СИСТЕМА
УПРАВЛЕНИЯ
БАЗАМИ
ДАННЫХ

ЛИНТЕР®

ЛИНТЕР БАСТИОН

ЛИНТЕР СТАНДАРТ

Поставщики данных ADO.NET

НАУЧНО-ПРОИЗВОДСТВЕННОЕ ПРЕДПРИЯТИЕ

РЕЛЭКС

Товарные знаки

РЕЛЭКС™, ЛИНТЕР® являются товарными знаками, принадлежащими АО НПП «Реляционные экспертные системы» (далее по тексту – компания РЕЛЭКС). Прочие названия и обозначения продуктов в документе являются товарными знаками их производителей, продавцов или разработчиков.

Интеллектуальная собственность

Правообладателем продуктов ЛИНТЕР® является компания РЕЛЭКС (1990-2026). Все права защищены.

Данный документ является результатом интеллектуальной деятельности, права на который принадлежат компании РЕЛЭКС.

Все материалы данного документа, а также его части/разделы могут свободно размещаться на любых сетевых ресурсах при условии указания на них источника документа и активных ссылок на сайты компании РЕЛЭКС: relex.ru и linter.ru.

При использовании любого материала из данного документа несетевым/печатным изданием обязательно указание в этом издании источника материала и ссылок на сайты компании РЕЛЭКС: relex.ru и linter.ru.

Цитирование информации из данного документа в средствах массовой информации допускается при обязательном упоминании первоисточника информации и компании РЕЛЭКС.

Любое использование в коммерческих целях информации из данного документа, включая (но не ограничиваясь этим) воспроизведение, передачу, преобразование, сохранение в системе поиска информации, перевод на другой (в том числе компьютерный) язык в какой-либо форме, какими-либо средствами, электронными, механическими, магнитными, оптическими, химическими, ручными или иными, запрещено без предварительного письменного разрешения компании РЕЛЭКС.

О документе

Материал, содержащийся в данном документе, прошел доскональную проверку, но компания РЕЛЭКС не гарантирует, что документ не содержит ошибок и пропусков, поэтому оставляет за собой право в любое время вносить в документ исправления и изменения, пересматривать и обновлять содержащуюся в нем информацию.

Контактные данные

394006, Россия, г. Воронеж, ул. Бахметьева, 2Б.

Тел./факс: (473) 2-711-711, 2-778-333.

e-mail: info@linter.ru.

Техническая поддержка

С целью повышения качества программного продукта ЛИНТЕР и предоставляемых услуг в компании РЕЛЭКС действует автоматизированная система учёта и обработки пользовательских рекламаций. Обо всех обнаруженных недостатках и ошибках в программном продукте и/или документации на него просим сообщать нам в раздел Поддержка на сайте ЛИНТЕР.

Содержание

Предисловие 7
Назначение документа 7
Для кого предназначен документ 7
Необходимые предварительные знания 7
Дополнительные документы 7
Общие сведения 8
Назначение поставщика данных ADO.NET 8
Характеристики поставщика данных ADO.NET 8
Использование поставщика данных ADO.NET-провайдера в ОС Windows в среде разработки Visual Studio 8
Добавление локального источника данных nuget 8
Использование поставщика данных ADO.NET-провайдера в ОС Linux в среде разработки JetBrains Rider 10
Добавление локального источника данных nuget 10
Сборка и запуск клиентского приложения 11
Сборка приложения из командной строки 11
Запуск приложения из командной строки 11
Подготовка БД 12
Основные понятия и определения 13
Открытые классы провайдера 14
Класс LinterClientFactory 14
Конструктор 15
Свойства 15
CanCreateDataSourceEnumerator 15
Методы 15
CreateCommand 15
CreateCommandBuilder 17
CreateConnection 19
CreateConnectionStringBuilder 19
CreateDataAdapter 21
CreateDataSourceEnumerator 22
CreateParameter 24
Класс DbConnection 25
Конструкторы 27
LinterDbConnection() 27
LinterDbConnection(String) 27
LinterDbConnection(LinterDbConnection) 27
Свойства 28
ConnectionString 28
ConnectionTimeout 31
Database 32
DataSource 32
ServerVersion 33
State 33
Методы 34
BeginTransaction 34
BeginTransaction(IsolationLevel) 35
ChangeDatabase 37
Close 37
CreateCommand 39
EnlistTransaction 39
GetSchema 40
GetSchema(String) 46

GetSchema(String, String[ ]) 51
Open 53
События 55
StateChange 55
Класс DbCommand 55
Конструкторы 57
LinterDbCommand() 57
LinterDbCommand(String) 57
LinterDbCommand(String, LinterDbConnection) 58
LinterDbCommand(String, LinterDbConnection, LinterDbTransaction) 58
Свойства 59
CommandText 59
CommandTimeout 64
CommandType 65
Connection 65
DesignTimeVisible 66
Parameters 67
Transaction 67
UpdatedRowSource 68
Методы 71
Cancel 71
CreateParameter 73
ExecuteNonQuery 74
ExecuteReader 81
ExecuteReader(CommandBehavior) 85
ExecuteScalar 88
Prepare 89
Класс DbDataReader 91
Свойства 94
Depth 94
FieldCount 94
HasRows 95
IsClosed 97
Item(Int32) 98
Item(String) 99
RecordsAffected 100
VisibleFieldCount 102
IsBeforeReadState 102
Методы 104
Close 104
GetBoolean 106
GetByte 107
GetBytes 108
GetChar 112
GetChars 115
GetData 117
GetDataTypeName 118
GetDateTime 119
GetDecimal 121
GetDouble 122
GetEnumerator 124
GetFieldType 125
GetFloat 127
GetGuid 128
GetInt16 130

GetInt32 131
GetInt64 133
GetLinterBlobForUpdate 134
GetName 136
GetOrdinal 138
GetProviderSpecificFieldType 139
GetProviderSpecificValue 140
GetProviderSpecificValues 140
GetSchemaTable 141
FastGetSchemaTable 145
GetString 145
GetValue 147
GetValues 148
IsDBNull 150
NextResult 151
Read 153
Класс DbTransaction 154
Свойства 155
Connection 155
IsolationLevel 155
Методы 156
Commit 156
Commit(String) 159
Rollback 160
Rollback(String) 161
Save(String) 163
Класс DbParameter 163
Конструкторы 165
LinterDbParameter() 165
LinterDbParameter(String, Object) 165
LinterDbParameter(String, ELinterDbType) 166
LinterDbParameter(String, ELinterDbType, Int) 166
LinterDbParameter(String, ELinterDbType, Int, String) 167
LinterDbParameter(String, ELinterDbType, Int, ParameterDirection, Bool, Byte, Byte, String, DataRowVersion, Object) 167
Свойства 168
DbType 168
Direction 171
IsNullable 172
ParameterName 173
Size 174
SourceColumn 175
SourceColumnNullMapping 177
SourceVersion 178
Value 181
LinterDbType 182
Precision 183
Scale 183
Методы 184
ResetDbType 184
Класс DbParameterCollection 185
Свойства 187
Count 187
IsFixedSize 188
IsReadOnly 189

IsSynchronized ... 189
Item(Int32) ... 191
Item(String) ... 192
SyncRoot ... 193
Методы ... 194
Add(Object) ... 194
Add(String, Object) ... 195
Add(String, ELinterDbType) ... 196
Add(String, ELinterDbType, Int32) ... 198
Add(String, ELinterDbType, Int32, String) ... 200
Add(LinterDbParameter) ... 201
AddRange ... 202
Clear ... 204
Contains(String) ... 204
Contains(Object) ... 205
CopyTo ... 207
GetEnumerator ... 208
IndexOf(String) ... 209
IndexOf(Object) ... 211
Insert ... 212
Remove ... 213
RemoveAt(Int32) ... 214
RemoveAt(String) ... 216
Класс DbDataAdapter ... 217
Конструкторы ... 220
LinterDbDataAdapter() ... 220
LinterDbDataAdapter(LinterDbCommand) ... 221
LinterDbDataAdapter(String, LinterDbConnection) ... 221
LinterDbDataAdapter(String, String) ... 222
Свойства ... 223
AcceptChangesDuringFill ... 223
AcceptChangesDuringUpdate ... 224
ContinueUpdateOnError ... 227
DeleteCommand ... 229
FillLoadOption ... 231
InsertCommand ... 234
MissingMappingAction ... 236
MissingSchemaAction ... 238
ReturnProviderSpecificTypes ... 241
SelectCommand ... 242
TableMappings ... 245
UpdateBatchSize ... 248
UpdateCommand ... 251
Методы ... 254
Fill(DataSet) ... 254
Fill(DataTable) ... 257
Fill(DataSet, String) ... 259
Fill(Int32, Int32, DataTable) ... 260
Fill(DataSet, Int32, Int32, String) ... 263
FillSchema(DataSet, SchemaType, String) ... 266
FillSchema(DataSet, SchemaType) ... 271
FillSchema(DataTable, SchemaType) ... 272
GetFillParameters ... 273
Update(DataRow[]) ... 275
Update(DataSet) ... 278

Update(DataTable) 280
Update(DataSet, String) 282
События 284
FillError 284
RowUpdating 287
RowUpdated 287
Класс DbCommandBuilder 290
Конструкторы 295
LinterDbCommandBuilder 295
LinterDbCommandBuilder(LinterDbDataAdapter) 295
Свойства 295
CatalogLocation 295
CatalogSeparator 296
ConflictOption 296
DataAdapter 299
QuotePrefix 301
QuoteSuffix 301
SchemaSeparator 302
SetAllValues 303
Методы 306
DeriveParameters 306
GetDeleteCommand 309
GetDeleteCommand(Boolean) 310
GetInsertCommand 312
GetInsertCommand(Boolean) 313
GetUpdateCommand 315
GetUpdateCommand(Boolean) 316
QuoteIdentifier 318
RefreshSchema 320
UnquoteIdentifier 321
Класс DbConnectionStringBuilder 322
Конструкторы 324
LinterDbConnectionStringBuilder 324
LinterDbConnectionStringBuilder(String) 324
Свойства 325
BrowsableConnectionString 325
ConnectionString 326
Count 328
DataSource 329
IsFixedSize 330
IsReadOnly 331
Item 331
Keys 333
Password 335
PersistSecurityInfo 336
UserID 337
Values 339
Методы 341
Add 341
AppendKeyValuePair(StringBuilder, String, String) 344
AppendKeyValuePair(StringBuilder, String, String, Boolean) 345
Clear 347
ContainsKey 349
EquivalentTo 351
Remove 352

ShouldSerialize ... 353
TryGetValue ... 354
Класс LinterBlob ... 356
Конструкторы ... 356
Свойства ... 356
Методы ... 356
Append ... 356
Clear ... 357
Класс LinterSqlException ... 357
Конструкторы ... 360
Свойства ... 360
Data ... 360
ErrorCode ... 363
Errors ... 364
HelpLink ... 365
InnerException ... 365
Message ... 366
Number ... 369
Source ... 369
SqlLineNumber ... 369
SqlPositionInLine ... 370
StackTrace ... 370
TargetSite ... 371
Методы ... 371
GetBaseException ... 371
GetObjectData ... 372
Общие свойства и методы классов ADO.NET-провайдера ... 374
Общие свойства ... 374
Container ... 374
Site ... 375
Общие методы ... 375
Dispose ... 376
Equals ... 377
GetHashCode ... 379
GetLifetimeService ... 381
GetType ... 383
InitializeLifetimeService ... 386
ToString ... 387
Обработка событий ... 391
Обработка исключений ... 394
Прерывание запроса ... 396
Провайдер Entity Framework 6 ... 397
Строки подключения ... 399
Отображение типов данных ... 400
Использование хранимых процедур для выполнения операций INSERT|UPDATE|DELETE ... 401
Приложение 1. Освобождение ресурсов ... 405
Приложение 2. Пример асинхронной обработки данных ... 408

Предисловие

Назначение документа

Документ содержит описание поставщиков данных (провайдеров) ADO.NET и Entity Framework 6 (EF6) для СУБД ЛИНТЕР.

Для каждого класса поставщиков данных даётся информация об его соответствии общей спецификации `ADO.NET/EF6` и об особенностях реализации для СУБД ЛИНТЕР.

Документ предназначен для СУБД ЛИНТЕР БАСТИОН 6.0 сборка 20.8, далее по тексту СУБД ЛИНТЕР.

Для кого предназначен документ

Документ предназначен для программистов, разрабатывающих приложения с использованием реляционных баз данных на платформе .NET.

Необходимые предварительные знания

Для работы с ADO.NET-провайдером необходимо знать:

- основы реляционных баз данных;
- язык баз данных SQL СУБД ЛИНТЕР;
- спецификации ADO.NET;
- язык программирования C#;
- уметь работать в операционной системе на уровне пользователя.

Дополнительные документы

- Архитектура СУБД
- Сетевой администратор
- Сетевые средства
- Создание и конфигурирование базы данных
- Справочник по SQL
- Интерфейс нижнего уровня
- Системные таблицы и представления
- Справочник кодов завершения
- Запуск и останов СУБД ЛИНТЕР в среде ОС Windows
- Запуск и останов СУБД ЛИНТЕР в среде ОС Linux

Общие сведения

Назначение поставщика данных ADO.NET

Поставщик данных ADO.NET для СУБД ЛИНТЕР – это интерфейс для предоставления пользователю доступа к данным, хранящимся в БД ЛИНТЕР.

Поставщик данных ADO.NET поддерживает версии .NET 6, 7, 8, 9. Поставщик данных ADO.NET используется провайдером данных ORM Entity Framework 6.

Примечание
Если необходимая версия .NET отсутствует в перечне поддерживаемых версий, следует обратиться в раздел Поддержка на сайте ЛИНТЕР.

Характеристики поставщика данных ADO.NET

Поставщик данных обеспечивает:

1) передачу клиентских SQL-запросов к СУБД ЛИНТЕР;
2) выполнение SQL-запросов, базирующихся на спецификациях X/Open и SQL Access Group (SAG) SQL CAE 1992 года;
3) предоставление клиентским приложениям результатов обработки SQL-запросов;
4) предоставление клиентским приложениям кодов завершения обработки SQL-запросов;
5) поддержку стандартных типов данных;
6) статическое и динамическое формирование SQL-предложений;
7) прием и передачу значений данных в формате, задаваемом клиентским приложением.

Использование поставщика данных ADO.NET-провайдера в ОС Windows в среде разработки Visual Studio

Добавление локального источника данных nuget

Примечание
Для корректной работы ADO.NET-провайдера необходимы установленные в системе клиентские библиотеки низкого уровня Linter

Для использования поставщиков данных ADO.NET и EF6 необходимо предварительно добавить локальный путь, по которому располагаются NuGet-пакеты `EntityFramework.Linter.6.0.20.2.nupkg` и `System.Data.LinterClient.6.0.20.2.nupkg` в источники данных NuGet. Для этого в среде Visual Studio необходимо зайти в пункт меню **Tools=>Options**, в появившемся окне открыть раздел **NuGet Package Manager=>Package Sources**, добавить новый источник, задать имя (например, "Linter") и в параметре Source указать путь до расположения NuGet-пакетов провайдеров. (рис. 1).

