateException Ошибка уничтожения соединения

4.3.3. Метод Connection.open

Открытие соединения выполняется вызовом метода open().

Синтаксис Java:

```
public void open(String settings) throws CGateException
```

Синтаксис C#:

```
public void Open(string settings)
```

Возможные исключения:

CGateException	Ошибка открытия соединения

4.3.4. Метод Connection.close

Открытие соединения выполняется вызовом метода close().

Синтаксис Java:

```
public void close() throws CGateException
```

Синтаксис C#:

```
public void close()
```

Возможные исключения:

CGateException	Ошибка закрытия соединения

4.3.5. Метод Connection.process

Обработка сообщений соединения выполняется вызовом метода process().

Синтаксис Java:

```
public int process(int timeout)
```

Синтаксис C#:

```
public int Process(int timeout)
```

Возможные исключения: отсутсвуют

Возвращаемые значения:

CG_ERR_OK	Успешное завершение операции
CG_ERR_INVALIDSTATE	Недопустимое состояние соединения
CG_ERR_INTERNAL	Внутренняя ошибка

4.3.6. Свойство Connection.state

Обработка сообщений соединения выполняется вызовом метода state().

Синтаксис Java:

```
public int getState() throws CGateException
```

Синтаксис C#:

```
public State State { get; }
```

Возможные исключения:

CGateException	Ошибка закрытия соединения

Возвращаемые значения:

CLOSED	Соединение закрыто
ERROR	Соединение в состоянии ошибки
OPENING	Соединение в процессе открытия
ACTIVE	Соединение активно

4.4. Объект Listener

Объект Listener обеспечивает доступ к набору функций подписчика (см. Подписчик).

Описание	Java	.NET	Функция CGate API
Создание объекта подписчика	Listener(Connection conn, String settings, ISubscriber subscriber)	Listener(Connection conn, string settings)	cg_lsn_new
Уничтожение объекта подписчика	void dispose()	void Dispose()	cg_lsn_destroy
Открытие подписчика	void open(String settings)	void Open(string settings)	cg_lsn_open
Закрытие подписчика	void close()	void Close()	cg_lsn_close
Получение состояние соединения	int getState()	State	cg_lsn_getstate
Получение схемы подписчика	int getScheme()	Scheme	cg_lsn_getscheme
Установка обработчика	-	Handler	-

Важно

После завершения работы с подписчиком должен быть вызван метод dispose(), который явным образом освобождает ресурсы, связанные с подписчиком.

4.4.1. Конструктор Listener

Инициализирует новый экземпляр класса.

Синтаксис Java:

```
public Listener(Connection conn, String settings, ISubscriber subscriber) throws CGateException
```

Синтаксис C#:

```
public Listener(Connection conn, string settings)
```

Где:

conn Соединение, в привязке к которому создаётся подписчик

settings Строка инициализации подписки (см. cg_lsn_new)

subscriber Пользовательский обработчик сообщений (только Java; для .NET следует использовать свойства Handler)

Возможные исключения:

CGateException Ошибка создания подписки

Для Java параметр subscriber указывает на экземпляр класса, реализующего интерфейс ISubscriber:

```
public interface ISubscriber {

    public int onMessage(Connection conn, Listener listener, Message message);
}
```

При возникновении какого либо события подписчика, например, приход нового сообщения или изменение состояния подписчика, будет вызван метод onMessage объекта, переданного в параметре subscriber.

В качестве параметров будут переданы:

conn Соединение, к которому привязан подписчик

listener Подписчик, в котором произошло событие

msg Сообщение

Для .NET параметр subscriber отсутсвует; вместо него введено специальное свойтство Handler, которое позволяет устанавливать обработчик сообщений естественным для среды .NET образом.

4.4.2. Метод Listener.dispose

Очистка ресурсов подписчика выполняется вызовом метода dispose().

Синтаксис Java:

```
public void dispose() throws CGateException
```

Синтаксис C#:

```
public void Dispose()
```

Возможные исключения:

CGateException	Ошибка уничтожения подписчика

4.4.3. Метод Listener.open

Предпринимает попытку открытия подписчика.

Синтаксис Java:

```
public void open(String settings) throws CGateException
```

Синтаксис C#:

```
public void Open(string settings)
```

Возможные исключения:

CGateException	Ошибка открытия подписчика

4.4.4. Метод Listener.close

Закрывает подписку.

Синтаксис Java:

```
public void close() throws CGateException
```

Синтаксис C#:

```
public void close()
```

Возможные исключения:

CGateException	Ошибка закрытия подписчика

4.4.5. Свойство Listener.State

Возвращает текущее состояние подписчика.

