Российская торговая система

Библиотека COM-объектов P2ClientGate

Описание API

Содержание

1. Назначение и состав библиотеки

Библиотека COM-объектов **P2ClientGate** является официальными программным интерфейсом, предоставляемым сторонним компаниям для создания программного обеспечения, работающего на фондовом рынке РТС. Данный интерфейс обеспечивает возможность создания и отсылки бизнес-сообщений в ТС, а также получения рыночной информации из нее (репликация данных).

Библиотека включает в себя следующие объекты:

- **Application** — основной объект, предназначенный для инициализации библиотеки **P2ClientGate**.
- **Connection** — объект предназначен для создания и работы с соединениями, а также приема и обработки сообщений.
- **Message** — объект предназначен для обработки полей бизнес-сообщения и отправки сообщений.
- **MessageFactory** — объект предназначен для создания бизнес-сообщений.
- **DataStream** — объект предназначен для организации получения репликационных данных.
- **TableSet** — объект предназначен для работы с клиентской схемой репликации.
- **DataBuffer** — служебный объект, служащий для оптимизации работы с данными.
- **Record** — объект предназначен для работы с записями.

2. Требуемое программное обеспечение

Для работы библиотеке P2ClientGate требуется установка и поддержание соединения с коммуникационной инфраструктурой РТС, обеспечиваемое ПО P2MQRouter. Программный модуль P2MQRouter отвечает за установку и поддержание TCP-соединений с серверами РТС, обеспечивает поддержку различных протоколов аутентификации пользователей в сети, обеспечивает маршрутизацию и доставку сообщений между логическими узлами сети.

Соединение между модулем P2MQRouter и конечными клиентскими приложениями осуществляется также по протоколу TCP. При этом, если приложение работает на том же физическом компьютере, что и роутер и для соединения использует IP-адрес loopback-интерфейса (127.0.0.1), то дополнительных параметров при соединении не требуется. Если роутер и приложение-клиент находятся на разных компьютерах, то требуется регистрация имени приложения клиента и сопоставление ему локального пароля.

Подробнее модуль P2MQRouter описан в документе P2MQRouter_Module_Description.doc

3. Application

Основной объект библиотеки **P2ClientGate**, предназначенный для ее инициализации. Он позволяет задавать параметры инициализации, отличные от параметров по умолчанию (пользовательский ini-файл), а также устанавливать тип парсера, используемого при отправке (приеме) данных. Тип парсера устанавливается в зависимости от того, на какой платформе работают приложения ТС — **Plaza** или **Plaza-II**. Использование разных парсеров обусловлено тем, что сообщения для различных платформ отличаются форматом поля **Body**.

Для того, чтобы задать параметры инициализации, отличные от параметров по умолчанию следует явно создать объект **Application** и в вызове метода **StartUp** указать имя пользовательского ini-файла и путь к нему. Тип парсера задается как свойство объекта и изменяется с помощью стандартных методов **get** и **put**. Делать это необходимо до начала работы с другими объектами библиотеки.

Если ничего из вышеперечисленного менять не требуется и используются параметры по умолчанию, объект **Application** явно создавать не обязательно, он формируется автоматический при создании любого другого объекта библиотеки. По умолчанию параметры инициализации считываются из файла *P2ClientGate.ini* в директории запуска. Парсер по умолчанию — парсер для **Plaza-II**.

Объект **Application** содержит один единственный интерфейс — **IP2Application**.

3.1. Интерфейс IP2Application

3.1.1. Свойства

ParserType [in/out] ULONG — тип парсера. Возможны следующие значения:

- **1** — Plaza.
- **2** — Plaza-II. Значение по умолчанию.

3.1.2. Методы

3.1.2.1. StartUp

```
StartUp ([in] BSTR IniFileName);
```

Назначение

Инициализация библиотеки **P2ClientGate** с параметрами, заданными в пользовательском ini-файле.

Аргументы

IniFileName — имя файла инициализации и путь к нему.

3.1.2.2. CleanUp

```
CleanUp (void);
```

Назначение

Деинициализация библиотеки **P2ClientGate**.