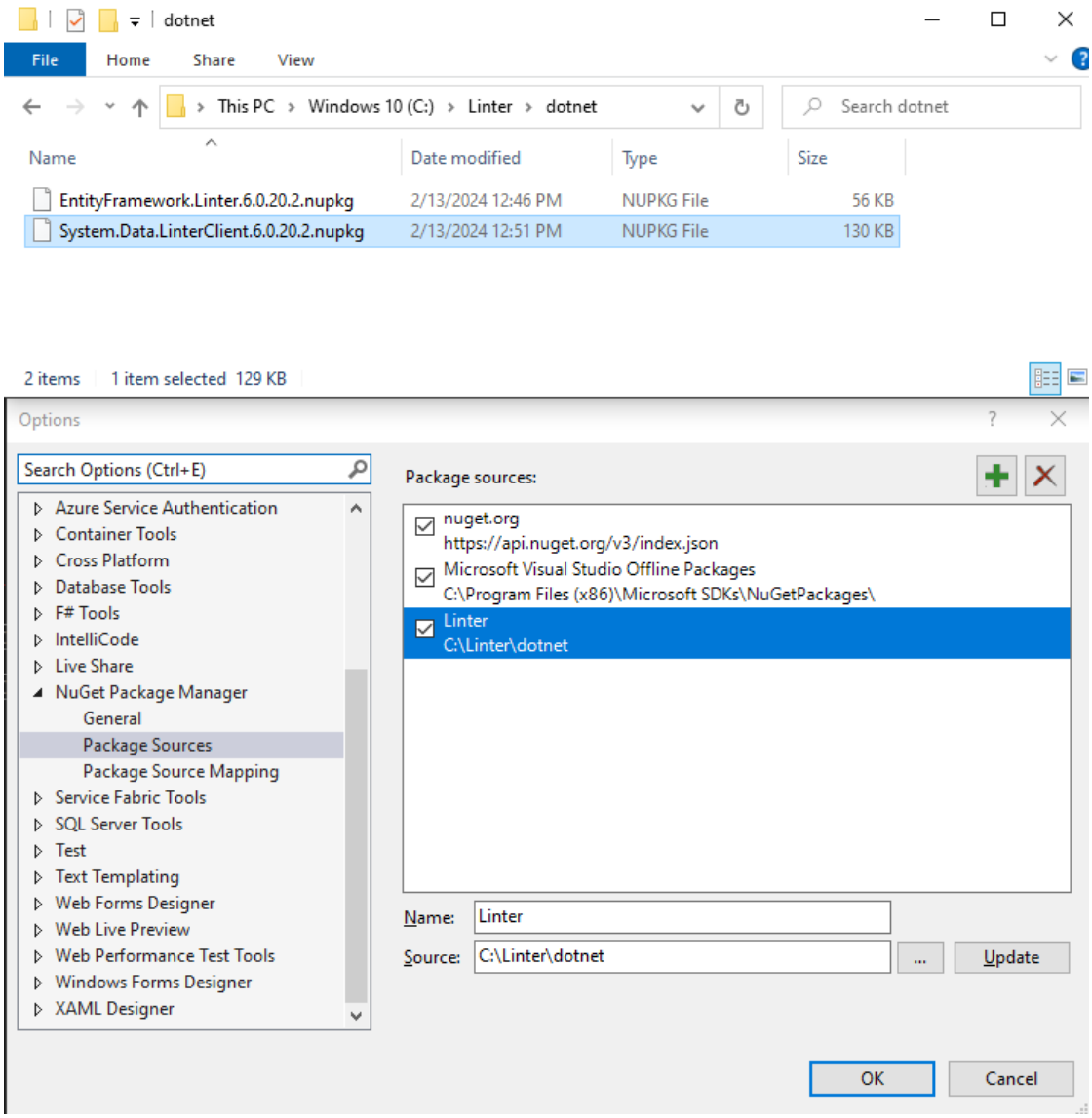


Рисунок 1. Диалог NuGet Package Sources

После этого, для использования поставщиков данных ADO.NET и EF6 в любом .NET-проекте достаточно добавить соответствующий пакет к проекту стандартными средствами, выбрав созданный ранее локальный источник пакетов NuGet (например, "Linter", рис. 2).

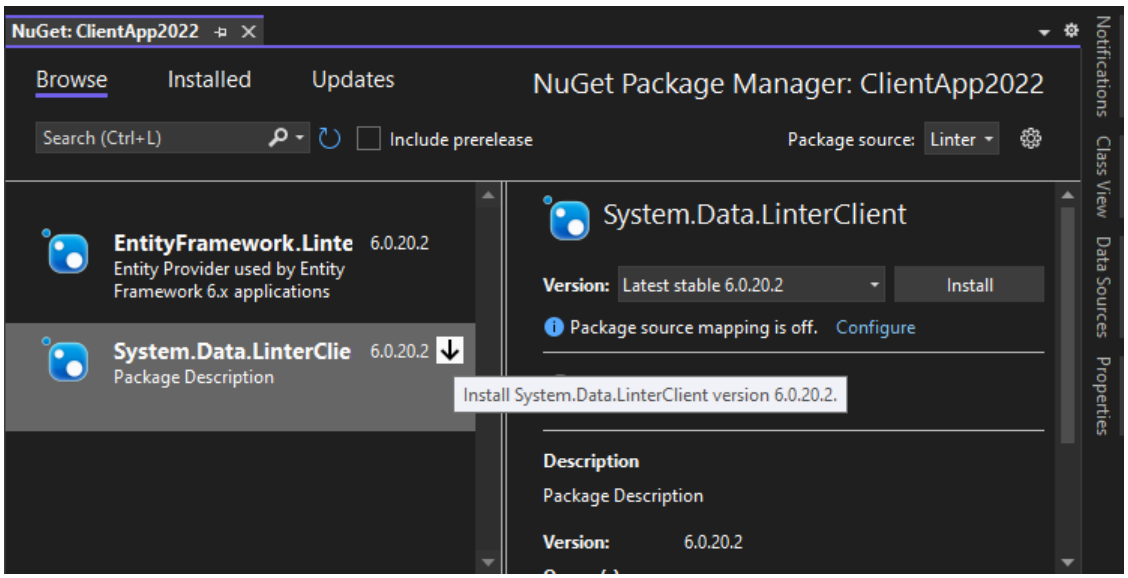


Рисунок 2. Диалог NuGet Package Manager

Использование поставщика данных ADO.NET-провайдера в ОС Linux в среде разработки JetBrains Rider

Добавление локального источника данных nuget

Примечание
Для корректной работы ADO.NET-провайдера необходимы установленные в системе клиентские библиотеки низкого уровня Linter

Для использования поставщиков данных ADO.NET и EF6 необходимо предварительно добавить локальный путь, по которому располагаются NuGet-пакеты `EntityFramework.Linter.6.0.20.2.nupkg` и `System.Data.LinterClient.6.0.20.2.nupkg` в источники данных NuGet. Для этого в среде Rider необходимо зайти в пункт меню **Tools=>NuGet=>Manage NuGet Packages for Solution**, в появившемся окне открыть раздел **Sources**, добавить новый источник, задать имя (например, "Linter") и в параметре Source указать путь до расположения NuGet-пакетов провайдеров. (рис. 3).

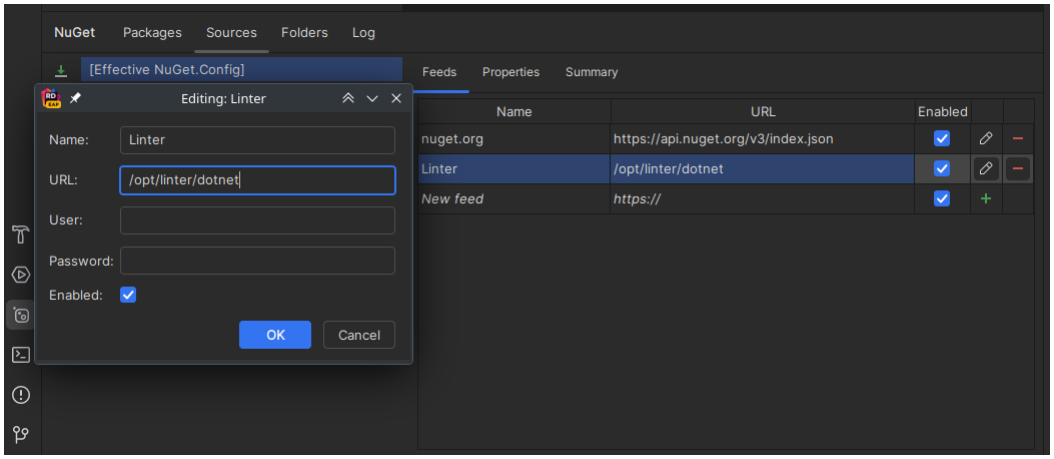


Рисунок 3. Диалог Rider NuGet Sources

После этого, для использования поставщиков данных ADO.NET и EF6 в любом .NET-проекте достаточно добавить соответствующий пакет к проекту стандартными средствами, выбрав созданный ранее локальный источник пакетов NuGet (например, "Linter", рис. 4).

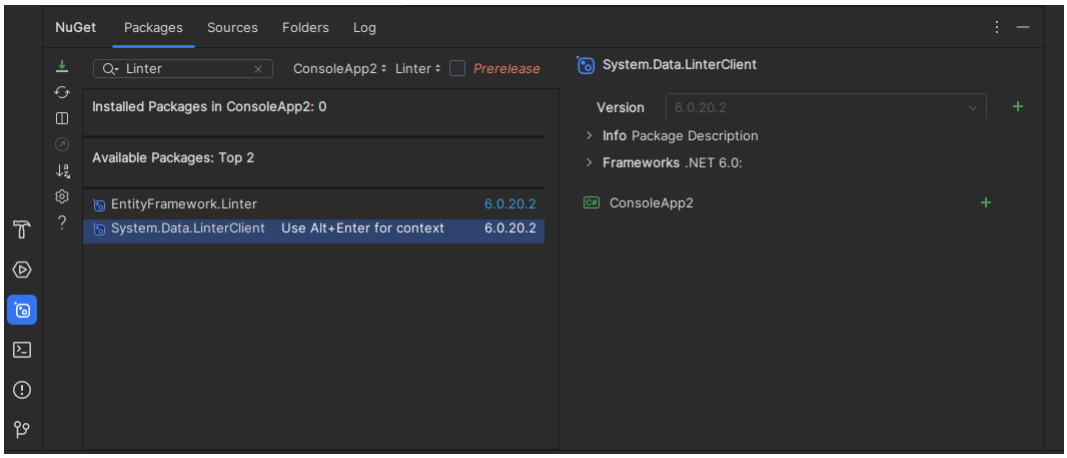


Рисунок 4. Диалог Rider NuGet Packages

Сборка и запуск клиентского приложения

Сборка приложения из командной строки

Для сборки клиентского приложения, разработанного без применения инструментальных средств, можно использовать построение из командной строки с помощью CLI-команды `dotnet build` (https://learn.microsoft.com/ru-ru/dotnet/core/tools/).

Запуск приложения из командной строки

Для запуска клиентского приложения, разработанного без применения инструментальных средств, можно использовать запуск из командной строки с помощью CLI-команды dotnet run (https://learn.microsoft.com/ru-ru/dotnet/core/tools/).

Для запуска приложения необходимо добавить в окружение путь к библиотекам из каталога bin установочного каталога СУБД ЛИНТЕР. В случае, если библиотеки не будут найдены в окружении, dotnet выдаст сообщение вида:

```
Unable to load DLL 'InterLib' or one of its dependencies
```

Пример сборки и запуска для ОС типа Linux:

```
dotnet build
LD_LIBRARY_PATH=/usr/linter/bin PATH=$PATH:/usr/linter/bin dotnet
 run
```

Пример сборки и запуска для ОС типа Windows:

```
dotnet build
set Path=%Path%;C:\Linter\bin
dotnet run
```

Подготовка БД

Для работы ADO.NET-провайдера необходимы системные представления, которые не создаются автоматически при создании БД. Поэтому после создания БД надо выполнить SQL-скрипты из подкаталога `/dict` установочного каталога СУБД ЛИНТЕР в следующем порядке:

1) systab.sql;
2) catalog.sql;
3) catalog_oledb.sql;
4) cstables.sql;
5) charsets.sql;
6) ora_cat.sql.

Если при работе ADO.NET-провайдера генерируется исключение типа LinterSqlException с полем Number, равным 1407 «Сортируемая запись не помещается в странице» или 1504 «Длина строки больше чем длина страницы», то надо выполнить следующий запрос:

```
ALTER DATABASE SET RECORD SIZE LIMIT 65535;
```

и перезапустить ядро СУБД ЛИНТЕР.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Основные понятия и определения

Открытые классы – общедоступные (public) классы провайдера.

Закрытые классы – внутренние (internal) классы провайдера, недоступные разработчикам клиентских приложений.

Общие свойства и методы – свойства и методы, присущие большинству классов ADO.NET-провайдера. Они описаны в отдельном разделе «Общие свойства и методы классов ADO.NET-провайдера».

Открытые классы провайдера

В ADO.NET-провайдере СУБД ЛИНТЕР версии 2.0 пользовательские классы для обработки данных наследуются от открытых классов, определённых в пространстве имен `System.Data.Common`. Разработчики клиентских приложений могут создавать свои собственные, специфические для используемого ADO.NET-провайдера, экземпляры этих классов с помощью фабрики классов `LinterClientFactory`.

Фабрика классов провайдера позволяет генерировать программный код доступа к различным источникам данных с минимальными затратами на кодирование. С помощью класса `LinterClientFactory` можно создать экземпляры следующих классов:

Имя класса	Функциональное назначение класса
DbCommand	Формирование запроса к источнику данных
DbCommandBuilder	Построитель команд соединения с источником данных
DbConnection	Соединение с источником данных
DbConnectionStringBuilder	Формирование строки подключения к источнику данных
DbDataAdapter	Автономная работа с данными
DbDataReader	Выборка данных
DbParameter	Определение типов данных и значений параметров хранимых процедур и параметризованных запросов
DbParameterCollection	Управление параметрами (добавление/удаление) хранимых процедур и параметризованных запросов
DbTransaction	Управление транзакциями
LinterBlob	Работа с BLOB-данными
LinterClientFactory	Фабрика классов провайдера
LinterSqlException	Код завершения СУБД ЛИНТЕР

Класс LinterClientFactory

Класс `LinterClientFactory` является фабрикой объектов, которая дает возможность создавать экземпляры других классов для поставщика данных .NET.

Свойства класса приведены в таблице 1.

Таблица 1. Свойства класса LinterClientFactory

Свойство	Описание
CanCreateDataSourceEnumerator	Индикатор поддержки класса `DbDataSourceEnumerator`.

Методы класса приведены в таблице 2.

Таблица 2. Методы класса LinterClientFactory

Метод	Описание
CreateCommand	Создает экземпляр класса `LinterDbCommand`, используемый в дальнейшем для формирования текста SQL-запроса к серверу источника данных.

© Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Метод	Описание
CreateCommandBuilder	Создает экземпляр класса `LinterDbCommandBuilder` (построитель команд), унаследованный от абстрактного класса `DbCommandBuilder`.
CreateConnection	Создает экземпляр класса `LinterDbConnection`, используемый для соединения с сервером источника данных.
CreateConnectionStringBuilder	Создает объект типа `LinterDbConnectionStringBuilder`.
CreateDataAdapter	Создает экземпляр класса `LinterDbDataAdapter`.
CreateDataSourceEnumerator	Создает объект типа `LinterDbDataSourceEnumerator`, с помощью которого можно получить список (перечисление) всех доступных ADO.NET-провайдеру источников данных (для возможного последующего подключения к ним).
CreateParameter	Создает экземпляр класса `LinterDbParameter`, используемый для определения типов данных и значений параметров хранимых процедур или параметризованных SQL-операторов.

Конструктор

Конструктор класса отсутствует. Инициализация класса выполняется с помощью конструкции:

```
DbProviderFactories.RegisterFactory("System.Data.LinterClient",
 typeof(LinterClientFactory));
var factory =
 DbProviderFactories.GetFactory("System.Data.LinterClient");
```

Свойства

CanCreateDataSourceEnumerator

Декларация

```
public override bool CanCreateDataSourceEnumerator {get;};
```

Значение свойства

True, если экземпляр класса `DbProviderFactory` поддерживает класс `DbDataSourceEnumerator`; в противном случае – false.

Методы

CreateCommand

Метод создает экземпляр класса `LinterDbCommand`, используемый в дальнейшем для формирования текста SQL-запроса к СУБД ЛИНТЕР. Созданный объект `LinterDbCommand` необходимо связывать с конкретным соединением (их может быть несколько) с ЛИНТЕР-сервером.

Синтаксис

```
public override DbCommand CreateCommand();
```

Возвращаемое значение

Объект LinterDbCommand типа System.Data.LinterClient.LinterDbCommand.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CreateCommandSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта LinterDbCommand, связанного с
 установленным соединением
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "select make, model from auto";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    while (reader.Read())
    {
      Console.WriteLine(String.Format("{0}, {1}", reader[0],
 reader[1]));
    }
    // Освобождение ресурсов
    reader.Close();
    con.Close();
  }
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
}
```

CreateCommandBuilder

Метод создает экземпляр класса `LinterDbCommandBuilder` (построитель команд), унаследованный от абстрактного класса `DbCommandBuilder`.

Класс `LinterDbCommandBuilder` автоматически генерирует для каждой таблицы, задействованной в наборе данных (`DataSet`) необходимые SQL-операторы для синхронизации этих изменений с БД.

Синтаксис

```
public override DbCommandBuilder CreateCommandBuilder();
```

Возвращаемое значение

Объект `LinterDbCommandBuilder` типа `System.Data.LinterClient.LinterDbCommandBuilder`.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CreateCommandBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    DbCommand cmd = factory.CreateCommand();
    cmd.CommandText = "select * from auto";
    cmd.Connection = con;
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = cmd;
    // Связывание объектов DbDataAdapter и DbCommandBuilder
    // Объект DbDataAdapter не создает автоматически SQL-
операторы, необходимые для
```

```
    // согласования изменений, внесенных в объект DataSet. Однако,
 если задано
    // свойство SelectCommand объекта DbDataAdapter, то можно
 создать объект
    // DbCommandBuilder, который будет автоматически создавать
 SQL-операторы для
    // однотабличных обновлений. В этом случае необходимо
 установить свойство
    // DataAdapter класса DbCommandBuilder чтобы объект
 DbCommandBuilder
    // зарегистрировал себя в качестве слушателя для события
 RowUpdating.
    // Одновременно можно связать друг с другом только один объект
 DbDataAdapter или
    // DbCommandBuilder.
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Изменение свойства SelectCommand
    // Для создания операторов INSERT, UPDATE или DELETE объект
 DbCommandBuilder
    // использует свойство SelectCommand для автоматического
 извлечения нужного
    // набора метаданных. Если изменить свойство SelectCommand
 после получения
    // метаданных (например, после первого обновления), необходимо
 обновить
    // метаданные путем вызова метода RefreshSchema. Кроме того,
 свойство
    // SelectCommand должно возвращать по крайней мере один
 первичный ключ или
    // уникальный столбец. Если таковые отсутствуют, то создается
 исключение
    // InvalidOperationException и команды не создаются.
    cmd.CommandText = "select * from person";
    // Обновление схемы
    builder.RefreshSchema();
    // Освобождение ресурсов
    // Если вызывается метод Dispose, объект DbCommandBuilder
 теряет связь с объектом
    // DbDataAdapter и созданные команды в дальнейшем не
 используются.
    builder.Dispose();
    // Закрытие подключения к БД
    con.Close();
  }
}
```

CreateConnection

Метод создаёт экземпляр класса `LinterDbConnection`, используемый для соединения с ЛИНТЕР-сервером. Для параллельной работы с несколькими ЛИНТЕР-серверами должны использоваться разные объекты `LinterDbConnection`.