Синтаксис Java:

```
public int getState() throws CGateException
```

Синтаксис C#:

```
public State State { get; }
```

Возможные исключения:

CGateException	Ошибка получения состояния подписчика

Возвращаемые значения:

CLOSED	Подписчик закрыт
ERROR	Подписчик в состоянии ошибки
OPENING	Подписчик в процессе открытия
ACTIVE	Подписчик активен

4.4.6. Свойство Listener.Scheme

Возвращает текущую схему данных подписчика.

Синтаксис Java:

```
public Scheme getScheme() throws CGateException
```

Синтаксис C#:

```
public Scheme Scheme { get; }
```

Возможные исключения:

CGateException Ошибка получения схемы данных

Возвращаемое значение - это описание текущей схемы данных подписчика или null, если подписчик работает в режиме без схемы.

Важно

Схема данных подписчика доступна с момента получения сообщения OPEN до момента закрытия подписчика или его перехода в состояние ошибки.

Важно

Схема данных подписчика может изменяться между двумя событиями CLOSE и OPEN; т.е. после повторного открытия подписчика, его схема может отличаться от той, которая была действительна во время прошлой сессии активности.

4.4.7. Свойство Listener.Handler

Позволяет устанавливать пользовательский обработчик сообщений подписчика.

Синтаксис C#:

```
public MessageHandler Handler { get; set; }
```

Пользовательские обработчики должны соответствовать следующему виду:

```
delegate int MessageHandler(Connection conn, Listener listener, Message msg);
```

, где в качестве параметров будут переданы:

conn Соединение, к которому привязан подписчик

listener Подписчик, в котором произошло событие

msg Сообщение

4.5. Объект Publisher

Объект Publisher обеспечивает доступ к набору функций публикатора (см. Публикатор).

Описание	Java	.NET	Функция CGate API
Создание объекта публикатора	Publisher(Connection conn, String settings)	Publisher(Connection conn, string settings)	cg_pub_new
Уничтожение объекта публикатора	void dispose()	void Dispose()	cg_pub_destroy
Открытие публикатора	void open(String settings)	void Open(string settings)	cg_pub_open
Закрытие публикатора	void close()	void Close()	cg_pub_close
Получение состояния публикатора	int getState()	State	cg_pub_getstate
Получение схемы публикатора	int getScheme()	Scheme	cg_pub_getscheme
Создание сообщения для отправки	Message newMessage(int idType, Object id)	Message NewMessage(MessageFlag idType, Object id);	cg_pub_msgnew
Отправка сообщения	void post(Message msg, int flags)	Message Post(Message msg, PublisherFlag flags);	cg_pub_post

Важно

После завершения работы с публикатором должен быть вызван метод dispose(), который явным образом освобождает ресурсы, связанные с публикатором.

4.5.1. Конструктор Publisher

Инициализирует новый экземпляр класса.

Синтаксис Java:

```
public Publisher(Connection conn, String settings) throws CGateException
```

Синтаксис C#:

```
public Publisher(Connection conn, string settings)
```

Где:

conn Соединение, в привязке к которому создаётся публикатор

settings Строка инициализации публикатора (см. cg_pub_new)

Возможные исключения:

CGateException Ошибка создания публикатора

4.5.2. Метод Publisher.dispose

Очистка ресурсов публикатора выполняется вызовом метода dispose().

Синтаксис Java:

```
public void dispose() throws CGateException
```

Синтаксис C#:

```
public void Dispose()
```

Возможные исключения:

CGateException Ошибка уничтожения публикатора

4.5.3. Метод Publisher.open

Предпринимает попытку открытия публикатора.

Синтаксис Java:

```
public void open(String settings) throws CGateException
```

Синтаксис C#:

```
public void Open(string settings)
```

Возможные исключения:

CGateException Ошибка открытия публикатора

4.5.4. Метод Publisher.close

Закрывает публикатор.

Синтаксис Java:

```
public void close() throws CGateException
```

Синтаксис C#:

```
public void close()
```

Возможные исключения:

CGateException Ошибка закрытия публикатора

4.5.5. Свойство Publisher.State

Возвращает текущее состояние публикатора.

Синтаксис Java:

```
public int getState() throws CGateException
```

Синтаксис C#:

```
public State State { get; }
```

Возможные исключения:

CGateException Ошибка получения состояния публикатора

Возвращаемые значения:

CLOSED Публикатор закрыт

ERROR Публикатор в состоянии ошибки

OPENING Публикатор в процессе открытия

ACTIVE Публикатор активен

4.5.6. Свойство Publisher.Scheme

Возвращает текущую схему данных публикатора.