4. Connection

Объект предназначен для создания и работы с соединениями. Он позволяет создавать (разрывать) локальные соединения приложения с роутером, инициировать вход роутера в сеть и выход из нее, а также принимать и обрабатывать сообщения. При этом для приема и обработки сообщений объект предоставляет возможность организовать подписку на получение определенных сообщений. Подписчик получает сообщения в свой коллбэк по заранее указанному при подписке набору параметров:

1. Все сообщения.
2. Сообщения от определенного отправителя.
3. Сообщения из определенной категории.
4. Сообщения с определенной категорией и типом.

Подписки типа 2 и 3 могут быть использованы прочими системными компонентами библиотеки. Например, модуль клиента репликации должен получать все сообщения из категории репликационных. Подписки типа 4 могут быть рекомендованы для реализации обработчиков конкретных бизнес-сообщений.

Регистрация подписчика осуществляется с помощью метода **RegisterReceiver**. Вызов подписок производится последовательно в порядке их регистрации.

Объект включает в себя следующие интерфейсы:

- **IP2Connection**
- **IP2ConnectionEvent**
- **IP2MessageReceiver**

4.1. Интерфейс IP2Connection

Основной интерфейс объекта, который используется приложением для создания и работы с соединениями, а также приема и обработки сообщений.

4.1.1. Свойства

- **Status** [out] LONG — состояние соединения или роутера. Возможны следующие значения:

- **0x00000001** (константа CS_CONNECTION_DISCONNECTED) — соединение еще не установлено.
- **0x00000002** (константа CS_CONNECTION_CONNECTED) — соединение установлено.
- **0x00000004** (константа CS_CONNECTION_INVALID) — нарушен протокол работы соединения, дальнейшая работа возможна только после повторной установки соединения.
- **0x00000008** (константа CS_CONNECTION_BUSY) — соединение временно заблокировано функцией получения сообщения.
- **0x00010000** (константа CS_ROUTER_DISCONNECTED) — роутер запущен, но не присоединен к сети. Роутер не создает удаленных исходящих соединений, имени не имеет, принимает только локальные соединения, при этом локальные приложения могут взаимодействовать между собой посредством роутера.
- **0x00020000** (константа CS_ROUTER_RECONNECTING) — роутер получил аутентификационную информацию (имя и пароль), пытается установить исходящее соединение, но ни одно исходящее соединение еще не установлено.
- **0x00040000** (константа CS_ROUTER_CONNECTED) — роутер установил по крайней мере одно исходящее соединение, аутентификационная информация подтверждена.
- **0x00080000** (константа CS_ROUTER_LOGINFAILED) — по крайней мере один из сервисов отклонил аутентификационную информацию. В этом случае аутентификационная информация становится недействительной. Для продолжения работы необходимо провести последовательный вызов методов **Logout** и **Login**.
- **0x00100000** (константа CS_ROUTER_NOCONNECT) — за заданное количество попыток роутер не смог установить ни одного исходящего соединения, но аутентификационная информация подтверждена. Роутер больше не будет пытаться установить исходящие соединения. Для продолжения работы необходимо провести последовательный вызов методов **Logout** и **Login**.

- **AppName** [in/out] BSTR — имя приложения, для которого необходимо установить соединение. Имя приложения должно быть уникальным в рамках одного роутера.
- **NodeName** [out] BSTR — имя роутера.
- **Host** [in/out] BSTR — IP-адрес узла либо UNC-имя.
- **Port** [in/out] ULONG — номер порта TCP/IP.
- **Password** [in] BSTR — пароль для локального соединения.
- **Timeout** [in/out] ULONG — таймаут, в течение которого ожидается установка локального соединения.
- **LoginStr** [in/out] BSTR — строка с аутентификационной информацией роутера (логии/пароль). Формат строки: USERNAME=<user_name>;PASSWORD=<password>.

Свойства **AppName**, **NodeName**, **Host**, **Port**, **Password** и **Timeout** должны быть заданы до момента вызова метода **Connect**. В случае изменения данных свойств для того, чтобы изменения вступили в силу необходимо провести последовательный вызов методов **Disconnect** и **Connect**.