Синтаксис

```
public override DbConnection CreateConnection();
```

Возвращаемое значение

Объект `LinterDbConnection` типа `System.Data.LinterClient.LinterDbConnection`.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CreateConnectionSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    Console.WriteLine("Версия сервера: {0}", con.ServerVersion);
    Console.WriteLine("Состояние: {0}", con.State);
    // Освобождение ресурсов
    con.Close();
  }
}
```

CreateConnectionStringBuilder

Метод создает объект типа `LinterDbConnectionStringBuilder`.

Класс `LinterDbConnectionStringBuilder` позволяет разработчикам приложений задавать в исходном коде программы произвольные пары «ключ/значение» и передавать полученную результирующую строку подключения провайдеру данных.

Разработчик приложения может создавать, назначать и изменять строки подключения. Для этого построитель строк подключения предоставляет строго типизированные свойства, соответствующие известным парам «ключ/значение». Чтобы обеспечить поддержку неизвестных значений, разработчики приложений могут также предоставлять произвольные пары «ключ/значение».

Класс `LinterDbConnectionStringBuilder` также может использоваться для управления строками подключения, которые хранятся в файле конфигурации приложения.

Разработчики могут создавать строки подключения, используя либо строго типизированный класс построителя строк подключения `LinterDbConnectionStringBuilder`, либо класс `DbConnectionStringBuilder`.

Класс `LinterDbConnectionStringBuilder` не выполняет проверок на наличие допустимых пар «ключ/значение». Следовательно, он допускает создавать недопустимые строки подключения.

Синтаксис

```
public override DbConnectionStringBuilder
 CreateConnectionStringBuilder();
```

Возвращаемое значение

Объект `LinterDbConnectionStringBuilder` типа `System.Data.LinterClient.LinterDbConnectionStringBuilder`.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Data.LinterClient;

class CreateConnectionStringBuilderSample
{
  static void Main()
  {
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    DbConnectionStringBuilder builder =
      factory.CreateConnectionStringBuilder();
    // Класс DbConnectionStringBuilder не зависит от баз данных,
 поэтому допускает
    // конструирование любых произвольных строк подключения.
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    builder.ConnectionString = "User
 ID=SYSTEM;Password=MANAGER8;Data Source=LOCAL";
    // Установка уровня изоляции транзакций.
    builder.Add("Isolation Level", IsolationLevel.ReadCommitted);
    // С помощью класса DbConnectionStringBuilder может быть
 сформировано свойство
    // ConnectionString.
    Console.WriteLine(builder.ConnectionString);
    // Используем тот же самый объект DbConnectionStringBuilder
 для создания объекта
    // LinterDbConnection.
    builder.Clear();
    builder.Add("User ID", "SYSTEM");
    builder.Add("Password", "MANAGER8");
    builder.Add("Data Source", "LOCAL");
    LinterDbConnection linterDbConnection = new
      LinterDbConnection(builder.ConnectionString);
    Console.WriteLine(linterDbConnection.ConnectionString);
    // Передавая объекту DbConnectionStringBuilder готовую строку
 подключения, можно
    // получить и изменить любой элемент.
    builder.ConnectionString = "User
 ID=SYSTEM;Password=MANAGER8;Data Source=LOCAL";
    builder["Data Source"] = "SERV1";
    builder.Remove("User ID");
    // Обратите внимание, что вызов метода Remove для
 несуществующего элемента, не
    // приводит к ошибке.
    builder.Remove("BadItem");
    // Установка значения добавляет новый элемент при
 необходимости.
    builder["Charset"] = "CP1251";
    builder.Remove("password");
    builder["User ID"] = "Hello";
    Console.WriteLine(builder.ConnectionString);
  }
}
```

CreateDataAdapter

Метод создает экземпляр класса `LinterDbDataAdapter`.

Класс `LinterDbDataAdapter` наследуется от класса `DbDataAdapter` и предназначен для работы с отсоединенным набором данных `DataSet` (раздел Класс DbDataAdapter)

Синтаксис

```
public override DbDataAdapter CreateDataAdapter();
```

Возвращаемое значение

Объект `LinterDbDataAdapter` типа `System.Data.LinterClient.LinterDbDataAdapter`.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CreateDataAdapterSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с ЛИНТЕР-сервером
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта LinterDbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.CommandText = "select * from auto";
    cmd.Connection = con;
    // Создание объекта LinterDbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = cmd;
    // Создание набора данных DataSet
    DataSet dataset = new DataSet();
    // Заполнение набора данных
    adapter.Fill(dataset);
    // Закрытие подключения к БД
    con.Close();
  }
}
```

CreateDataSourceEnumerator

Метод создает объект типа `LinterDbDataSourceEnumerator`, с помощью которого можно получить список (перечисление) всех доступных ADO.NET-провайдеру источников данных (для возможного последующего подключения к ним).

Сам список источников данных предоставляется методом `GetDataSources` в виде таблицы 3.

Примечание
В текущей версии провайдера метод возвращает пустой объект `DataTable`.

Таблица 3. Структура таблицы с информацией о поддерживаемых источниках данных

Порядковый номер столбца	Имя столбца	Тип данных столбца	Значение столбца
0	`ServerName`	`System.String`	Имя сервера источника данных (LOCAL – для локального сервера, имя ЛИНТЕР-сервера из конфигурационного файла `nodetab` – для удаленного сервера)
1	`InstanceName`	`System.String`	Имя экземпляра сервера. В зависимости от поставщика может существовать несколько экземпляров на одном сервере
2	`IsClustered`	`System.String`	Указывает, является ли сервер частью кластера **Примечание** В СУБД ЛИНТЕР не поддерживается
3	`Version`	`System.String`	Версия сервера в формате <версия><релиз><номер сборки>, например: 6.0.17.48
4	`FactoryName`	`System.String`	Имя соответствующей фабрики поставщиков
5	`ServiceName`	`System.String`	Имя сетевого драйвера
6	`Protocol`	`System.String`	Имя сетевого протокола используемого для доступа к источнику данных, например, TCPIP
7	`Port`	`System.String`	Сетевой адрес удаленного сервера

Синтаксис

```
public override DbDataSourceEnumerator
 CreateDataSourceEnumerator();
```

Возвращаемое значение

Объект `DbDataSourceEnumerator` типа `System.Data.Common.DbDataSourceEnumerator`.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
class CreateDataSourceEnumeratorSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    if (factory.CanCreateDataSourceEnumerator)
    {
      DbDataSourceEnumerator dsenum =
 factory.CreateDataSourceEnumerator();
      DataTable dt = dsenum.GetDataSources();
      // Отобразить параметры всех поддерживаемых источников
 данных
      foreach (DataRow dr in dt.Rows)
      {
        Console.WriteLine(dt.Columns[0] + " : " + dr[0]);
        Console.WriteLine(dt.Columns[1] + " : " + dr[1]);
        Console.WriteLine(dt.Columns[2] + " : " + dr[2]);
        Console.WriteLine(dt.Columns[3] + " : " + dr[3]);
      }
    }
    else
    {
      Console.Write("Перечисление источников данных провайдером не
 поддерживается ");
    }
  }
}
```

CreateParameter

Метод создает экземпляр класса `LinterDbParameter`, используемый для определения типов данных и значений параметров хранимых процедур или параметризованных SQL-операторов (см. также раздел Класс DbParameterCollection).

Синтаксис

```
public override DbParameter CreateParameter();
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Возвращаемое значение

Объект `LinterDbParameter` типа `System.Data.LinterClient.LinterDbParameter`.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CreateParameterSample
{
  static void Main()
  {
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    DbParameter p = factory.CreateParameter();
    p.ParameterName = ":PNAME";
    p.DbType = DbType.String;
    p.Size = 80;
    p.Value = "Значение параметра";
    Console.WriteLine(p.Value);
  }
}
```

Класс DbConnection

Класс `DbConnection` устанавливает соединение клиентского приложения с источником данных. С помощью свойств этого класса можно задать тип источника данных, его местоположение и некоторые другие атрибуты. Класс `DbConnection` выступает в качестве канала, по которому другие классы, например, `DbDataAdapter` или `DbCommand`, взаимодействуют с СУБД ЛИНТЕР при обработке SQL-операторов.

Для освобождения объекта DbConnection надо использовать оператор `using` или блок `try...finally` (см. приложение 1).

Конструкторы класса приведены в таблице 4.

Таблица 4. Конструкторы класса DbConnection

Конструктор	Описание
LinterDbConnection()	Инициализация объекта класса `LinterDbConnection`.
LinterDbConnection(String)	Инициализация объекта класса `LinterDbConnection` с заданной строкой соединения с источником данных.

Конструктор	Описание
LinterDbConnection(LinterDbConnection)	Инициализация объекта класса `LinterDbConnection` с помощью предварительно созданного объекта `LinterDbConnection`.

Свойства класса приведены в таблице 5.

Таблица 5. Свойства класса DbConnection

Свойство	Описание
ConnectionString	Значение строки подключения с источником данных.
ConnectionTimeout	Тайм-аут соединения с источником данных.
Database	Имя БД источника данных
DataSource	Имя сервера источника данных, с которым установлено соединение.
ServerVersion	Номер версии сервера источника данных, с которым установлено соединение.
State	Индикатор текущего состояния соединения с источником данных

Методы класса приведены в таблице 6.

Таблица 6. Методы класса DbConnection

Метод	Описание
BeginTransaction	Начинает транзакцию по заданному соединению.
BeginTransaction(IsolationLevel)	Начинает транзакцию по заданному соединению с указанным уровнем изоляции транзакции.
ChangeDatabase	Меняет текущее соединение с источником данных для последующей установки нового соединения.
Close	Закрывает соединение с текущим источником данным.
CreateCommand	Создает объект `DbCommand`, связанный с текущим соединением.
EnlistTransaction	Выполняет ручное прикрепление транзакции в текущем соединении к распределенной транзакции. **Примечание** В текущей версии ADO.NET-провайдера метод не поддерживается.
GetSchema	Предоставляет список (коллекцию) всех поддерживаемых источником данных объектов БД в текущем соединении.
GetSchema(String)	Предоставляет метаданные указанных объектов БД (коллекции данных), связанных с текущим соединением.
GetSchema(String, String[ ])	Предоставляет метаданные в соответствии с запрошенными атрибутами указанного объекта

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Метод	Описание
	БД (коллекции данных), связанного с текущим соединением.
Open	Открывает соединение с источником данных в соответствии с параметрами, указанными в строке подключения.

События класса приведены в таблице 7.

Таблица 7. События класса DbConnection

Событие	Описание
StateChange	Генерируется при изменении состояния соединения с источником данных.

Конструкторы

ADO.NET-провайдер СУБД ЛИНТЕР поддерживает 3 конструктора класса `LinterDbConnection`:

LinterDbConnection()

Синтаксис

```
public LinterDbConnection();
```

Возвращаемое значение

Инициализированный объект класса `LinterDbConnection`.

LinterDbConnection(String)

Синтаксис

```
public LinterDbConnection(string strConnectionString);
```

`strConnectionString` – строка соединения с источником данных.

Возвращаемое значение

Инициализированный объект класса `LinterDbConnection`.

LinterDbConnection(LinterDbConnection)

Синтаксис

```
public LinterDbConnection(LinterDbConnection con);
```

`con` – предварительно созданный объект типа `LinterDbConnection`.

Возвращаемое значение

Инициализированный объект класса `LinterDbConnection`.

Примеры

```
LinterDbConnection con1 = new LinterDbConnection();
LinterDbConnection con2 = new LinterDbConnection(
```

```
  "User ID=SYSTEM;Password=MANAGER8;Data Source=LOCAL");
LinterDbConnection con3 = new LinterDbConnection(con2);
```

Свойства

ConnectionString

Устанавливает начальные параметры соединения с СУБД ЛИНТЕР. Значение по умолчанию – пустая строка.

Строка подключения состоит из выражений вида <параметр>=<значение>, разделённых точкой с запятой.

Параметры строки подключений

`Data Source=<имя сервера>`

Имя ЛИНТЕР-сервера, с которым должно быть установлено соединение. Параметр необязательный. Параметр надо указать, если необходимо установить соединение с удалённым ЛИНТЕР-сервером (его имя должно быть прописано в конфигурационном файле `nodetab`, используемом сетевым драйвером клиента, см. документ «Сетевые средства»).

Если параметр не указан или указан параметр `Data Source=LOCAL`, то клиентское приложение сначала пытается отослать сообщение локальному ядру СУБД по механизму межпроцессного обмена с идентификатором, заданным переменной окружения `LINTER_MBX`. Если локальное ядро не запущено или использует другой идентификатор обмена, то клиентское приложение пытается отослать сообщение сетевому драйверу клиента с идентификатором, заданным переменной окружения `NET_MBX` (см. документ «Архитектура СУБД», пункт «Механизм взаимодействия клиентских приложений с ядром СУБД»).

БД может быть создана с помощью следующих средств:

- `gendb` (см. документ «Создание и конфигурирование базы данных»);
- `linadm` (см. документ «Сетевой администратор»);

После создания БД надо создать системные представления, которые необходимы для работы ADO.NET-провайдера (см. пункт «Подготовка БД»).

`User ID=<имя пользователя>`

Имя зарегистрированного в БД пользователя. Параметр необязательный для пользователей со встроенной в ОС аутентификацией (см. документ «Справочник по SQL», конструкция `CREATE USER` с опцией `PROTOCOL`).

`Password=<пароль>`

Пароль пользователя. Параметр необязательный для пользователей, созданных с опцией `IDENTIFIED BY SYSTEM` или `PROTOCOL` (см. документ «Справочник по SQL», конструкция `CREATE USER`).

`Integrated Security=<аутентификация>`

Свойство определяет, может ли источник данных требовать предоставления ему регистрационных (учетных) данных (имя пользователя и пароль).

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Допустимые значения:

- false – источник данных должен запрашивать регистрационные данные;
- true – регистрационные данные в строке подключения можно не указывать (используется встроенная в ОС аутентификация).

Параметр необязательный (значение по умолчанию false).

Если будет указано одновременно Integrated Security=true, а также имя пользователя и пароль, то значение Integrated Security имеет преимущество (при установке соединения вместо имени пользователя и пароля будут отправлены пустые строки).

`Persist Security Info=<отображение регистрационных данных>`

Свойство определяет, разрешено ли источнику данных возвращать значения параметров строки подключения:

Допустимые значения:

- false (настоятельно рекомендуется) – важные сведения (например, пароль) не возвращаются как часть строки подключения. При сбросе строки подключения сбрасываются также все ее значения, включая пароль. Параметр необязательный (значение по умолчанию false);
- true – предоставление регистрационных данных не запрещается.

`IsolationLevel=<уровень изоляции>`

Задает уровень изоляции транзакций (см. документ «Справочник по SQL»). Параметр необязательный.

Допустимые значения:

- Optimistic;

Примечание
Режим OPTIMISTIC устарел. Применять не рекомендуется.

- Pessimistic (значение по умолчанию).

`Autocommit=<режим канала>`

Устанавливает (значение true)/отменяет (значение false) режим автоматического фиксирования (`COMMIT`) изменений в БД по данному соединению.

Значение по умолчанию true (т.е. для фиксирования изменений в БД явно выполнять SQL-оператор COMMIT не требуется).

`Minimum Pool Size=<число>`

Минимальное разрешенное количество подключений в пуле. Значение по умолчанию `0`.