Синтаксис Java:

```
public Scheme getScheme() throws CGateException
```

Синтаксис C#:

```
public Scheme Scheme { get; }
```

Возможные исключения:

CGateException Ошибка получения схемы данных

Возвращаемое значение - это описание текущей схемы данных публикатора или null, если публикатор работает в режиме без схемы.

> **Важно**
>
> Схема данных публикатора может изменяться между двумя событиями CLOSE и OPEN; т.е. после повторного открытия публикатора, его схема может отличаться от той, которая была действительна во время прошлой сессии активности.

4.5.7. Метод Publisher.newMessage

Создаёт новое сообщение для отправки.

Синтаксис Java:

```
public void newMessage(int idType, Object id) throws CGateException
```

Синтаксис C#:

```
public void NewMessage(MessageFlag idType, Object id)
```

, где:

idType Тип идентификатора сообщения. Одно из значений:

- KEY_INDEX - параметр id является номером требуемого сообщения в схеме
- KEY_ID - параметр id является уникальным числовым идентификатором требуемого сообщения в схеме
- KEY_NAME - параметр id является строкой - именем требуемого сообщения в схеме

id Идентификатор сообщения (Integer или String, в зависимости от значения параметра idType)

Возможные исключения:

CGateException Ошибка создания сообщения

Созданное сообщение содержит буфер, по размеру соответствующий описанию сообщения в схеме данных.

4.5.8. Метод Publisher.post

Отправляет сообщение.

Синтаксис Java:

```
public void post(Message msg, int flags) throws CGateException
```

Синтаксис C#:

```
public void Post(Message msg, PublisherFlag flags)
```

, где:

msg Сообщения для отправки

flags Флаги отправки сообщения. В настоящее время поддерживается единственный флаг NEED_REPLY, который означает необходимость получения ответа на отправленное сообщение.

Возможные исключения:

CGateException Ошибка отправки сообщения

Отправленное сообщение после вызова метода post() не используется и может быть удалено или использовано для повторной отправки.

Важно

Объект Publisher может отправлять только те сообщения, которые были созданы этим же экземпляром объекта.

4.6. Объект Message

Объект Message обеспечивает доступ к сообщениям.

Описание	Java	.NET	CGate API
Уничтожение объекта сообщения	void dispose()	void Dispose()	cg_pub_msgfree
Получение типа сообщения	int getType()	Type	поле type структуры cg_msg_t
Получение буфера с данными	ByteBuffer getData()	Data	поле data структуры cg_msg_t
Получение отладочного представления сообщения	String toString()	string ToString()	cg_msg_dump

Пользователь отвечает за уничтожение созданных им сообщений для отправки явным вызовом метода dispose(). Сообщения, которые пользователь получает в обработчик подписки не должны уничтожаться, так как владельцем таких сообщений является библиотека P2 CGate.

4.6.1. Метод Message.dispose

Очистка ресурсов сообщения выполняется вызовом метода dispose().

Синтаксис Java:

```
public void dispose() throws CGateException
```

Синтаксис C#:

```
public void Dispose()
```

Возможные исключения:

CGateException Ошибка уничтожения сообщения

4.6.2. Свойство Message.Type

Возвращает тип сообщения.

Синтаксис Java:

```
public int getType()
```

Синтаксис C#:

```
public MessageType Type { get; }
```

4.6.3. Свойство Message.Data

Возвращает буфер данных сообщения

Синтаксис Java:

```
public java.nio.ByteBuffer getData()
```

Синтаксис C#:

```
public System.IO.UnmanagedMemoryStream Data { get; }
```

Размер буфера с данными доступен через соответствующий вызов объекта, возвращаемого свойством. Формат буфера соответствует используемой схеме данных.

Свойство Data может возвращать null - это означает, что сообщение не содержит данных.

4.6.4. Метод Message.toString

Возвращает текстовое представление сообщения.

Синтаксис Java:

```
public String toString()
```

Синтаксис C#:

```
public string ToString()
```

Данное представление может использоваться в отладочных нуждах.

4.6.5. Типы сообщений

Для более комфортной работы с P2 CGate введены классы, описывающие конкретные типы сообщений. Такие классы содержат дополнительную информацию. Пользователь может получить доступ к дополнительным свойствам сообщения выполнив преобразования типа объекта (каст), основанное на анализе свойства Message.Type.

4.6.5.1. Объект OpenMessage

Описывает сообщения типа CG_MSG_OPEN - открытие подписчика.

Объект не содержит дополнительных полей.