Параметры аутентификации роутера (**LoginStr**) должны быть заданы до момента вызова метода **Login**.

4.1.2. Методы

4.1.2.1. Connect

```
Connect ([out, retval] ULONG* errClass);
```

Назначение

Создание локального соединения приложения с роутером.

Аргументы

errClass — класс ошибки в том случае, если функция вернет результат отличный от **P2ERR_OK**. Используются следующие классы ошибок:

- **0** (константа P2MQ_ERRORCLASS_OK) — имеет смысл повторно попытаться установить соединение с теми же параметрами (возможно, имеются временные перебои в работе локальной сети и т.п.).
- **1** (константа P2MQ_ERRORCLASS_IS_USELESS) — вероятно повторная попытка установить соединение приведет к тем же результатом, необходимо изменить параметры функции.

4.1.2.2. Disconnect

```
Disconnect (void);
```

Назначение

Разрыв локального соединения.

4.1.2.3. Login

```
Login ();
```

Назначение

Инициирование входа роутера в сеть. Роутер создает исходящее удаленное соединение с вышестоящим роутером и аутентифицируется в сети.

4.1.2.4. Logout

```
Logout(void);
```

Назначение

Разрыв всех удаленных соединений роутера.

4.1.2.5. ProcessMessage

```
ProcessMessage (
[out] ULONG* cookie,
[in] ULONG pollTimeout);
```

Назначение

Прием и обработка сообщений, в том числе и репликационных.

Аргументы

- **cookie** — уникальный идентификатор сообщения;
- **pollTimeout** — таймаут в миллисекундах, в течение которого ожидается получение сообщения.

4.1.2.6. RegisterReceiver

```
RegisterReceiver (
[in] IP2MessageReceiver* newReceiver,
[out,retval] ULONG* cookie);
```

Назначение

Регистрация подписчика.

Аргументы

- **newReceiver** — указатель на интерфейс обратного вызова;
- **cookie** — уникальный идентификатор сообщения. Используется для того чтобы можно было отменить подписку, а также именно по нему в методе Connection.ProcessMessage определяется получатель сообщения.

4.1.2.7. UnRegisterReceiver

```
UnRegisterReceiver ([in] ULONG cookie);
```

Назначение

Отмена регистрации подписчика.

4.1.2.8. ResolveService

```
ResolveService (
[in] BSTR service,
[out,retval] BSTR* address);
```

Назначение

Получение полного адреса приложения по имени сервиса, который оно предоставляет.

Аргументы

- **service** — имя сервиса;
- **address** — полный адрес приложения.

4.2. Интерфейс IP2ConnectionEvent

Интерфейс обратного вызова для обработки факта изменения статуса соединения.

4.2.1. Методы

4.2.1.1. ConnectionStatusChanged

```
ConnectionStatusChanged (
[in] IP2Connection* conn,
[in] TConnectionStatus newStatus);
```

Назначение

Нотификация об изменении состояния соединения или роутера.

Аргументы

- **conn** — указатель на интерфейс соединения;
- **newStatus** — новое состояние соединения или роутера.

4.3. Интерфейс IP2MessageReceiver

Интерфейс обратного вызова для обработки сообщений по подписке.

4.3.1. Методы

4.3.1.1. GetFilter

```
GetFilter (
[out] VARIANT* from,
[out] VARIANT* type,
[out] VARIANT* category);
```

Назначение

Задает правила отбора сообщений для заданного подписчика. Отбирать можно по отправителю, типу и категории сообщения. Вызывается при регистрации подписки (ресивера).

Аргументы

- **from** — отправитель сообщения;
- **type** — тип сообщения;
- **category** — категория сообщения.

4.3.1.2. PutMessage

```
PutMessage ([in] IDispatch* pMsg);
```

Назначение

Обработка сообщений.

Аргументы

pMsg — указатель на интерфейс сообщения.