Пул соединений снижает количество открытий новых соединений, т.к. пул поддерживает владение физическим соединением. Он управляет соединениями с помощью поддержания набора активных соединений для каждой конфигурации данного соединения. Каждый раз, когда пользователь вызывает метод `Open` в соединении, организатор пулов ищет в пуле доступное соединение. Если соединение

пула доступно, вместо открытия нового соединения он возвращает его инициатору открытия соединения. При вызове приложением метода `Close` в соединении вместо закрытия организатор пулов возвращает его в набор активных соединений пула. После возвращения соединения в пул оно готово к повторному использованию при следующем вызове метода `Open`.

Примечание
В текущей версии провайдера параметр не поддерживается. Зарезервирован для будущего применения.

`Maximum Pool Size=<число>`

Максимальное разрешенное количество подключений в пуле. Значение по умолчанию `100`.

Примечание
В текущей версии провайдера параметр не поддерживается. Зарезервирован для будущего применения.

`ConnectionTimeout=<число>`

Тайм-аут соединения (неотрицательное целочисленное значение). Задает временной промежуток (в сек), в течение которого провайдер будет ожидать успешное завершение соединения с источником данным. Если в заданный интервал времени соединение установлено не будет, генерируется исключение.

Параметр необязательный (значение по умолчанию 15 сек).

Примечание
В текущей версии провайдера параметр не поддерживается. Зарезервирован для будущего применения.

`Channel Priority=<число>`

Приоритет канала для данного соединения (см. документ «Интерфейс нижнего уровня»). Параметр необязательный (значение по умолчанию `0`).

`Messages Language=<язык сообщений>`

Язык информационных и диагностических сообщений, выдаваемых провайдером на консоль.

Допустимые значения:

- `en` или `US` – английский;
- `ru` или `RU` – русский.

Параметр необязательный (значение по умолчанию `en/US`).

Примечание
В текущей версии провайдера параметр не поддерживается. Зарезервирован для будущего применения.

`Charset=<кодовая страница>|DEFAULT`

Задает имя кодовой страницы, используемой для представления символьных данных в процессе обмена данными между ЛИНТЕР-сервером и клиентским приложением.

Параметр необязательный (значение по умолчанию `Encoding.Default.WebName`).

Для ЛИНТЕР-сервера заданная кодировка должна быть определена в поле NAME системной таблицы `$$$CHARSET`, пример: `Charset=CP1251` или Charset=UTF-8 (см. документы «Системные таблицы и представления» и «Справочник по SQL»).

Если имя кодовой страницы неверное, то будет использована кодовая страница DEFAULT (только первые 127 символов).

Декларация

```
public override string ConnectionString {set; get;};
```

Значение свойства

Строка соединения с источником данных.

Исключения

`ArgumentException`	Не поддерживается ключевое слово, указанное в строке подключения или недопустимое значение в строке подключения (особенно, когда логическое значение или числовое значение ожидалось, но так и не было предоставлено).

Примеры

```
// прочитать свойство (получить текст строки подключения)
string s = con.ConnectionString;
// установить свойство (устанавливается новая полная строка)
con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
// изменить свойство (чтобы изменить некоторые параметры
 необходимо менять сразу всю строку)
con.ConnectionString = "User ID=SYS;Password=MANAGER8;Data
 Source=LOCAL";
```

ConnectionTimeout

Тайм-аут соединения (свойство доступно только для чтения, устанавливается в строке подключения). Предоставляет значение временного промежутка (в сек), в течение которого провайдер будет ожидать успешное завершение соединения с источником данным. Если в заданный интервал времени соединение установлено не будет, генерируется исключение.

Примечание
В текущей версии провайдера параметр не поддерживается. Зарезервирован для будущего применения.

Декларация

```
public override int ConnectionTimeout {get;};
```

Значение свойства

Тайм-аут соединения (целочисленное положительное значение типа System.Int32).

Исключения

Отсутствуют.

Пример

```
// пример получения свойства
int t = con.ConnectionTimeout;
```

Database

Свойство, доступное только для чтения. Предоставляет имя базы данных ЛИНТЕР-сервера, с которым установлено соединение (или имя базы данных ЛИНТЕР-сервера из строки подключения, если соединение еще не установлено). Если база данных не именована, возвращается пустая строка.

При подключении клиентского приложения к другому ЛИНТЕР-серверу свойство динамически изменяется.

Чтобы параллельно подключиться к нескольким ЛИНТЕР-серверам, нужно создать несколько объектов класса `LinterDbConnection`. У каждого из этих объектов должно быть свое значение свойства Database.

Примечание
В текущей версии провайдера параметр не поддерживается. Зарезервирован для будущего применения.

Декларация

```
public override string Database {get;};
```

Значение свойства

Имя БД СУБД ЛИНТЕР (символьная строка типа System.String).

Исключения

Отсутствуют.

Пример

```
// пример получения свойства
string s = con.Database;
```

DataSource

Свойство, доступное только для чтения. Возвращает имя ЛИНТЕР-сервера, с которым установлено соединение (или имя ЛИНТЕР-сервера из строки подключения, если соединение еще не установлено). Для локального ЛИНТЕР-сервера выдается

имя LOCAL. Если имя ЛИНТЕР-сервером не установлено в строке подключения, возвращается пустая строка.

При подключении клиентского приложения к другому ЛИНТЕР-серверу свойство динамически изменяется.

Чтобы параллельно подключиться к нескольким ЛИНТЕР-серверам, нужно создать несколько объектов класса LinterDbConnection. У каждого из этих объектов должно быть свое значение свойства DataSource.

Декларация

```
public override string DataSource {get;};
```

Значение свойства

Имя ЛИНТЕР-сервера, с которым установлено соединение.

Исключения

Отсутствуют.

Пример

```
// пример получения свойства
string s = con.DataSource;
```

ServerVersion

Свойство, доступное только для чтения. Предоставляет номер версии ЛИНТЕР-сервера, с которым установлено соединение.

Декларация

```
public override string ServerVersion {get;};
```

Значение свойства

Символьная строка типа System.String в формате XX.YY.ZZZZ, где XX – номер версии, YY – номер релиза версии, ZZZZ – номер сборки, например, 06.01.0012.

Исключения

InvalidOperationException	Соединение не установлено.

Пример

```
// пример получения свойства
string s = con.ServerVersion;
```

State

Свойство, доступное только для чтения. Предоставляет информацию о текущем состоянии соединения клиентского приложения с ЛИНТЕР-сервером.

Состояние соединения устанавливается методами Open() и Close().

Декларация

```
public override System.Data.ConnectionState State {get;};
```

Значение свойства

Значение из перечисления System.Data.ConnectionState, указывающее состояние подключения.

Возможные значения ConnectionState:

`Closed`	Соединение закрыто
`Open`	Соединение установлено
`Connecting`	Соединение устанавливается (зарезервировано для будущего применения)
`Executing`	По соединению выполняется SQL-оператор (зарезервировано для будущего применения)
`Fetching`	По соединению выполнятся выборка данных из БД (зарезервировано для будущего применения)
`Broken`	Соединение разорвано (выдается только для ранее открытого соединения). Соединение можно закрыть и затем повторно открыть (зарезервировано для будущего применения).

Состояние соединения изменяется:

- от `Closed` к `Open` при выполнении метода `Open` для объекта соединения;
- от `Open` к `Closed` при выполнении метода `Close` или `Dispose` для объекта соединения.

Исключения

Отсутствуют.

Пример

```
// Создаем экземпляр класса LinterDbConnection, формируем строку
 подключения и отображаем её статус
public void CreateLinterDbConnection()
{
  LinterDbConnection connection = new LinterDbConnection();
  connection.ConnectionString =
    "User ID=SYSTEM;Password=MANAGER8;DataSource=local";
  Console.WriteLine("Connection State: " +
 connection.State.ToString());
}
```

Методы

BeginTransaction

Метод начинает транзакцию по заданному соединению. Используется уровень изоляции транзакций, заданный в строке подключения. Если в строке подключения уровень изоляции транзакций не задан, используется значение по умолчанию: `Pessimistic`.

Синтаксис

```
public DbTransaction BeginTransaction();
```

Возвращаемое значение

Объект типа `System.Data.Common.DbTransaction`, представляющий новую транзакцию.

Исключения

`InvalidOperationException`	Соединение не открыто или предыдущая транзакция не закончена.
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

```
// пример старта транзакций по нескольким соединениям
DbTransaction tran1 = con1.BeginTransaction();
DbTransaction tran2 = con2.BeginTransaction();
```

BeginTransaction(IsolationLevel)

Метод начинает транзакцию по заданному соединению с указанным уровнем изоляции транзакции.

Заданный уровень изоляции транзакций распространяется на все SQL-операторы по соединению.

Метод перекрывает уровень транзакций, заданный в строке подключения.

Нельзя изменить уровень изоляции в ходе выполнения транзакции.

Синтаксис

```
public DbTransaction BeginTransaction(IsolationLevel
 isolationLevel);
```

`isolationLevel` – уровень изоляции транзакции (перечисление System.Data.IsolationLevel)

Допустимые значения: `ReadCommitted` (`Pessimistic`), `Snapshot` (`Optimistic`).

Примечания

1. Режим Snapshot можно использовать для задания режима Optimistic только в приложениях, ориентированных на работу исключительно с СУБД ЛИНТЕР.
2. Режим Snapshot (Optimistic) устарел. Применять не рекомендуется.

Возвращаемое значение

Объект типа `System.Data.Common.DbTransaction`, представляющий новую транзакцию.

Исключения

`InvalidOperationException`	Соединение не открыто или предыдущая транзакция не закончена.

ArgumentOutOfRangeException	Уровень изоляции транзакций не поддерживается.
LinterSqlException	Код завершения СУБД ЛИНТЕР не равен 0.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class BeginTransactionSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User
 ID=SYSTEM;Password=MANAGER8;DataSource=LOCAL";
    con.Open();
    // Старт транзакции
    DbTransaction tran =
 con.BeginTransaction(IsolationLevel.ReadCommitted);
    // Команде необходимо присвоить объекты транзакции и
 подключения
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.Transaction = tran;
    try
    {
      cmd.CommandText = "insert into auto (personid) values
 (1001)";
      cmd.ExecuteNonQuery();
      cmd.ExecuteNonQuery();
      // Фиксация транзакции
      tran.Commit();
      Console.WriteLine("В базу данных записаны две строки");
    }
    catch (Exception ex)
    {
      Console.WriteLine("Ошибка фиксации транзакции: {0}",
 ex.GetType());
      Console.WriteLine("  Сообщение: {0}", ex.Message);
      // Откат транзакции
```

```
      try
      {
        tran.Rollback();
      }
      catch (Exception ex2)
      {
        // Данный блок catch должен обработать любые ошибки,
 которые могут
        // произойти при откате транзакции (например, закрытое
 соединение)
        Console.WriteLine("Ошибка отката транзакции: {0}",
 ex2.GetType());
        Console.WriteLine("  Сообщение: {0}", ex2.Message);
      }
    }
    // Освобождение ресурсов
    con.Close();
  }
}
```

ChangeDatabase

Метод меняет текущее соединение с СУБД ЛИНТЕР для последующей установки соединения с другим ЛИНТЕР-сервером.

Примечание
В текущей версии провайдера метод не поддерживается. Зарезервирован для будущего применения.

Синтаксис

```
public override void ChangeDatabase(string databaseName);
```

`databaseName` – имя ЛИНТЕР-сервера, с которым должно быть установлено новое соединение

Возвращаемое значение

Значение типа void.

Исключения

Отсутствуют.

Close

Метод закрывает соединение с текущим ЛИНТЕР-сервером (является предпочтительным способом закрытия любого открытого соединения) и выполняет фиксацию (`Commit`) всех незаконченных транзакций.

Метод можно вызывать несколько раз – исключение не генерируется.

Если объект DbConnection находится вне области видимости клиентского приложения, то при завершении работы такого приложения соединение закрыто не будет. В этом случае приложение должно явно закрыть соединение с помощью метода Close или Dispose (метод Dispose можно использовать, если в дальнейшем не предполагается работать с объектом DbConnection).

Для закрытия соединения надо использовать оператор using или блок try...finally (см. приложение 1).

Синтаксис

```
public override void Close();
```

Возвращаемое значение

Значение типа void.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ConnectionCloseSample
{
  static void Main()
  {
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User
 ID=SYSTEM;Password=MANAGER8;DataSource=LOCAL";
    try
    {
      // Соединение с БД
      con.Open();
      Console.WriteLine("Версия ЛИНТЕР: {0}", con.ServerVersion);
    }
    catch (Exception ex)
    {
      Console.WriteLine("Ошибка: {0}", ex.GetType());
      Console.WriteLine("  Сообщение: {0}", ex.Message);
    }
    finally
    {
      // Освобождение ресурсов
```

```
      if (con != null)
      {
        con.Close();
      }
    }
  }
}
```

CreateCommand

Метод создает объект `DbCommand`, связанный с текущим соединением.

Синтаксис

```
public DbCommand CreateCommand();
```

Возвращаемое значение

Объект `DbCommand` типа `System.Data.Common.DbCommand`.

Исключения

Отсутствуют.

Пример

```
// пример создания объекта и формирование на его основе SQL-
оператора
DbCommand cmd = con.CreateCommand();
cmd.CommandText = "select * from auto";
```

EnlistTransaction

Метод выполняет ручное прикрепление транзакции в текущем соединении к распределенной транзакции.

Примечание
В текущей версии ADO.NET-провайдера метод не поддерживается. При его вызове генерируется исключение `NotSupportedException`.

Синтаксис

```
public virtual void EnlistTransaction(Transaction transaction);
```

`transaction` – ссылка на существующий объект `Transaction`, в котором выполняется прикрепление.

Возвращаемое значение

Значение типа void.

Исключения

Отсутствуют.

GetSchema

Метод предоставляет список (коллекцию) всех поддерживаемых ЛИНТЕР-сервером объектов БД в текущем соединении.

Синтаксис

```
public override DataTable GetSchema();
```

Возвращаемое значение

Объект типа `System.Data.DataTable` со списком поддерживаемых объектов БД и количеством доступных клиентскому приложению атрибутов и идентификационных частей этих объектов (таблица 8).

Таблица 8. Структура записей объекта DataTable для коллекции схем MetaDataCollections

Имя столбца	Тип данных столбца	Описание
`CollectionName`	`string`	Наименование списка объектов БД (имя коллекции), которое можно передать методу `GetSchema` (String) для получения общей информации о коллекции (столбец 1 таблицы 9)
`NumberOfRestriction`	`int`	Количество атрибутов у объекта, по которым может запрашиваться индивидуальная информация. Например, для таблиц можно получать информацию о следующих 4-х атрибутах: • местоположение таблицы (на диске/в оперативной памяти); • схема таблицы; • имя таблицы; • тип таблицы. А для последовательности – только о 2-х атрибутах: схема последовательности и имя последовательности. Перечень запрашиваемых атрибутов объекта задается во втором параметре метода `GetSchema` (String, array String) (допустимые имена атрибутов в столбце 3 таблицы 9). **Примечание** В текущей версии провайдера атрибут таблицы «местоположение таблицы» не поддерживается
`NumberOfIdentifierParts`	`int`	Количество составных частей в полном идентификаторе объекта. Например, в идентификаторе таблицы это число равно `2`: имя владельца таблицы и имя таблицы, в идентификаторе столбца это число равно `3`:

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Имя столбца	Тип данных столбца	Описание
		имя владельца таблицы, имя таблицы и имя столбца

Перечень (коллекцию) всех поддерживаемых ЛИНТЕР-сервером объектов БД и их допустимые атрибуты представлен в таблице 9.