4.6.5.2. Объект CloseMessage

Описывает сообщения типа CG_MSG_CLOSE - закрытие подписчика.

Описание	Java	.NET	CGate API
Причина закрытия объекта	int getReason()	Reason CloseReason()	-

4.6.5.3. Объект DataMessage

Описывает сообщения типа CG_MSG_DATA - сообщение с данными.

Дополнительные свойства объекта:

Описание	Java	.NET	CGate API
Номер сообщения в схеме данных	int getMsgIndex()	MsgIndex	поле msg_index структуры cg_msg_data_t
Числовой идентификатор сообщения в схеме данных	int getMsgId()	MsgId	поле msg_id структуры cg_msg_data_t
Имя сообщения в схеме данных	int getMsgName()	MsgName	поле msg_name структуры cg_msg_data_t
Адрес отправителя/получателя сообщения	string getAddress()	Address	поле addr структуры cg_msg_data_t
Пользовательский номер сообщения	int getUserId()/void setUserId(int val)	UserId	поле user_id структуры cg_msg_data_t
Список полей сообщения	Value[] getFields()	Fields	-
Получение поля по имени	Value] getField(String name)	Field[string]	-

4.6.5.4. Объект StreamDataMessage

Описывает сообщения типа CG_MSG_STREAM_DATA - сообщение с потоковыми данными.

Дополнительные свойства объекта:

Описание	Java	.NET	CGate API
Номер сообщения в схеме данных	int getMsgIndex()	MsgIndex	поле msg_index структуры cg_msg_streamdata_t
Числовой идентификатор сообщения в схеме данных	int getMsgId()	MsgId	поле msg_id структуры cg_msg_streamdata_t
Имя сообщения в схеме данных	int getMsgName()	MsgName	поле msg_name структуры cg_msg_streamdata_t
Номер сообщения в потоке	long getRev()	Rev	поле rev структуры cg_msg_streamdata_t
Список полей сообщения	Value[] getFields()	Fields	-
Получение поля по имени	Value] getField(String name)	Field[string]	-

4.6.5.5. Объект TnBeginMessage

Описывает сообщения типа CG_MSG_TN_BEGIN - идентифицирует начало транзакции для потоковых данных.

Объект не содержит дополнительных полей.

4.6.5.6. Объект TnCommitMessage

Описывает сообщения типа CG_MSG_TN_COMMIT - идентифицирует завершение транзакции для потоковых данных.

Объект не содержит дополнительных полей.

4.6.5.7. Объект P2MQTimeoutMessage

Описывает сообщения типа CG_MSG_P2MQ_TIMEOUT - сообщение о превышении времени ожидания ответа на отправленное сообщение.

Дополнительные свойства объекта:

Описание	Java	.NET	CGate API
Пользовательский номер сообщения	int getUserId()/void setUserId(int val)	UserID	поле user_id структуры cg_msg_data_t

4.6.5.8. Объект P2ReplLifeNumMessage

Описывает сообщения типа CG_MSG_P2REPL_LIFENUM - сообщение об изменении номера жизни схемы данных.

Дополнительные свойства объекта:

Описание	Java	.NET	CGate API
Новый номер жизни схемы данных	int getLifeNumber()	LifeNumber	поле life_number структуры cg_lifenumber_t
Поле flag (зарезервировано)	-	-	-

4.6.5.9. Объект P2ReplClearDeletedMessage

Описывает сообщения типа CG_MSG_P2REPL_CLEARDELETED - сообщение об удалении диапазона данных по указанной таблице.

Дополнительные свойства объекта:

Описание	Java	.NET	CGate API
Номер таблицы	int getTableIdx()	TableIdx	поле table_idx структуры cg_data_cleardeleted_t
Номер ревизии, ниже которого данные удаляются	long getTableRev()	TableRev	поле table_rev структуры cg_data_cleardeleted_t

Описание	Java	.NET	CGate API
Поле flags для внутреннего использования	-	-	поле flags структуры cg_data_cleardeleted_t

4.6.5.10. Объект P2ReplOnlineMessage

Описывает сообщения типа CG_MSG_P2REPL_ONLINE - сообщение о переходе потока данных в состояние ONLINE.

Объект не содержит дополнительных полей.

4.6.5.11. Объект P2ReplStateMessage

Описывает сообщения типа CG_MSG_P2REPL_REPLSTATE - сообщение, содержащее состояние потока данных для повторного открытия.

Дополнительные свойства объекта:

Описание	Java	.NET	CGate API
Данные для переоткрытия потока	String getReplState()	ReplState	значение *data сообщения