5. Message

Объект предназначен для обработки полей бизнес-сообщения и отправки сообщений. Он включает в себя следующие интерфейсы:

- **IP2BLMessage**
- **IP2AsyncMessageEvents**

5.1. Интерфейс IP2BLMessage

Основной интерфейс объекта, который используется приложением для обработки полей бизнес-сообщения и отправки сообщений.

5.1.1. Свойства

- **Name** [out] BSTR — имя сообщения
- **DestAddr** [in/out] BSTR — адрес получателя.
- **Field** ([in] BSTR name) [in/out] VARIANT — свойство, описывающее поле сообщения. Оно позволяет по имени поля получить или задать его значение.

  В стандартном сообщении предопределены следующие поля:

  – **P2_From** — текстовое поле — отправитель сообщения (заполняется автоматически)

  – **P2_To** — текстовое поле — получатель сообщения (адрес сервера. Конкретные адреса серверов указаны в документации на соответствующую систему)

  – **P2_Type** — четыре байта — тип сообщения.

  – **P2_SendId** — восемь байт — идентификатор исходящего сообщения (заполняется автоматически).

  – **P2_ReplyId** — восемь байт — ссылка на идентификатор оригинального сообщения (заполняется автоматически).

  – **P2_Category** — текстовое поле — категория сообщения.

  – **P2_Body** — поле переменной длины — тело сообщения.

  Пользователь может добавлять свои поля к сообщению. Ниже приведены правила мапирования типов данных COM и платформы Plaza2.

  P2Message::get_Field

  u1 —> VT_U1

  u2 —> VT_U2

  u4 —> VT_U4

  u8 —> VT_U8

  i1 —> VT_I1

  i2 —> VT_I2

  i4 —> VT_I4

  i8 —> VT_I8

  a —> VT_U1

  c —> VT_BSTR

  d,s —> VT_BSTR

t —> VT_BSTR

f —> VT_BSTR

b —> не поддерживается

P2Message::put_Field

u1 <— VARIANT::ChangeType

u2 <— VARIANT::ChangeType

u4 <— VARIANT::ChangeType

u8 <— VT_U1,VT_U2,VT_U4,VT_U8,VT_I1,VT_I2,VT_I4,VT_I8

c <— VT_BSTR

s,d <— VT_U1,VT_U2,VT_U4,VT_U8,VT_I1,VT_I2,VT_I4,VT_I8,VT_BSTR

t <— VT_BSTR

f <— VT_BSTR

a <— VT_BSTR[0]

b <— не поддерживается

Описание правил заполнения конкретных сообщений для разных типов сервисов приведены в документации по этим сервисам.

5.1.2. Методы

5.1.2.1. Send

```
Send (
[in] IP2Connection* conn,
[in] ULONG timeout,
[out,retval] IP2BLMessage** reply);
```

Назначение

Отправка сообщений типа ***Send*** – с блокировкой вызывающего потока до момента получения ответа на сообщение или до истечения таймаута.

Аргументы

- **conn** — указатель на интерфейс соединения;
- **timeout** — таймаут в миллисекундах, в течение которого ожидается ответное сообщение;
- **reply** — ответное сообщение.

5.1.2.2. Post

```
Post ([in] IP2Connection* conn);
```

Назначение

Отправка сообщений типа ***Post*** – без ожидания ответа. отправляющий поток разблокируется непосредственно после записи тела сообщения в сокет TCP-соединения с роутером

5.1.2.3. SendAsync

```
SendAsync (
[in] IP2Connection* conn,
[in] ULONG timeout,
[in] IDispatch* event);
```

Назначение

Асинхронная отправка сообщений типа ***Send*** (с получением ответного сообщения без блокировки потока).

Аргументы

- **conn** — указатель на интерфейс соединения;
- **timeout** — таймаут в миллисекундах, в течение которого ожидается ответное сообщение;
- **event** — указатель на интерфейс обратного вызова.

5.2. Интерфейс IP2AsyncMessageEvents

Интерфейс обратного вызова для получения результата доставки сообщения, отправленного с помощью метода **SendAsync**.

5.2.1. Методы

5.2.1.1. DeliveryEvent

```
DeliveryEvent (
[in] interface IP2BLMessage* reply,
[in] ULONG errCode);
```

Назначение

Результат доставки сообщения.