Таблица 9. Перечень возможных коллекций СУБД ЛИНТЕР

Имя коллекции	Содержимое коллекции	Допустимые атрибуты элементов коллекции
MetaDataCollections	Сведения обо всех коллекциях схем	
DataSourceInformation	Сведения об источнике данных	
DataTypes	Типы поддерживаемых данных	
Restrictions	Типы атрибутов	
ReservedWords	Зарезервированные слова	
Tables	Список таблиц БД	1) '*Catalog*' – местоположение таблицы (в текущей версии провайдера не поддерживается) 2) '*Schema*' – схема таблицы 3) '*Table*' – имя таблицы 4) '*Type*' – тип таблицы Возможные значения атрибута Type: • `SYSTEM TABLE` – системная таблица; • `TABLE` – пользовательская таблица; • `VIEW` – представление
Synonyms	Список синонимов БД	1) '*Catalog*' – местоположение синонима (в текущей версии провайдера не поддерживается) 2) '*Schema*' – схема синонима Для общих синонимов – PUBLIC 3) '*Synonym*' – имя синонима
Views	Список представлений БД	1) '*Catalog*' – местоположение представления (в текущей версии провайдера не поддерживается) 2) '*Schema*' – схема представлений 3) '*View*' – имя представления

Имя коллекции	Содержимое коллекции	Допустимые атрибуты элементов коллекции
Sequences	Список последовательностей БД	1) '*Schema*' – схема последовательности 2) '*Sequence*' – имя последовательности
Columns	Список столбцов указанного объекта БД (таблицы, представления или синонима)	1) '*Catalog*' – местоположение объекта (в текущей версии провайдера не поддерживается) 2) '*Schema*' – схема объекта 3) '*Table*' – имя объекта 4) '*Column*' – имя столбца **Примечание** При получении столбцов синонима будут возвращены столбцы таблицы, на которую указывает синоним. При этом, если синонимы указывают на другие синонимы, то выполняется переход по всей цепочке синонимов до таблицы.
Users	Список пользователей БД	1) '*User*' – имя пользователя
Roles	Список ролей БД	1) '*Role*' – имя роли
ForeignKeys	Список внешних ключей БД	1) '*Catalog*' – местоположение таблицы с этим ключом (в текущей версии провайдера не поддерживается) 2) '*Schema*' – схема таблицы 3) '*Table*' – имя таблицы 4) '*ForeignKey*' – имя внешнего ключа
ForeignKeyColumns	Список столбцов составного ключа	1) '*Catalog*' – местоположение таблицы с этим ключом (в текущей версии провайдера не поддерживается) 2) '*Schema*' – схема таблицы 3) '*Table*' – имя таблицы 4) '*ForeignKey*' – имя составного ключа 5) '*ForeignKeyColumn*' – имя столбца в составном ключе
Indexes	Список индексов БД	1) '*Catalog*' – местоположение таблицы с этим индексом (в текущей версии провайдера не поддерживается) 2) '*Schema*' – схема таблицы 3) '*Table*' – имя таблицы

Имя коллекции	Содержимое коллекции	Допустимые атрибуты элементов коллекции
		4) '*Index*' – имя индекса. В текущей версии провайдера информация о типе индекса не поддерживается
IndexColumns	Список столбцов составного индекса	1) '*Catalog*' – местоположение таблицы с этим индексом (в текущей версии провайдера не поддерживается) 2) '*Schema*' – схема таблицы 3) '*Table*' – имя таблицы 4) '*Index*' – имя индекса 5) '*Column*' – имя столбца в составном индексе
Procedures	Список хранимых процедур БД	1) '*Catalog*' – местоположение процедуры (в текущей версии провайдера не поддерживается) 2) '*Schema*' – схема процедуры 3) '*Procedure*' – имя процедуры
ProcedureParameters	Список параметров хранимой процедуры	1) '*Catalog*' – местоположение процедуры (в текущей версии провайдера не поддерживается) 2) '*Schema*' – схема процедуры 3) '*Procedure*' – имя процедуры
ProcedureColumns	Список полей курсора, возвращаемого хранимой процедурой	1) '*Catalog*' – местоположение процедуры (в текущей версии провайдера не поддерживается) 2) '*Schema*' – схема процедуры 3) '*Procedure*' – имя процедуры
Triggers	Список триггеров БД	1) '*Catalog*' – местоположение триггера (в текущей версии провайдера не поддерживается) 2) '*Schema*' – схема триггера 3) '*Trigger*' – имя триггера. В текущей версии провайдера информация о привязке триггера к таблице и типе триггера не поддерживается
Levels	Список мандатных уровней доступа	1) '*Level*' – имя мандатного уровня доступа
Stations	Список рабочих станций	1) '*Station*' – имя станции
Connections	Список активных пользователей СУБД	1) '*UserName*'– имя пользователя, открывшего канал

Имя коллекции	Содержимое коллекции	Допустимые атрибуты элементов коллекции
ReplicationServers	Список серверов репликации	1) '*ReplicationServer*' – имя удаленного сервера
Devices	Список устройств	1) '*Device*' – имя устройства
Groups	Список групп защиты	1) '*Group*' – имя группы
CharacterSets	Список кодовых таблиц БД	1) '*Id*' – системный идентификатор кодовой таблицы 2) '*Name*' – имя кодовой таблицы

Примечание

Поддержка ниже перечисленных коллекций будет реализована в следующей версии ADO.NET-провайдера:

- список хранимых событий;
- список алиасов;
- список форматов внешних файлов;
- список правил репликации;
- список фильтров полнотекстового поиска;
- список правил трансляции кодировок.

Исключения

`InvalidOperationException`	Соединение не открыто.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetSchemaSample
{
  static void Main()
  {
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User
 ID=SYSTEM;Password=MANAGER8;DataSource=LOCAL";
    try
    {
      // Соединение с БД
      con.Open();
```

```
      // Получение сведений о схеме базы данных
      DataTable schema = con.GetSchema();
      // Вывод полученных сведений на экран
      OutputDataTable(schema);
    }
    catch (Exception ex)
    {
      Console.WriteLine("Ошибка: " + ex.Message);
    }
    finally
    {
      // Освобождение ресурсов
      con.Close();
    }
  }
  private static void OutputDataTable(DataTable dataTable)
  {
    Console.WriteLine(new String('-', 60));
    foreach (DataColumn column in dataTable.Columns)
    {
      Console.Write(column.ColumnName + " | ");
    }
    Console.WriteLine();
    Console.WriteLine(new String('-', 60));
    foreach (DataRow row in dataTable.Rows)
    {
      for (int i = 0; i < dataTable.Columns.Count; i++)
      {
        if (row.IsNull(i))
        {
          Console.Write("<NULL> | ");
        }
        else
        {
          Console.Write(row[i] + " | ");
        }
      }
      Console.WriteLine();
    }
  }
}
```

Результат выполнения примера:

```
------------------------------------------------------------------
CollectionName       |NumberOfRestriction|NumberOfIdentifierParts|
```

```
-------------------------------------------------------------------
CharacterSets         |2                    |1                    |
Columns               |4                    |3                    |
Connections           |1                    |1                    |
DataSourceInformation|0                    |0                    |
DataTypes             |0                    |0                    |
Devices               |1                    |1                    |
ForeignKeyColumns     |5                    |4                    |
ForeignKeys           |4                    |3                    |
Groups                |1                    |1                    |
IndexColumns          |5                    |4                    |
Indexes               |4                    |3                    |
Levels                |1                    |1                    |
MetaDataCollections   |0                    |0                    |
ProcedureColumns      |3                    |3                    |
ProcedureParameters   |3                    |3                    |
Procedures            |3                    |2                    |
ReplicationServers    |1                    |1                    |
ReservedWords         |0                    |0                    |
Restrictions          |0                    |0                    |
Roles                 |1                    |1                    |
Sequences             |2                    |2                    |
Stations              |1                    |1                    |
Synonyms              |3                    |2                    |
Tables                |4                    |2                    |
Triggers              |3                    |2                    |
Users                 |1                    |1                    |
Views                 |3                    |2                    |
```

GetSchema(String)

Метод предоставляет метаданные указанных объектов БД (коллекции данных), связанных с текущим соединением.

Синтаксис

```
public override DataTable GetSchema(string collectionName);
```

`collectionName` – имя коллекции данных (строковое значение из столбца 1 в таблице 9).

Возвращаемое значение

Объект `DataTable` типа `System.Data.DataTable` (структуру записей см. в таблице 8).

Исключения

`ArgumentException`	Коллекция данных не поддерживается.
`InvalidOperationException`	Соединение не открыто.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

LinterSqlException	Код завершения СУБД ЛИНТЕР не равен 0.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetSchemaSample
{
  static void Main()
  {
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User
 ID=SYSTEM;Password=MANAGER8;DataSource=LOCAL";
    try
    {
      // Соединение с БД
      con.Open();
      // Получение сведений о схеме базы данных
      DataTable schema = con.GetSchema("Restrictions");
      // Вывод полученных сведений на экран
      OutputDataTable(schema);
    }
    catch (Exception ex)
    {
      Console.WriteLine("Ошибка: " + ex.Message);
    }
    finally
    {
      // Освобождение ресурсов
      con.Close();
    }
  }
  private static void OutputDataTable(DataTable dataTable)
  {
    Console.WriteLine(new String('-', 60));
    foreach (DataColumn column in dataTable.Columns)
    {
      Console.Write(column.ColumnName + " | ");
    }
    Console.WriteLine();
    Console.WriteLine(new String('-', 60));
    foreach (DataRow row in dataTable.Rows)
```

```
    {
      for (int i = 0; i < dataTable.Columns.Count; i++)
      {
        if (row.IsNull(i))
        {
          Console.Write("<NULL> | ");
        }
        else
        {
          Console.Write(row[i] + " | ");
        }
      }
      Console.WriteLine();
    }
  }
}
```

Результат выполнения примера:

```
-------------------------------------------------------------------------------
CollectionName     |RestrictionName  |RestrictionDefault|
RestrictionNumber|
-------------------------------------------------------------------------------
CharacterSets      |Id               |                  |0
        |
CharacterSets      |Name             |                  |1
        |
Columns            |Catalog          |                  |0
        |
Columns            |Schema           |                  |1
        |
Columns            |Table            |                  |2
        |
Columns            |Column           |                  |3
        |
Connections        |UserName         |                  |0
        |
Devices            |Device           |                  |0
        |
ForeignKeyColumns  |Catalog          |                  |0
        |
ForeignKeyColumns  |Schema           |                  |1
        |
ForeignKeyColumns  |Table            |                  |2
        |
```

```
ForeignKeyColumns  |ForeignKey       |                  |3
        |
ForeignKeyColumns  |ForeignKeyColumn |                  |4
        |
ForeignKeys        |Catalog          |                  |0
        |
ForeignKeys        |Schema           |                  |1
        |
ForeignKeys        |Table            |                  |2
        |
ForeignKeys        |ForeignKey       |                  |3
        |
Groups             |Group            |                  |0
        |
IndexColumns       |Catalog          |                  |0
        |
IndexColumns       |Schema           |                  |1
        |
IndexColumns       |Table            |                  |2
        |
IndexColumns       |Index            |                  |3
        |
IndexColumns       |Column           |                  |4
        |
Indexes            |Catalog          |                  |0
        |
Indexes            |Schema           |                  |1
        |
Indexes            |Table            |                  |2
        |
Indexes            |Index            |                  |3
        |
Levels             |Level            |                  |0
        |
ProcedureColumns   |Catalog          |                  |0
        |
ProcedureColumns   |Schema           |                  |1
        |
ProcedureColumns   |Procedure        |                  |2
        |
ProcedureParameters|Catalog          |                  |0
        |
ProcedureParameters|Schema           |                  |1
        |
ProcedureParameters|Procedure        |                  |2
        |
```

```
Procedures          |Catalog           |                    |0
        |
Procedures          |Schema            |                    |1
        |
Procedures          |Procedure         |                    |2
        |
ReplicationServers |ReplicationServer|                    |0
        |
Roles               |Role              |                    |0
        |
Sequences           |Schema            |                    |0
        |
Sequences           |Sequence          |                    |1
        |
Stations            |Station           |                    |0
        |
Synonyms            |Catalog           |                    |0
        |
Synonyms            |Schema            |                    |1
        |
Synonyms            |Synonym           |                    |2
        |
Tables              |Catalog           |                    |0
        |
Tables              |Schema            |                    |1
        |
Tables              |Table             |                    |2
        |
Tables              |Type              |                    |3
        |
Triggers            |Catalog           |                    |0
        |
Triggers            |Schema            |                    |1
        |
Triggers            |Trigger           |                    |2
        |
Users               |User              |                    |0
        |
Views               |Catalog           |                    |0
        |
Views               |Schema            |                    |1
        |
Views               |View              |                    |2
        |
```

GetSchema(String, String[ ])

Метод предоставляет метаданные в соответствии с запрошенными атрибутами указанного объекта БД (коллекции данных), связанного с текущим соединением.

Синтаксис

```
public override DataTable GetSchema
(
    String collectionName,
    String[] restrictionValues
);
```

`collectionName` – имя коллекции данных (строковое значение из столбца 1 в таблице 1). Если указано значение NULL, метаданные предоставляются обо всех коллекциях БД и о количестве атрибутов элементов коллекции.

`restrictionValues` – массив строковых значений, содержащий:

- перечень требуемых атрибутов метаданных (одно или несколько значений из столбца 3 в таблице 1). В этом случае выдается информация об этом атрибуте всех объектов указанной коллекции;
- конкретное значение атрибута (атрибутов) – имя пользователя, название таблицы, имя столбца и т.п. В этом случае выдается информация только об объектах с заданными атрибутами из указанной коллекции.

Запрашиваемые атрибуты элементов коллекции должны задаваться строго в соответствии с их порядковыми номерами (указанными в столбце 3 таблицы 1), поэтому если информация о каком-то атрибуте не требуется, на его позиции необходимо указать null.

Пример. Узнать всех владельцев таблиц AUTO. Согласно таблице 3 (столбец 3) в коллекции TABLES имени владельца соответствует 3 уровень детализации, поэтому массив restrictionValues должен выглядеть так (первые два атрибута пропущены):

```
string[] { null, null, "AUTO", null };
```

или так

```
string[] { null, null, "AUTO" };
```

(последние элементы массива можно опускать)

Если явное значение атрибута не указано (в смысле null), то из БД будут извлекаться элементы коллекции со всеми значениями атрибута.

Возвращаемое значение

Объект `DataTable` типа `System.Data.DataTable`.

Структура возвращаемых записей зависит от коллекции.

Исключения

`ArgumentException`	Коллекция данных не поддерживается.

InvalidOperationException	Соединение не открыто.
LinterSqlException	Код завершения СУБД ЛИНТЕР не равен 0.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetSchemaSample
{
  static void Main()
  {
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User
 ID=SYSTEM;Password=MANAGER8;DataSource=LOCAL";
    try
    {
      // Соединение с БД
      con.Open();
      // Получение сведений о схеме базы данных
      string[] restrictions = new string[] { null, "SYSTEM",
 "AUTO", null };
      DataTable schema = con.GetSchema("Columns", restrictions);
      // Вывод полученных сведений на экран
      OutputDataTable(schema);
    }
    catch (Exception ex)
    {
      Console.WriteLine("Ошибка: " + ex.Message);
    }
    finally
    {
      // Освобождение ресурсов
      con.Close();
    }
  }
  private static void OutputDataTable(DataTable dataTable)
  {
    int columnsNumber = Math.Min(5, dataTable.Columns.Count);
    Console.WriteLine(new String('-', 60));
    for (int i = 0; i < columnsNumber; i++)
    {
      Console.Write(dataTable.Columns[i].ColumnName + " | ");
```


© Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    }
    Console.WriteLine();
    Console.WriteLine(new String('-', 60));
    foreach (DataRow row in dataTable.Rows)
    {
      for (int i = 0; i < columnsNumber; i++)
      {
        if (row.IsNull(i))
        {
          Console.Write("<NULL> | ");
        }
        else
        {
          Console.Write(row[i] + " | ");
        }
      }
      Console.WriteLine();
    }
  }
}
```

Результат выполнения примера:

```
------------------------------------------------------------
TABLE_CATALOG|TABLE_SCHEMA|TABLE_NAME|COLUMN_NAME|DATA_TYPE|
------------------------------------------------------------
             |SYSTEM      |AUTO      |MAKE       |CHAR     |
             |SYSTEM      |AUTO      |MODEL      |CHAR     |
             |SYSTEM      |AUTO      |BODYTYPE   |CHAR     |
             |SYSTEM      |AUTO      |CYLNDERS   |INT      |
             |SYSTEM      |AUTO      |HORSEPWR   |INT      |
             |SYSTEM      |AUTO      |DSPLCMNT   |INT      |
             |SYSTEM      |AUTO      |WEIGHT     |INT      |
             |SYSTEM      |AUTO      |COLOR      |CHAR     |
             |SYSTEM      |AUTO      |YEAR       |INT      |
             |SYSTEM      |AUTO      |SERIALNO   |CHAR     |
             |SYSTEM      |AUTO      |CHKDATE    |INT      |
             |SYSTEM      |AUTO      |CHKMILE    |INT      |
             |SYSTEM      |AUTO      |PERSONID   |INT      |
```

Open

Метод открывает соединение с ЛИНТЕР-сервером в соответствии с параметрами, указанными в строке подключения.

Синтаксис

```
public override void Open();
```

Возвращаемое значение

Значение типа void.

Исключения

`InvalidOperationException`	Недопустимая строка подключения, соединение уже открыто или версия ADO.NET-провайдера не совместима с версией СУБД ЛИНТЕР.
`ArgumentOutOfRangeException`	Уровень изоляции транзакций, заданный в строке подключения, не поддерживается.
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ConnectionOpenSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8";
    try
    {
      con.Open();
      Console.WriteLine("Соединение открыто");
    }
    catch (Exception ex)
    {
      Console.WriteLine("Ошибка при открытии соединения: " +
 ex.Message);
    }
    finally
    {
      con.Close();
      Console.WriteLine("Соединение закрыто");
    }
  }
}
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

События

StateChange

Событие генерируется при изменении состояния соединения с ЛИНТЕР-сервером. (т.е. когда состояние соединения изменяется с закрытого на открытое или наоборот).

Синтаксис

```
public override event StateChangeEventHandler StateChange
```

Пример

Пример обработки событий см. в разделе Обработка событий.

Класс DbCommand

Экземпляр класса `DbCommand` представляет собой SQL-запрос или вызов хранимой процедуры, снабженный методами для выполнения этого запроса. Объекты `DbCommand` могут выполнять группу запросов, разделенных знаком «;» в пределах одной команды. В этом случае коллекция параметров должна объединять все параметры из всех запросов.

Примечание
В текущей версии ADO.NET провайдера не поддерживается выполнение группы запросов, среди которых есть запрос создания хранимой процедуры, триггера или `EXECUTE BLOCK`. В этом случае команда должна содержать только один запрос, который выполняет любое из вышеуказанных действий.

Класс `DbCommand` дает возможность выполнять запрос разными способами:

- если запрос не возвращает записи, необходимо вызвать метод `ExecuteNonQuery`;
- если запрос возвращает записи, необходимо вызвать метод `ExecuteReader`, который возвращает объект `DataReader`, позволяющий просматривать записи;
- если необходимо получить только первое поле первой записи, необходимо вызвать метод `ExecuteScalar`.

Класс `DbCommand` дает возможность выполнять запросы в асинхронном режиме (см. приложение 2).

Для освобождения объекта DbCommand надо использовать оператор using или блок try...finally (см. приложение 1).

Конструкторы класса приведены в таблице 10.

Таблица 10. Конструкторы класса DbCommand

Конструктор	Описание
LinterDbCommand()	Создает новый объект `DbCommand` с параметрами по умолчанию.
LinterDbCommand(String)	Создает новый объект `DbCommand`

Конструктор	Описание
	с указанным SQL-запросом.
LinterDbCommand(String, LinterDbConnection)	Создает новый объект `DbCommand` с указанным SQL-запросом по заданному соединению с источником данных.
LinterDbCommand(String,LinterDbConnection,LinterDbTransaction)	Создает новый объект `DbCommand` с указанным SQL-запросом по заданному соединению с источником данных и режимом обработки транзакций.

Свойства класса приведены в таблице 11.

Таблица 11. Свойства класса DbCommand

Свойство	Описание
CommandText	Предоставляет/устанавливает текущее значение текста SQL-запроса.
CommandTimeout	Предоставляет/устанавливает максимально допустимый интервал ожидания завершения выполнения SQL-запроса сервером источника данных.
CommandType	Индикатор типа SQL-запроса (доступ к таблице или к хранимой процедуре).
Connection	Предоставляет/устанавливает объект `DbConnection` (соединение), который должен использоваться (используется) для соединения с источником данных.
DesignTimeVisible	Индикатор видимости объекта `DbCommand` в настраиваемом элементе управления интерфейса разработчика программного обеспечения (Windows Forms Designer).
Parameters	Предоставляет описание параметров параметризованного SQL-запроса или хранимой процедуры.
Transaction	Предоставляет/устанавливает значение объекта `DbTransaction`, используемого в соединении с источником данных.
UpdatedRowSource	Индикатор механизма изменения данных в объекте `DataRow` после выполнения метода `Update DbDataAdapter`.

Методы класса приведены в таблице 12.

Таблица 12. Методы класса DbCommand

Метод	Описание
Cancel	Отмена выполнения команды по текущему соединению.

Метод	Описание
CreateParameter	Создает новый экземпляр объекта `DbParameter`.
ExecuteNonQuery	Выполняет SQL-запрос, не возвращающий данные.
ExecuteReader	Выполняет свойство CommandText по соединению Connection и помещает результаты (обычно выборку данных) в объект `DbDataReader`.
ExecuteReader(CommandBehavior)	Выполняет свойство CommandText по соединению Connection и помещает результаты (обычно выборку данных) согласно заданным в аргументе CommandBehavior условиям в объект `DbDataReader`.
ExecuteScalar	Выполняет SQL-запрос и возвращает первый столбец первой строки результирующей выборки данных.
Prepare	Подготавливает параметризованный SQL-запрос или хранимую процедур с параметрами для последующего выполнения.

Конструкторы

ADO.NET-провайдер СУБД ЛИНТЕР обеспечивает поддержку четырех конструкторов класса LinterDbCommand:

LinterDbCommand()

Синтаксис

```
public LinterDbCommand();
```

Возвращаемое значение

Конструктор создает новый объект `DbCommand` со следующими параметрами:

- текст SQL-запроса – пустая строка;
- тайм-аут ожидания завершения команды (свойство CommandTimeout) – 30 сек;
- пустая коллекция параметров;
- режим синхронизации БД – синхронизация в оба направления (UpdateRowSource.Both).

После создания объекта `DbCommand` необходимо определить соединение, по которому будет выполняться команда (свойство Connection). Если запрос должен выполняться в транзакции, то необходимо установить свойство Transaction.

LinterDbCommand(String)

Синтаксис

```
public LinterDbCommand(String strCommandText);
```

`strCommandText` – текст SQL-запроса.

Возвращаемое значение

Конструктор создает новый объект `DbCommand` со следующими параметрами:

- текст SQL-запроса – задается в аргументе strCommandText;
- тайм-аут ожидания завершения команды (свойство CommandTimeout) – 30 сек;
- пустая коллекция параметров;
- режим синхронизации БД – синхронизация в оба направления (UpdateRowSource.Both).

После создания объекта `DbCommand` необходимо определить соединение, по которому будет выполняться команда (свойство Connection). Если запрос должен выполняться в транзакции, то необходимо установить свойство Transaction.

LinterDbCommand(String, LinterDbConnection)

Синтаксис

```
public LinterDbCommand (String strCommandText, LinterDbConnection
 connection);
```

`strCommandText` – текст SQL-запроса.

`connection` – объект `LinterDbConnection`, используемый для соединения с СУБД.

Возвращаемое значение

Конструктор создает новый объект `DbCommand` для заданного соединения со следующими параметрами:

- текст SQL-запроса – задается в аргументе strCommandText;
- соединение, по которому должен выполняться SQL-запрос, задается в аргументе connection;
- тайм-аут ожидания завершения команды (свойство CommandTimeout) – 30 сек;
- пустая коллекция параметров;
- режим синхронизации БД – синхронизация в оба направления (UpdateRowSource.Both).

Если запрос должен выполняться в транзакции, то необходимо установить свойство Transaction.

LinterDbCommand(String, LinterDbConnection, LinterDbTransaction)

Синтаксис

```
public LinterDbCommand (String strCommandText, LinterDbConnection
 connection, LinterDbTransaction transaction);
```

`strCommandText` – текст SQL-запроса.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

`connection` – объект `LinterDbConnection`, используемый для соединения с СУБД.

`transaction` – режим обработки транзакций.

Возвращаемое значение

Конструктор создает новый объект `DbCommand` для заданного соединения со следующими параметрами:

- текст SQL-запроса – задается в аргументе strCommandText;
- соединение, по которому должен выполняться SQL-запрос, задается в аргументе connection;
- режим обработки транзакций задается в аргументе transaction;
- тайм-аут ожидания завершения команды (свойство CommandTimeout) – 30 сек;
- пустая коллекция параметров;
- режим синхронизации БД – синхронизация в оба направления (UpdateRowSource.Both).

Свойства

CommandText

Свойство предоставляет или устанавливает текущее значение текста SQL-запроса.

Если в свойстве CommandType указано, что должна выполняться хранимая процедура, то в свойстве CommandText надо задать только имя этой процедуры (в этом случае выполнение процедуры инициируется методом Execute), иначе в свойстве CommandText надо задать ключевое слово CALL или EXECUTE <имя процедуры>(<параметры>).

Декларация

```
public abstract string CommandText {get; set;};
```

`CommandText` – текст SQL-запроса, заканчивающийся символом «;» (или пакета SQL-запросов, разделенных символом «;»).

Значение свойства

Строка типа System.String с текстом SQL-запроса. Если свойство не установлено, возвращается пустая строка ("").

Исключения

Отсутствуют.

Примеры

1) Выполнение запроса вставки записи в таблицу.

```
// C#
using System;
```

```
using System.Data;
using System.Data.Common;

class CommandTextSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText =
      "insert into auto (make,model,personid) values
 ('Ford','Focus',1111)";
    // Выполнение SQL-запроса
    int rowsAffected = cmd.ExecuteNonQuery();
    Console.WriteLine("Количество обработанных строк: {0}",
 rowsAffected);
    // Освобождение ресурсов
    con.Close();
  }
}
```

2) Выполнение запроса создания таблицы.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandTextSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "create table test (test_name varchar(70))";
    // Выполнение SQL-запроса
    cmd.ExecuteNonQuery();
    // Освобождение ресурсов
    con.Close();
  }
}
```

3) Выполнение хранимой процедуры.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandTextSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание входного параметра хранимой процедуры
    DbParameter par = factory.CreateParameter();
    par.ParameterName = "name";
    par.Direction = ParameterDirection.Input;
    par.DbType = DbType.String;
    par.Size = 66;
    par.Value = "test_cursor";
    // Создание команды для выполнения хранимой процедуры
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.CommandText = "p_ret_cursor"; // имя хранимой процедуры
    cmd.Parameters.Add(par);
```

```
    cmd.Prepare();
    // Выполнение хранимой процедуры, возвращающей курсор
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка полученных данных
    while (reader.Read())
    {
      Console.WriteLine(String.Format("{0}, {1}", reader[0],
 reader[1]));
    }
    // Освобождение ресурсов
    reader.Close();
    con.Close();
  }
}
```

Примечание
Если в хранимой процедуре есть параметр типа BLOB, то вызов такой хранимой процедуры надо сделать из другой хранимой процедуры.

```
create or replace table tab1 (a int, b blob);

insert into tab1 (a, b) values (1, hex('010203'));
create or replace procedure proc1(in b blob)
  result int
code
  return blob_size(b);
end;
create or replace procedure proc2(in a int)
  result int
declare
  var b blob;
code
  execute direct "select b from tab1 where a = " + itoa(a) into b;
  return proc1(b);
end;
// C#
using System;
using System.Data;
using System.Data.Common;
class CommandTextSample
{
  static void Main()
  {
    DbConnection con = null;
    try
    {
```

```
      DbProviderFactory factory =

 DbProviderFactories.GetFactory("System.Data.LinterClient");
      con = factory.CreateConnection();
      con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8";
      con.Open();
      DbCommand cmd = factory.CreateCommand();
      cmd.Connection = con;
      cmd.CommandText = "call proc2(1);";
      object a = cmd.ExecuteScalar();
      Console.WriteLine(a);
    }
    catch (Exception ex)
    {
      Console.WriteLine("Ошибка: " + ex.Message);
    }
    finally
    {
      if (con != null)
      {
        con.Close();
      }
    }
  }
}
```

4) Выполнение пакета SQL-запросов.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CreateCommandSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
```

```
    cmd.Connection = con;
    // Формирование пакета SQL-запросов
    cmd.CommandText = "select make from auto;select name from
 person";
    // Выполнение пакета SQL-запросов
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов пакета запросов
    while (reader.HasRows)
    {
      while (reader.Read())
      {
        Console.WriteLine(reader[0]);
      }
      // Перемещение к следующему запросу в пакете
      reader.NextResult();
    }
    // Освобождение ресурсов
    reader.Close();
    con.Close();
  }
}
```

CommandTimeout

Свойство предоставляет или устанавливает максимально допустимый интервал ожидания (в сек) завершения выполнения SQL-запроса ЛИНТЕР-сервером.

Если в течение заданного промежутка времени выполнение SQL-запроса не завершилось, то запрос игнорируется и генерируется исключение LinterSqlException.

Если свойство не установлено, по умолчанию используется значение 30 сек.

Примечание
Зарезервировано для будущего применения.

Декларация

```
public abstract int CommandTimeout {get; set;};
```

`CommandTimeout` – положительное значение или `0`.

Нулевое значение означает бесконечное время ожидания, и в CommandTimeout его следует избегать.

Значение свойства

Значение типа System.Int32 с текущим интервалом ожидания.

Исключения

Отсутствуют.

CommandType

Свойство указывает ADO.NET-провайдеру, как ему следует интерпретировать свойство CommandText.

Если для свойства CommandType задано значение StoredProcedure, то для свойства CommandText следует указать имя хранимой процедуры, к которой должен быть получен доступ.

Если для свойства CommandType задано значение TableDirect, то для свойства CommandText следует указать имя таблицы, к которой должен быть получен доступ.

Чтобы получить доступ к нескольким таблицам, необходимо указать разделенный запятыми список имен этих таблиц, без пробелов и других разделителей. Если свойство CommandText содержит несколько таблиц, возвращается объединение указанных таблиц.

Декларация

```
public abstract CommandType CommandType {get; set;};
```

Значение свойства

Характеристика свойства CommandType (значение типа System.Data.CommandType):

Имя члена CommandType	*Описание*
Text	Текстовая команда SQL (по умолчанию).
StoredProcedure	Имя хранимой процедуры.
TableDirect	Имя таблицы.

Исключения

Отсутствуют.

Примеры

1) Получить свойства.

```
CommandType cmdType = cmd.CommandType;
```

2) Установить нужные свойства.

```
cmd.CommandType = CommandType.StoredProcedure;
```

Connection

Свойство устанавливает объект (или предоставляет информацию об объекте) `DbConnection` (соединение), который должен использоваться при выполнении объекта `DbCommand`.

Декларация

```
public DbConnection Connection {get; set;};
```

Значение свойства

Объект типа `System.Data.Common.DbConnection`, используемый для соединения с СУБД ЛИНТЕР при выполнении `DbCommand`.

Если объект System.Data.Common.DbConnection не создан, возвращается null.

Исключения

Отсутствуют.

Примеры

1) Проверить, что соединение с ЛИНТЕР-сервером «Marketing» создано и установлено.

```
if (cmd.Connection != null &&
  cmd.Connection.DataSource == " Marketing" &&
  cmd.Connection.State == ConnectionState.Open)
{
  Console.WriteLine("Соединение создано и установлено");
}
```

2) Получить информацию о текущем соединении.

```
Console.WriteLine("Имя ЛИНТЕР-сервера: {0}",
 cmd.Connection.DataSource);
Console.WriteLine("Состояние: {0}", cmd.Connection.State);
```

3) Установить новое соединение с ЛИНТЕР-сервером.

```
    cmd.Connection = con2;
```

DesignTimeVisible

Свойство устанавливает или предоставляет значение, определяющее, должен ли объект DbCommand быть видимым в настраиваемом элементе управления интерфейса разработчика программного обеспечения (Windows Forms Designer).

Если значение явно не установлено, по умолчанию принимается значение true (т.е. объект DbCommand должен быть видимым).

Декларация

```
[BrowsableAttribute(false)] public abstract bool DesignTimeVisible
 {get; set;};
```

Значение свойства

Значение типа System.Boolean: true – объект DbCommand является видимым, иначе – false.

Исключения

Отсутствуют.

Примеры

1) Проверить видимость объекта DbCommand.

```
  if (cmd.DesignTimeVisible)
{
  Console.WriteLine("Объект видимый");
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
}
else
{
  Console.WriteLine("Объект невидимый");
}
```

2) Изменить текущее состояние видимости объекта DbCommand.

```
cmd.DesignTimeVisible = false;
```

Примечание
Свойство используется, как правило, для управления видимостью (доступностью) объекта в пользовательском графическом интерфейсе.

Parameters

Свойство предоставляет описание параметров параметризованного SQL-запроса или хранимой процедуры.

Декларация

```
[BrowsableAttribute(false)] public DbParameterCollection
 Parameters {get;};
```

Значение свойства

Значение типа System.Data.Common.DbParameterCollection.

Примечание
Работа с коллекцией параметров описана в разделе Класс DbParameterCollection.

Исключения

Отсутствуют.

Пример

Получить объект DbParameterCollection.

```
DbParameterCollection pars = cmd.Parameters;
```

Transaction

Свойство предоставляет или устанавливает значение DbTransaction (раздел Класс DbTransaction) для соединения, по которому должен выполняться объект `DbCommand`.

Декларация

```
[BrowsableAttribute(false)] public DbTransaction Transaction {get;
 set;};
```

Значение свойства

Транзакция (значение типа System.Data.Common.DbTransaction), внутри которой выполняется объект `DbCommand`. Если активной транзакции нет, возвращается null-значение.

Исключения

Отсутствуют.

Пример

Проверить, есть транзакция? Если да, закончить её и начать новую.