Аргументы

- **reply** — указатель на интерфейс сообщения типа ***Reply***.
- **errCode** — код ошибки. Возможные значения:
  - **P2MQ_TIMEOUT** — за указанное время таймаута ответ не пришел.
  - **P2ERR_OK** — ответное сообщение пришло.

6. MessageFactory

Объект предназначен для создания сообщений. При этом он позволяет при создании сообщений оперировать не только схемой сообщений по умолчанию (заданной в *P2ClientGate.ini*), а реализовывать набор классов сообщений, создаваемых по различным схемам.

По умолчанию схема сообщений задана как схема БД с именем ***message***. Имена сообщений соответствуют названиям таблиц, перечисленным в схеме.

Пример описания схем сообщений в ini-файле

```
[dbscheme:message]
table=ADD_QUOTE
table=_Proto_1
table=_Proto_2
table=DEL_QUOTE
table=_Proto_5

[table:message:ADD_QUOTE]
field=issue_name,c7
```

```
field=firm_name,c7

field=type,c1

field=price,d16.5

field=qty,d16.0



[table:message:_Proto_1]

field=id,i8



[table:message:_Proto_2]

field=err,u4



[table:message:DEL_QUOTE]

field=id,i8



[table:message:_Proto_5]

field=err,u4
```

Если требуется использовать другие схемы сообщений, следует при инициализации объекта указать пользовательский ini-файл, содержащий такие схемы.

Для каждого сервиса, доступного через шлюз P2ClientGate , РТС будет формировать готовые файлы-описания схем.

Помимо создания исходящих сообщений, схемы сообщений используются для разбора входящий сообщений (reply). При создании сообщения (метод **CreateMessageByName**) по имени сообщения находится его схема, и в соответствии с этой схемой поля заносятся в поле **Body** исходящего сообщения. При обработке входящего сообщения из него извлекается категория (строка) и тип сообщения, который затем преобразуется в строку, далее эти две строки объединяются и по результирующей строке в ini-файле ищется описание схемы сообщения. Затем содержание поля **Body** входящего сообщения разбирается в поля, описанные в найденной схеме.

Объект **MessageFactory** содержит один единственный интерфейс — **IP2BLMessageFactory**.

6.1. Интерфейс IP2BLMessageFactory

6.1.1. Методы

6.1.1.1. Init

Init (

BSTR structFile,

BSTR signFile);

Назначение

Инициализация объекта.

Аргументы

- **structFile** — файл, содержащий схему сообщений.
- **signFile** — не используется.

6.1.1.2. CreateMessageByName

CreateMessageByName (

[in] BSTR msgName,

[out,retval] IP2BLMessage** newMsg);

Назначение

Создание сообщения по имени.

Аргументы

msgName — имя сообщения (имя таблицы БД).

7. DataStream

Каждый сервер бизнес-логики в РТС для передачи потока данных клиентам использует механизм репликации данных, позволяющий клиенту получать и поддерживать в состоянии он-лайн копию подмножества данных реляционной БД сервера.

Объект **DataStream** инкапсулирует функционал работы с одним потоком репликацонных данных. Он включает в себя следующие интерфейсы:

- **IP2DataStream**
- **IP2DataStreamEvents**

7.1. Интерфейс IP2DataStream

Основной интерфейс объекта, который используется для организации получения репликационных данных.

7.1.1. Свойства

- **TableSet** [in/out] — набор таблиц в схеме репликации. Реализуется как отдельный COM-объект (см. раздел **TableSet**).
- **StreamName** [in/out] BSTR — имя потока репликации.
- **DBConnString** [in/out] BSTR — строка соединения с БД. содержит перечень параметров для соединения, разделенных точкой с запятой. Перечень зависит от того, какой DB-драйвер используется для установки соединения

В настоящее время доступны два драйвера:

P2DBSQLite.dll – простая встраиваемая БД SQLite. Параметры: ини-файл с расширенными опциями sqlite (не обязателен), имя файла, который будет содержать БД

P2DBODBC.dll – интерфейс с драйверами ODBC. Параметры: ини-файл с расширенными опциями ODBC (не обязателен), строка соединения ODBC.