```
if (cmd.Transaction != null)
{
  cmd.Transaction.Commit();
  cmd.Transaction = cmd.Connection.BeginTransaction();
}
```

Примечание
Режим транзакции изменить нельзя.

UpdatedRowSource

Свойство определяет, каким образом ADO.NET-провайдер должен вносить изменения в объект `DataRow` (т.е. выполнять синхронизацию с БД) после выполнения метода `Update` `DbDataAdapter`.

Декларация

```
public abstract UpdateRowSource UpdatedRowSource {get; set;}
```

Значение свойства

Значение типа System.Data.UpdateRowSource:

Имя члена	Описание
None	Все изменения игнорируются (значение по умолчанию в случае, когда объект `DbCommand` создается автоматически (с помощью объекта `DbCommandBuilder`)).
OutputParameters	Измененные выходные параметры отображаются в измененной строке объекта `DataSet`.
FirstReturnedRecord	Данные первой измененной строки отображаются в измененной строке объекта `DataSet`.
Both	Измененные выходные параметры и первая измененная строка отображаются в измененной строке объекта `DataSet` (значение по умолчанию).

Примечание
Подробное описание значений UpdateRowSource приведено в руководстве Microsoft «Обновление источников данных с помощью объектов DataAdapter».

Исключения

Отсутствуют.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
class UpdatedRowSourceSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User
 ID=SYSTEM;Password=MANAGER8;DataSource=LOCAL";
    try
    {
      con.Open();
      // Создание таблицы БД
      DbCommand cmd = factory.CreateCommand();
      cmd.Connection = con;
      cmd.CommandText = "create if not exists table policy " +
        " (policy_id integer autoinc, policy_name varchar(70))";
      cmd.ExecuteNonQuery();
      // Создание объекта DataTable
      DataTable policy = new DataTable();
      policy.Columns.Add("policy_id", typeof(int));
      policy.Columns.Add("policy_name", typeof(string));
      // Создание объекта DbParameter
      DbParameter par = factory.CreateParameter();
      par.ParameterName = ":policy_name";
      par.SourceColumn = "policy_name";
      par.Direction = ParameterDirection.Input;
      par.DbType = DbType.String;
      par.Size = 70;
      // Создание объекта DbDataAdapter
      DbDataAdapter adapter = factory.CreateDataAdapter();
      adapter.SelectCommand = factory.CreateCommand();
      adapter.SelectCommand.Connection = con;
      adapter.SelectCommand.CommandText = "select * from policy";
      adapter.InsertCommand = factory.CreateCommand();
      adapter.InsertCommand.Connection = con;
      adapter.InsertCommand.CommandText = "insert into policy
 (policy_name) " +
```

```
        " values (:policy_name); select last_autoinc as
policy_id";
      adapter.InsertCommand.Parameters.Add(par);
      adapter.InsertCommand.UpdatedRowSource =
UpdateRowSource.FirstReturnedRecord;
      // Заполнение объекта DataTable данными из таблицы БД
      adapter.Fill(policy);
      // Добавление записи в таблицу DataTable
      DataRow row = policy.NewRow();
      row["policy_name"] = "Политика " + DateTime.Now.ToString();
      policy.Rows.Add(row);
      // Синхронизация объекта DataTable с таблицей БД
      adapter.Update(policy);
      // Отображение полученного объекта DataTable
      OutputDataTable(policy);
    }
    catch (Exception ex)
    {
      Console.WriteLine("Ошибка: " + ex.Message);
    }
    finally
    {
      // Освобождение ресурсов
      con.Close();
    }
  }
  private static void OutputDataTable(DataTable dataTable)
  {
    Console.WriteLine(new String('-', 60));
    foreach (DataColumn column in dataTable.Columns)
    {
      Console.Write(column.ColumnName + " | ");
    }
    Console.WriteLine();
    Console.WriteLine(new String('-', 60));
    foreach (DataRow row in dataTable.Rows)
    {
      for (int i = 0; i < dataTable.Columns.Count; i++)
      {
        if (row.IsNull(i))
        {
          Console.Write("<NULL> | ");
        }
        else
        {
          Console.Write(row[i] + " | ");
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
        }
      }
      Console.WriteLine();
    }
  }
}
```

Результат выполнения примера:

```
----------------------------------------
policy_id | policy_name                |
----------------------------------------
1         | Политика 11.07.2012 16:28:57 |
```

Методы

Cancel

Метод пытается отменить (по возможности) выполнение команды по текущему соединению.

Если в данный момент по соединению нет выполняющейся команды, то метод не выполняет никаких действий.

Вызов метода не гарантирует, что команда обязательно будет отменена. Её выполнение может завершиться еще до того, как начнется отмена команды. В таком случае исключение не генерируется.

Синтаксис

```
public abstract void Cancel();
```

Возвращаемое значение

Значение типа void.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data.LinterClient;
using System.Threading;

class CommandCancelSample
{
  static LinterDbConnection m_connection = null;
  static LinterDbCommand m_command = null;

  static void Main()
```

```
  {
    Thread waitThread = new Thread(new ThreadStart(Wait));
    Thread abortThread = new Thread(new ThreadStart(Abort));
    waitThread.Start();
    abortThread.IsBackground = true;
    abortThread.Start();
  }

  public static void Wait()
  {
    try
    {
      m_connection = new LinterDbConnection();
      m_connection.ConnectionString =
"UserID=SYSTEM;Password=MANAGER8";
      m_connection.Open();

      m_command = new LinterDbCommand();
      m_command.Connection = m_connection;
      m_command.CommandText = "create or replace event E";
      m_command.ExecuteNonQuery();

      Console.WriteLine("Ожидание события E...");
      m_command.CommandText = "wait event E";
      m_command.ExecuteNonQuery();
      Console.WriteLine("Событие произошло.");
    }
    catch (LinterSqlException ex)
    {
      Console.WriteLine(
        "Исключение ядра СУБД ЛИНТЕР \n" +
        "Текст сообщения: " + ex.Message + "\n" +
        "Код СУБД ЛИНТЕР: " + ex.Number + "\n" +
        "Код операционной системы: " + ex.LinterSysErrorCode +
"\n");
    }
    catch (Exception ex)
    {
      Console.WriteLine(
        "Исключение ADO.NET провайдера \n" +
        "Тип ошибки: " + ex.GetType() + "\n" +
        "Сообщение: " + ex.Message + "\n");
    }
    finally
    {
      Console.WriteLine("Освобождение ресурсов.");
```

```
      if (m_connection != null)
      {
        m_connection.Close();
      }
      Console.WriteLine("Выполнение команды завершено.");
    }
  }

  public static void Abort()
  {
    Console.WriteLine("Чтобы отменить выполнение команды нажмите
 Esc.");
    do
    {
      if (Console.KeyAvailable)
      {
        if (Console.ReadKey(true).Key == ConsoleKey.Escape)
        {
          if (m_command != null)
          {
            m_command.Cancel();
            Console.WriteLine("Выполнение команды отменено.");
            return;
          }
        }
      }
    } while (true);
  }
}
```

```
Результат выполнения примера:

Чтобы отменить выполнение команды нажмите Esc.
Ожидание события E...
Выполнение команды отменено.
Исключение ядра СУБД ЛИНТЕР
Текст сообщения: [Linter error] query has been cancelled
Код СУБД ЛИНТЕР: 1097
Код операционной системы: 0
Освобождение ресурсов.
Выполнение команды завершено.
```

CreateParameter

Метод создает новый экземпляр объекта DbParameter.

Синтаксис

```
public DbParameter CreateParameter();
```

Возвращаемое значение

Объект `DbParameter`.

Исключения

Отсутствуют.

Пример

```
DbParameter p1 = command.CreateParameter();
```

ExecuteNonQuery

Метод выполняет запрос, не возвращающий данные. Он используется преимущественно тогда, когда запрос не должен возвращать значения или когда возвращенную выборку данных не нужно обрабатывать.

Данный метод рекомендуется использовать для выполнения следующих SQL-запросов:

- относящихся к операторам определения данных: создание/удаление/модификация объектов БД – таблиц, представлений, пользователей и т.п.;
- относящихся к операторам манипулирования данными – вставка/удаление/модификация записей;
- вызов хранимых процедур, не возвращающих данные;
- возвращающих не обрабатываемую клиентским приложением выборку данных.

Хотя метод ExecuteNonQuery и не возвращает ни одной строки, выходные параметры хранимых процедур заполняются данными.

Синтаксис

```
public abstract int ExecuteNonQuery();
```

Возвращаемое значение

Для запросов UPDATE, INSERT и DELETE возвращается количество строк, которые реально были обработаны с их помощью. Для всех прочих типов запросов возвращается `-1`.

Исключения

`InvalidOperationException`	Возможные причины: • текст команды не установлен; • команда не связана с соединением; • соединение не открыто; • команда не связана с транзакцией, открытой по соединению;

	• команда связана с открытым объектом `DataReader`.
`Exception`	Для входного параметра не установлено значение или для параметра длиной больше 4000 байт не установлен тип BLOB.
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

1) Выполнение команды DELETE

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ExecuteNonQuerySample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "delete from auto where personid=1001";
    // Выполнение SQL-запроса
    int rowsDeleted = cmd.ExecuteNonQuery();
    if (rowsDeleted > 0)
    {
      Console.WriteLine("Запись о пользователе с id=1001
 удалена");
    }
    else
    {
      Console.WriteLine("Пользователь с id=1001 не найден");
    }
    // Освобождение ресурсов
    cmd.Dispose();
    con.Dispose();
  }
```

```
}
```

2) Выполнение хранимой процедуры с выходным параметром

Примечание
В текущей версии ADO.NET провайдера параметры в команду надо добавлять в том порядке, в котором они идут в предложении CREATE PROCEDURE.

```
// В данном примере создается хранимая процедура PROC1, которая
 имеет
// два входных параметра A и B типа REAL и
// один выходной параметр C типа VARCHAR(50).
// В выходной параметр записывается результат деления A / B.
// Если B = 0, то в выходной параметр записывается значение NULL.
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class ExecuteNonQuerySample
{
  static void Main()
  {
    LinterDbConnection connection = null;

    try
    {
      // Создание объекта соединение
      connection = new
 LinterDbConnection("UserID=SYSTEM;Password=MANAGER8");

      // Создание объекта команда
      LinterDbCommand command = new LinterDbCommand();
      command.Connection = connection;
      command.CommandText =
      @"
create or replace procedure proc1(
  in a real;
  in b real;
  out c varchar(50))
code
  if b = 0 then
    c := NULL;
  else
    c := ""Результат деления a/b: "" + ftoa(a/b);
  endif
end;";
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
// Открытие соединения
connection.Open();

// Создание хранимой процедуры
command.ExecuteNonQuery();

// Настройка команды для выполнения хранимой процедуры
command.CommandText = "proc1";
command.CommandType = CommandType.StoredProcedure;

// Создание параметров

LinterDbParameter a = command.CreateParameter();
a.LinterDbType = ELinterDbType.Real;
a.Direction = ParameterDirection.Input;
a.ParameterName = "a";

LinterDbParameter b = command.CreateParameter();
b.LinterDbType = ELinterDbType.Real;
b.Direction = ParameterDirection.Input;
b.ParameterName = "b";

LinterDbParameter c = command.CreateParameter();
c.LinterDbType = ELinterDbType.VarChar;
c.Size = 50;
c.Direction = ParameterDirection.Output;
c.ParameterName = "c";

// Добавление параметров в коллекцию параметров команды
command.Parameters.Add(a);
command.Parameters.Add(b);
command.Parameters.Add(c);

// Трансляция запроса
command.Prepare();

// Установка значений входных параметров
a.Value = 5;
b.Value = 2;

// Выполнение хранимой процедуры
command.ExecuteNonQuery();

// Вывод на экран значения выходного параметра
```

```
      Console.WriteLine(Convert.IsDBNull(c.Value) ? "<NULL>" :
 c.Value);

      // Установка других значений входных параметров
      a.Value = 1;
      b.Value = 0;

      // Выполнение хранимой процедуры с новыми значениями входных
 параметров
      command.ExecuteNonQuery();

      // Вывод на экран нового значения выходного параметра
      Console.WriteLine(Convert.IsDBNull(c.Value) ? "<NULL>" :
 c.Value);
    }
    catch (LinterSqlException ex)
    {
      // Обработка исключений СУБД ЛИНТЕР
      Console.WriteLine(
        "Исключение ядра СУБД ЛИНТЕР \n" +
        "Текст сообщения: " + ex.Message + "\n" +
        "Код СУБД ЛИНТЕР: " + ex.Number + "\n" +
        "Код операционной системы: " + ex.LinterSysErrorCode +
 "\n");
    }
    catch (Exception ex)
    {
      // Обработка исключений других типов
      Console.WriteLine(
        "Исключение ADO.NET провайдера \n" +
        "Тип ошибки: " + ex.GetType() + "\n" +
        "Сообщение: " + ex.Message + "\n");
    }
    finally
    {
      // Освобождение ресурсов
      Console.WriteLine("Освобождение ресурсов.");
      if (connection != null)
      {
        connection.Close();
      }
    }
  }
}
Результат выполнения примера:
Результат деления a/b: 2.5
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
<NULL>
Освобождение ресурсов.
```

3) Выполнение группы запросов, разделенных знаком «;»

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class ExecuteNonQuerySample
{
  static void Main()
  {
    LinterDbConnection con = null;
    LinterDbDataReader reader = null;

    try
    {
      // Создание объекта соединение
      con = new
 LinterDbConnection("UserID=SYSTEM;Password=MANAGER8");

      // Создание объекта команда
      LinterDbCommand cmd = new LinterDbCommand();
      cmd.Connection = con;
      cmd.CommandText =
        "create or replace table tab1(c1 char(10), c2 char(10))";

      // Открытие соединения
      con.Open();

      // Создание таблицы в базе данных
      cmd.ExecuteNonQuery();

      // Формирование группы запросов, разделенных знаком ";"
      cmd.CommandText =
        "insert into tab1 (c1, c2) values ('AAA', :p1);" +
        "insert into tab1 (c1, c2) values ('BBB', :p2)";

      // Создание параметров
      LinterDbParameter p1 = cmd.CreateParameter();
      p1.LinterDbType = ELinterDbType.Char;
      p1.Size = 10;
      p1.ParameterName = "p1";

      LinterDbParameter p2 = cmd.CreateParameter();
```

```
      p2.LinterDbType = ELinterDbType.Char;
      p2.Size = 10;
      p2.ParameterName = "p2";

      // Добавление параметров в коллекцию параметров команды
      cmd.Parameters.Add(p1);
      cmd.Parameters.Add(p2);

      // Трансляция запроса
      cmd.Prepare();

      // Установка значений параметров
      p1.Value = "PPP1";
      p2.Value = "PPP2";

      // Выполнение запроса
      cmd.ExecuteNonQuery();

      // Формирование SQL-запроса для чтения данных из таблицы
      cmd.CommandText = "select c1, c2 from tab1";

      // Выполнение запроса чтения данных из таблицы
      reader = cmd.ExecuteReader();

      // Чтение данных из таблицы
      while (reader.Read())
      {
        Console.WriteLine(reader[0] + " | " + reader[1]);
      }
    }
    catch (LinterSqlException ex)
    {
      // Обработка исключений СУБД ЛИНТЕР
      Console.WriteLine(
        "Исключение ядра СУБД ЛИНТЕР \n" +
        "Текст сообщения: " + ex.Message + "\n" +
        "Код СУБД ЛИНТЕР: " + ex.Number + "\n" +
        "Код операционной системы: " + ex.LinterSysErrorCode +
 "\n");
    }
    catch (Exception ex)
    {
      // Обработка исключений других типов
      Console.WriteLine(
        "Исключение ADO.NET провайдера \n" +
        "Тип ошибки: " + ex.GetType() + "\n" +
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
            "Сообщение: " + ex.Message + "\n");
        }
        finally
        {
          // Освобождение ресурсов
          Console.WriteLine("Освобождение ресурсов.");
          if (reader != null)
          {
            reader.Close();
          }
          if (con != null)
          {
            con.Close();
          }
        }
      }
}
Результат выполнения примера:
AAA         | PPP1
BBB         | PPP2
Освобождение ресурсов.
```

ExecuteReader

Метод выполняет свойство CommandText по соединению Connection и помещает результаты (обычно выборку данных) в объект `DbDataReader` (работа с объектом `DbDataReader` описана в разделе Класс DbDataReader).

Примечание
В случае группы запросов, разделенных знаком «;», метод ExecuteReader возвращает объект DbDataReader для первого возвращающего данные запроса.

Если в свойстве CommandType указано, что будет выполняться хранимая процедура, то в свойстве CommandText должно быть указано имя этой процедуры. Метод `ExecuteReader` выполняет указанную процедуру, при этом если процедура возвращает курсор, то полученный объект `DbDataReader` можно использовать для чтения данных курсора.

Синтаксис

```
public DbDataReader ExecuteReader();
```

Возвращаемое значение

Результирующая выборка данных в виде объекта `System.Data.Common.DbDataReader`.

Примечание
Количество записей в полученной выборке узнать нельзя.

Исключения

`InvalidOperationException`	Возможные причины: • текст команды не установлен; • команда не связана с соединением; • соединение не открыто; • команда не связана с транзакцией, открытой по соединению; • команда связана с открытым объектом `DataReader`.
`Exception`	Для входного параметра не установлено значение или для параметра длиной больше 4000 байт не установлен тип BLOB.
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

1) Выполнение SELECT-запроса

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ExecuteReaderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "select distinct model, make from auto";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    while (reader.Read())
    {
```

```
      Console.WriteLine("Марка автомобиля : " +
 reader.GetString(0) +
        " Производитель : " + reader.GetString(1));
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

2) Выполнение процедуры с курсором