Примеры строк:

'P2DBSQLite.dll;dbTest.ini;C:\dbTest'

'P2DBODBC.dll;crypto.ini;DRIVER={SQLServer};SERVER=TEST1;DATABASE=crypto;UID=autotest;PWD=autotest'

- **Type** [in/out] enum TRequestType — тип потока репликации. Тип потока определяет источник и способ получения данных (снэпшот/онлайн). Возможны следующие значения:
  - **0** (RT_LOCAL) — данные получаются из локальной БД клиента репликации в режиме снэпшот.
  - **1** (RT_COMBINED_SNAPSHOT) — используются локальные данные плюс данные от сервера репликации в режиме снэпшот. После получения всех данных от сервера поток закрывается.
  - **2** (RT_COMBINED_DYNAMIC) — используются локальные данные плюс данные от сервера репликации в режиме снэпшот. После получения всех данных от сервера поток переходит в режим онлайн-репликации.
  - **3** (RT_REMOTE_SNAPSHOT) — данные получаются от сервера репликации в режиме снэпшот. Используется для клиентов репликации, у которых БД не задана.
- **State** [out] enum TDataStreamState — состояние потока репликации. Возможны следующие состояния:
  - **0** (DS_STATE_CLOSE) — поток закрыт.
  - **1** (DS_STATE_LOCAL_SNAPSHOT) — поток в состоянии получения снэпшота из локальной БД клиента репликации.
  - **2** (DS_STATE_REMOTE_SNAPSHOT) — поток в состоянии получения снэпшота от сервера репликации.
  - **3** (DS_STATE_ONLINE) — поток в состоянии получения онлайн-данных от сервера репликации.
  - **4** (DS_STATE_CLOSE_COMPLETE) — поток закрыт после получения всех требуемых данных.
  - **5** (DS_STATE_REOPEN) — поток переоткрыт и клиент будет получать все данные заново. Возможно, например, в случае изменения прав клиента или изменения номера жизни схемы БД.
  - **6** (DS_STATE_ERROR) — ошибка.

7.1.2. Методы

7.1.2.1. Open

```
Open ([in] IP2Connection* conn);
```

Назначение

Открытие потока репликационных данных.

Аргументы

conn — указатель на интерфейс соединения.

7.1.2.2. Close

```
Close (void);
```

Назначение

Закрытие потока репликационных данных.

7.2. Интерфейс IP2DataStreamEvents

Интерфейс обратного вызова для обработки событий. Под событиями здесь понимается получение данных и изменение состояния потока данных.

7.2.1. Методы

7.2.1.1. StreamStateChanged

```
StreamStateChanged (
[in] IP2DataStream* stream,
[in] TDataStreamState newState);
```

Назначение

Нотификация об изменении состояния потока репликационных данных.

Аргументы

- **stream** — указатель на интерфейс объекта поток данных.
- **newState** — новое состояние потока.

7.2.1.2. StreamDataInserted

```
StreamDataInserted (
[in] IP2DataStream* stream,
[in] BSTR tableName,
[in] IP2Record* rec);
```

Назначение

Нотификация о вставке записи в БД.

Аргументы

- **stream** — указатель на интерфейс объекта поток данных.
- **tableName** — имя таблицы, в которую заносится запись.
- **rec** — указатель на интерфейс объекта запись.

7.2.1.3. StreamDataUpdated

```
StreamDataUpdated (
[in] IP2DataStream* stream,
[in] BSTR tableName,
[in] LONGLONG id,
[in] IP2Record* rec);
```

Назначение

Нотификация об изменении записи в БД.

Аргументы

- **stream** — указатель на интерфейс объекта поток данных.
- **tableName** — имя таблицы, в которой изменяется запись.
- **id** — идентификатор записи.
- **rec** — указатель на интерфейс объекта запись.

7.2.1.4. StreamDataDeleted

```
StreamDataDeleted (
[in] IP2DataStream* stream,
[in] BSTR tableName,
[in] LONGLONG id,
[in] IP2Record* rec);
```

Назначение

Нотификация об удалении записи из БД.

Аргументы

- **stream** — указатель на интерфейс объекта поток данных.
- **tableName** — имя таблицы, из которой удаляется запись.
- **id** — идентификатор записи.

- **rec** — указатель на интерфейс объекта запись.

8. TableSet

Объект предназначен для работы с клиентской схемой репликации.

Манипулируя объектами TableSet можно запрашивать у серверов репликации ограниченный набор таблиц из всего множества таблиц, доступного в потоке, а в каждой таблице – подмножество полей. Таким образом, можно подписываться только на те данные, которые необходимы клиенту. Объект **TableSet** содержит один единственный интерфейс — **IP2TableSet.**

Данный объект поддерживает стандартную технологию перечисления в COM (см. раздел **Приложение. Примеры сценариев**).

8.1. Интерфейс IP2TableSet

8.1.1. Свойства

- **FieldList** ([in] BSTR tableName) [in/out] BSTR — набор полей из заданной таблицы. Передавая имя таблицы можно получить строку, в которой перечисляются все поля в этой таблице.
- **Rev** ([in] BSTR tableName) [in/out] LONGLONG — максимальное значение служебного поля **Rev** (revision) в таблице. Свойство позволяет, как читать этот параметр, так и задавать его.

8.1.2. Методы

8.1.2.1. InitFromIni

```
InitFromIni (
[in] BSTR structFile,
[in] BSTR signFile);
```

Назначение

Инициализация объекта и загрузка из ini-файла клиентской схемы репликации. Если используется серверная схема, то загрузка клиентской схемы не требуется.

Аргументы

- **structFile** — файл, содержащий клиентскую схему репликации.
- **signFile** — не используется.

8.1.2.2. Count

```
Count ([in] LONG cnt);
```

Назначение

Получение числа таблиц, содержащихся в схеме репликации.

9. DataBuffer

Служебный объект, служащий для оптимизации работы с данными. Этот объект можно присоединить к потоку данных и в дальнейшем за хранение данных в памяти будет отвечать именно этот объект, а пользователь может использовать его для получения данных.

Данный объект поддерживает стандартную технологию перечисления в COM (см. раздел **Приложение. Примеры сценариев**).

Объект **DataBuffer** содержит один единственный интерфейс — **IP2DataBuffer**.

9.1. Интерфейс IP2DataBuffer

9.1.1. Свойства

TableRecords ([in] BSTR tableName) [out] IP2TableRecords — набор записей в указанной таблице.

9.1.2. Методы

9.1.2.1. AttachToStream

```
AttachToStream ([in] IP2DataStream* stream);
```

Назначение

Присоединение к потоку данных.

Аргументы

- **stream** — указатель на интерфейс потока данных.

9.1.2.2. DetachFromStream

```
DetachFromStream ();
```

Назначение

Отсоединение от потока данных.

9.1.2.3. CountTables

```
CountTables ([out,retval] LONG* tblCnt);
```

Назначение

Получение количества таблиц в потоке данных.

9.1.2.4. Count

```
Count (
[in] BSTR tableName,
[out, retval] LONG *tblCnt);
```

Назначение

Получение количества записей в указанной таблице.

Аргументы

tableName — имя таблицы.

9.1.2.5. Clear

```
Clear ([in] BSTR tableName);
```

Назначение

Удаление всех кешированных данных из указанной таблицы.

Аргументы

tableName — имя таблицы.

9.1.2.6. ClearAll

```
ClearAll ();
```

Назначение

Очистка всех таблиц в памяти.

10. Record

Объект предназначен для работы с записями. Объект **Record** содержит один единственный интерфейс — **IP2Record**.

10.1. Интерфейс IP2Record

10.1.1. Свойства

Count [out] ULONG — количество полей в записи.

10.1.2. Методы

10.1.2.1. GetValAsString

```
GetValAsString (
[in] BSTR fieldName,
[out,retval] BSTR* pVal);
```

Назначение

Получение значения поля как String по имени поля.

Аргументы

fieldName — имя поля.

10.1.2.2. GetValAsStringByIndex

```
GetValAsStringByIndex (
[in] ULONG fieldIndex,
[out,retval] BSTR* pVal);
```

Назначение

Получение значения поля как String по индексу поля.

Аргументы

fieldIndex — индекс поля.

10.1.2.3. GetValAsLong

```
GetValAsString (
[in] BSTR fieldName,
[out,retval] LONG* pVal);
```

Назначение

Получение значения поля как Long по имени поля.

Аргументы

fieldName — имя поля.

10.1.2.4. GetValAsLongByIndex

```
GetValAsStringByIndex (
[in] ULONG fieldIndex,
[out,retval] LONG* pVal);
```

Назначение

Получение значения поля как Long по индексу поля.

Аргументы

fieldIndex — индекс поля.

10.1.2.5. GetValAsShort

```
GetValAsString (
[in] BSTR fieldName,
```

```
[out,retval] SHORT* pVal);
```

Назначение

Получение значения поля как Short по имени поля.

Аргументы

fieldName — имя поля.

10.1.2.6. GetValAsShortByIndex

```
GetValAsStringByIndex (
[in] ULONG fieldIndex,
[out,retval] SHORT* pVal);
```

Назначение

Получение значения поля как Short по индексу поля.

Аргументы

fieldIndex — индекс поля.

10.1.2.7. GetValAsVariant

```
GetValAsString (
[in] BSTR fieldName,
[out,retval] VARIANT* pVal);
```

Назначение

Получение значения поля как Variant по имени поля. Ниже приведены правила мапирования типов данных.

u1 —> VT_U1

u2 —> VT_U2

u4 —> VT_U4

u8 —> VT_U8

i1 —> VT_I1

i2 —> VT_I2

i4 —> VT_I4

i8 —> VT_I8

c —> VT_BSTR

Аргументы

fieldName — имя поля.

10.1.2.8. GetValAsVariantByIndex

```
GetValAsStringByIndex (
[in] ULONG fieldIndex,
[out,retval] VARIANT* pVal);
```

Назначение

Получение значения поля как Variant по индексу поля.

Аргументы

fieldIndex — индекс поля.

Приложение. Примеры сценариев

Некоторые объекты библиотеки **P2ClientGate** поддерживают стандартный механизм коллекций и перечислений в COM (Enum). Ниже приведены примеры сценариев, иллюстрирующие использование этого механизма.

```
================================================================================
JScript
================================================================================
var n, x;
var fso = new ActiveXObject("Scripting.FileSystemObject");
// Create Enumerator on Drives.
var e = new Enumerator(fso.Drives);
for (;!e.atEnd();e.moveNext()) {      // Loop over the drives collection.
    x = e.item();
    if (x.DriveType == 3)              // See if network drive.
       n = x.ShareName;                // Get share name
    else if (x.IsReady)                // See if drive is ready.
       n = x.VolumeName;               // Get volume name.
    else
       n = "[Drive not ready]";
    WScript.Echo(x.DriveLetter + " - " + n);
}


================================================================================
VBScript
================================================================================
Set fso = CreateObject("Scripting.FileSystemObject")
Set e = fso.Drives
For Each x in e
     If x.DriveType = 3 Then
          n = x.ShareName
     ElseIf x.IsReady Then
         n = x.VolumeName
     Else
         n = "[Drive not ready]"
     End If
     WScript.Echo x.DriveLetter & " - " & n
Next


================================================================================
Visual Basic
================================================================================
// Declare variables.
var n, x;
var fso : ActiveXObject = new ActiveXObject("Scripting.FileSystemObject");
// Create Enumerator on Drives.
var e : Enumerator = new Enumerator(fso.Drives);
for (;!e.atEnd();e.moveNext()) {      // Loop over the drives collection.
    x = e.item();
    if (x.DriveType == 3)              // See if network drive.
       n = x.ShareName;                // Get share name
    else if (x.IsReady)                // See if drive is ready.
       n = x.VolumeName;               // Get volume name.
    else
       n = "[Drive not ready]";
    print(x.DriveLetter + " - " + n);
}
```