```
// В данном примере происходит выполнение временной хранимой
 процедуры, которая
// создаётся, компилируется и выполняется за один шаг, поэтому для
 её работы
// отпадает необходимость создания объекта БД «процедура» и
 наличие в БД системных
// таблиц $$$PROC и $$$PRCD. Для создания и выполнения временной
 хранимой
// процедуры используется оператор EXECUTE BLOCK.
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class ExecuteBlockSample
{
  static void Main()
  {
    LinterDbConnection connection = null;
    LinterDbDataReader reader = null;

    try
    {
      // Создание объекта соединение
      connection = new
 LinterDbConnection("UserID=SYSTEM;Password=MANAGER8");

      // Создание объекта команда
      LinterDbCommand command = new LinterDbCommand();
      command.Connection = connection;
      command.CommandText =
      @"
execute block result cursor(MAKE char(20), MODEL char(20),
 BODYTYPE char(15))
```

```
declare
 var a typeof(result);
code
 open a for ""select MAKE, MODEL, BODYTYPE from AUTO fetch first
 3;"";
 return a;
end;";

      // Открытие соединения
      connection.Open();

      Console.WriteLine("Выполнение команды...");
      // Выполнение временной хранимой процедуры, возвращающей
 курсор
      reader = command.ExecuteReader();

      Console.WriteLine("Результат выполнения запроса: ");
      // Получение данных из курсора
      while (reader.Read())
      {
        Console.WriteLine(reader[0] + " | " + reader[1] + " | " +
 reader[2]);
      }
      Console.WriteLine("Выполнение команды завершено.");
    }
    catch (LinterSqlException ex)
    {
      // Обработка исключений СУБД ЛИНТЕР
      Console.WriteLine(
        "Исключение ядра СУБД ЛИНТЕР \n" +
        "Текст сообщения: " + ex.Message + "\n" +
        "Код СУБД ЛИНТЕР: " + ex.Number + "\n" +
        "Код операционной системы: " + ex.LinterSysErrorCode +
 "\n");
    }
    catch (Exception ex)
    {
      // Обработка исключений других типов
      Console.WriteLine(
        "Исключение ADO.NET провайдера \n" +
        "Тип ошибки: " + ex.GetType() + "\n" +
        "Сообщение: " + ex.Message + "\n");
    }
    finally
    {
      // Освобождение ресурсов
```

```
      Console.WriteLine("Освобождение ресурсов.");
      if (reader != null)
      {
        reader.Close();
      }
      if (connection != null)
      {
        connection.Close();
      }
    }
  }
}
Результат выполнения примера:
Выполнение команды...
Результат выполнения запроса:
FORD                | MERCURY COMET GT V8   | COUPE
ALPINE              | A-310                 | COUPE
AMERICAN MOTORS     | MATADOR STATION       | STATION WAGON
Выполнение команды завершено.
Освобождение ресурсов.
```

ExecuteReader(CommandBehavior)

Метод выполняет свойство CommandText по соединению Connection и помещает результаты (обычно выборку данных) согласно заданным в аргументе CommandBehavior условиям в объект `DbDataReader` (работа с объектом `DbDataReader` описана в разделе Класс DbDataReader).

Синтаксис

```
public DbDataReader ExecuteReader(CommandBehavior behavior);
```

`behavior` – условие выборки (член перечисления System.Data.CommandBehavior):

Имя члена	Описание
`Default`	Запрос может вернуть несколько наборов результатов. Выполнение запроса может повлиять на состояние БД. Default не устанавливает флаги CommandBehavior, поэтому вызов ExecuteReader(CommandBehavior.Default) функционально эквивалентен вызову ExecuteReader().
`SingleResult`	Запрос возвращает только одну выборку данных. **Примечание** В текущей версии значение SingleResult игнорируется.
`SchemaOnly`	Запрос возвращает метаданные о столбцах выборки данных. Сам запрос фактически не выполняется.

Имя члена	Описание
	Примечание В текущей версии значение SchemaOnly игнорируется.
`KeyInfo`	Запрос возвращает данные, метаданные и дополнительную информацию о том, какие столбцы имеют атрибут AUTOINC и PRIMARY KEY. Если выборка получена в результате объединения нескольких таблиц, то некоторые значения метаданных могут отсутствовать.
	Примечание В текущей версии значение KeyInfo установлено всегда.
`SingleRow`	Предполагается, что выборка данных должна содержать только одну строку (указание этого условия повышает производительность работы клиентского приложения).
	Примечание В текущей версии значение SingleRow игнорируется.
`SequentialAccess`	Содержит указание для DataReader способа обработки строк, содержащих столбцы с большими двоичными значениями. Вместо загрузки всей строки полностью способ SequentialAccess позволяет DataReader загружать данные как поток. Затем можно использовать метод GetBytes или метод GetChars, чтобы указать положение байта для начала операции чтения и ограниченный размер буфера для возврата данных.
	Примечание В текущей версии значение SequentialAccess игнорируется.
`CloseConnection`	После закрытия объекта DataReader необходимо автоматически закрыть и связанное с ним соединение Connection.

Примечание
Перечисление имеет атрибут FlagsAttribute, поддерживающий побитовое соединение составляющих его значений.

Возвращаемое значение

Результирующая выборка данных в виде объекта `System.Data.Common.DbDataReader`.

© Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Примечание
Количество записей в выборке данных узнать нельзя.

Исключения

`InvalidOperationException`	Возможные причины: • текст команды не установлен; • команда не связана с соединением; • соединение не открыто; • команда не связана с транзакцией, открытой по соединению; • команда связана с открытым объектом DataReader.
`Exception`	Для входного параметра не установлено значение или для параметра длиной больше 4000 байт не установлен тип BLOB.
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ExecuteReaderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "select distinct model, make from auto";
    // Выполнение SQL-запроса
    DbDataReader reader =
 cmd.ExecuteReader(CommandBehavior.CloseConnection);
    // Обработка результатов запроса
    while (reader.Read())
```

```
    {
      Console.WriteLine("Марка автомобиля : " +
 reader.GetString(0) +
        " Производитель : " + reader.GetString(1));
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    // При создании DbDataReader указан флаг
 CommandBehavior.CloseConnection,
    // поэтому соединение закрывается автоматически
    Console.WriteLine("Состояние соединения: " + con.State);
  }
}
```

ExecuteScalar

Метод выполняет запрос и возвращает первый столбец первой строки результирующей выборки данных (все другие столбцы и строки выборки данных игнорируются). Метод обычно используется для извлечения отдельного значения (например, значения агрегатной функции) из БД. В этом случае он более эффективен, чем метод `ExecuteReader`.

Примечание
В случае группы запросов, разделенных знаком «;» метод ExecuteScalar возвращает значение первого поля первой записи из результатов первого запроса, который возвращает данные.

Синтаксис

```
public abstract Object ExecuteScalar();
```

Возвращаемое значение

Первый столбец первой строки результирующей выборки данных в виде объекта `System.Object` или null-значение, если требуемые данные не найдены.

Исключения

`InvalidOperationException`	Возможные причины: • текст команды не установлен; • команда не связана с соединением; • соединение не открыто; • команда не связана с транзакцией, открытой по соединению; • команда связана с открытым объектом DataReader.
`Exception`	Для входного параметра не установлено значение или для параметра длиной больше 4000 байт не установлен тип BLOB.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

LinterSqlException	Код завершения СУБД ЛИНТЕР не равен 0.

Пример

Выполнение функции COUNT(*)

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ExecuteScalarSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "select count(*) from auto";
    // Выполнение SQL-запроса
    Object result = cmd.ExecuteScalar();
    // Обработка результатов запроса
    Console.WriteLine("Количество строк в таблице AUTO: " +
 result);
    // Освобождение ресурсов
    cmd.Dispose();
    con.Dispose();
  }
}
```

```
        Результат выполнения примера:
        Количество строк в таблице AUTO: 1000
```

Prepare

Метод используется для подготовки параметризованных SQL-запросов или хранимых процедур с параметрами для последующего (возможно, многократного) выполнения. При первом вызове метод выполняет претрансляцию SQL-запроса (хранимой процедуры) и выполняет привязку параметров. При последующих вызовах метода с тем же SQL-запросом выполняется только привязка параметров. Вызов метода можно прописать в исходном коде приложения, и тогда на этапе выполнения

(ExecuteReader, ExecuteScalar, ExecuteNonQuery) буферы параметров будут привязаны к буферу запроса. Но если метод prepare() явно не вызывается в приложении, то он будет выполнен автоматически самим ядром СУБД ЛИНТЕР или ADO.NET-провайдером СУБД ЛИНТЕР.

Примечание
Если в свойстве CommandType указан тип команды TableDirect, метод Prepare не выполняется.

Синтаксис

```
public abstract void Prepare();
```

Возвращаемое значение

Значение типа void.

Исключения

InvalidOperationException	Текст команды не установлен или команда не связана с соединением или соединение не открыто.
LinterSqlException	Код завершения СУБД ЛИНТЕР не равен 0.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class PrepareSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText = "p_inout_param";
    cmd.CommandType = CommandType.StoredProcedure;
    // Создание входного параметра
    DbParameter inParam = factory.CreateParameter();
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    inParam.DbType = DbType.String;
    inParam.Direction = ParameterDirection.Input;
    inParam.Size = 255;
    inParam.ParameterName = ":ch_a";
    // Создание входного-выходного параметра
    DbParameter inOutParam = factory.CreateParameter();
    inOutParam.DbType = DbType.Int32;
    inOutParam.Direction = ParameterDirection.InputOutput;
    inOutParam.ParameterName = ":i_b";
    // Добавление параметров в коллекцию
    cmd.Parameters.Add(inParam);
    cmd.Parameters.Add(inOutParam);
    // Подготовка команды
    cmd.Prepare();
    // Выполнение команды при разных значениях параметров
    foreach (int i in new int[] { 1, 2, 3 })
    {
      inParam.Value = "Значение " + i;
      inOutParam.Value = i;
      cmd.ExecuteNonQuery();
      Console.WriteLine("Входное значение = {0} выходное = {1}",
        i, inOutParam.Value);
    }
    // Освобождение ресурсов
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

Входное значение = 1 выходное = 11
Входное значение = 2 выходное = 12
Входное значение = 3 выходное = 13
```

Класс DbDataReader

Класс `DbDataReader` предназначен для однонаправленного просмотра (но не для изменения!) предоставляемых данных. После перехода к следующей записи выборки данных текущая запись становится недоступной.

Чтобы создать объект `DbDataReader` без применения конструктора, необходимо вызвать метод `ExecuteReader` объекта `DbCommand`.

Для освобождения объекта DbDataReader надо использовать оператор using или блок try...finally (см. приложение 1).

Свойства класса приведены в таблице 13.

Таблица 13. Свойства класса DbDataReader

Свойство	Описание
Depth	Уровень вложенности текущей строки выборки данных (в иерархических запросах).
FieldCount	Количество столбцов (с учетом скрытых столбцов) в выборке данных.
HasRows	Индикатор наличия строк в выборке данных (т.е. позволяет проверить, пуста или нет выборка данных).
IsClosed	Индикатор активности выборки данных.
Item(Int32)	Значение столбца в текущей строке выборки данных по его порядковому номеру.
Item(String)	Значение столбца в текущей строке выборки данных по его имени.
RecordsAffected	Количество реально обработанных записей в БД после выполнения последнего SQL-запроса манипулирования данными (INSERT, DELETE, UPDATE).
VisibleFieldCount	Количество видимых (не скрытых) столбцов в текущей выборке данных.
IsBeforeReadState	Индикатор возможности работы с полями выборки данных.

Методы класса приведены в таблице 14.

Таблица 14. Методы класса DbDataReader

Метод	Описание
Close	Закрывает объект `DbDataReader`.
GetBoolean	Предоставляет приведенное к типу данных boolean значение указанного поля текущей строки выборки данных.
GetByte	Предоставляет приведенное к типу данных byte значение указанного поля текущей строки выборки данных.
GetBytes	Предоставляет приведенный к типу данных byte[] массив данных указанного поля текущей строки выборки данных.
GetChar	Предоставляет приведенный к типу данных char (в UTF-16 кодировке) первый символ указанного поля текущей строки выборки данных.
GetChars	Предоставляет приведенный к типу данных char[] массив данных указанного поля текущей строки выборки данных.
GetData	Предоставляет объект `DbDataReader` указанного поля текущей строки выборки данных.
GetDataTypeName	Предоставляет тип данных указанного поля текущей строки выборки данных
GetDateTime	Предоставляет значение типа DateTime указанного поля текущей строки выборки данных.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Метод	Описание
GetDecimal	Предоставляет значение типа Decimal указанного поля текущей строки выборки данных.
GetDouble	Предоставляет значение типа Double указанного поля текущей строки выборки данных.
GetEnumerator	Предоставляет перечислитель, используемый для перебора элементов в коллекции данных.
GetFieldType	Предоставляет тип данных указанного поля текущей строки выборки данных в формате .NET.
GetFloat	Предоставляет значение типа Float указанного поля текущей строки выборки данных.
GetGuid	Предоставляет значение в виде глобального уникального идентификатора (GUID) указанного поля текущей строки выборки данных.
GetInt16	Предоставляет значение в виде 16-битового целого числа со знаком указанного поля текущей строки выборки данных.
GetInt32	Предоставляет значение в виде 32-битового целого числа со знаком указанного поля текущей строки выборки данных.
GetInt64	Предоставляет значение в виде 64-битового целого числа со знаком указанного поля текущей строки выборки данных.
GetLinterBlobForUpdate	Предоставляет экземпляр класса `LinterBlob` для работы с BLOB-данными.
GetName	Предоставляет имя указанного поля текущей строки выборки данных.
GetOrdinal	Предоставляет порядковый номер указанного именованного поля текущей строки выборки данных.
GetProviderSpecificFieldType	Предоставляет тип данных указанного поля текущей строки выборки данных в терминах СУБД ЛИНТЕР
GetProviderSpecificValue	Предоставляет значение указанного поля текущей строки выборки данных в виде экземпляра класса `Object`.
GetProviderSpecificValues	Предоставляет массив значений всех полей текущей строки выборки данных.
GetSchemaTable	Предоставляет метаданные текущей выборки данных.
FastGetSchemaTable	Аналогичен методу `GetSchemaTable` (отличие в способе получения метаданных из БД).
GetString	Предоставляет символьное значение указанного поля текущей строки выборки данных.
GetValue	Предоставляет значение указанного поля текущей строки выборки данных в виде .NET-объекта.
GetValues	Предоставляет массив значений полей текущей строки выборки данных в виде .NET-объектов.
IsDBNull	Индикатор null-значения указанного поля текущей строки выборки данных.

Метод	Описание
NextResult	Выполняет переход к следующей выборке данных (в случае пакетного выполнения SQL-запросов).
Read	Предоставляет доступ к полям текущей строки выборки данных и перемещает указатель текущей строки выборки данных на следующую строку.

Свойства

Depth

Свойство предоставляет уровень вложенности текущей строки выборки данных (имеет смысл только для иерархических запросов).

Декларация

```
public abstract int Depth {get;};
```

Значение свойства

Значение типа System.Int32. Для иерархических запросов уровень вложенности равен значению псевдостолбца LEVEL в иерархическом запросе (см. документ «Справочник по SQL»). Для остальных запросов значение всегда `0`.

Примечание
В текущей версии ADO.NET-провайдера не поддерживается (всегда возвращается `0`).

Исключения

Отсутствуют.

FieldCount

Свойство содержит данные о количестве столбцов (с учетом скрытых столбцов) в выборке данных.

Примечание
Для исключения подсчета скрытых столбцов надо использовать свойство VisibleFieldCount.

Декларация

```
public abstract int FieldCount {get;};
```

Значение свойства

Количество столбцов в выборке данных (значение типа System.Int32) или `0`, если запрос (типа INSERT, DELETE) не возвращает выборку данных.

Примечание
Если выборка данных пуста (например, возвращается `0` строк в запросе `select * from table where 1 = 2`), то FieldCount возвращает количество столбцов в таблице.

© Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class FieldCountSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "select personid, make, model from auto";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    Console.WriteLine("Кол-во столбцов в выборке данных: {0}",
      reader.FieldCount);
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

HasRows

Свойство информирует о наличии (но не о количестве!) записей в выборке данных (т.е. позволяет проверить, пуста или нет выборка данных). Значение не меняется при перемещении по выборке данных.

Декларация

```
public abstract bool HasRows {get;};
```

Значение свойства

Значение типа System.Boolean:

- true – выборка данных содержит одну или более строк;
- false – выборка данных пуста.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class HasRowsSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "select * from auto where personid = 500";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    if (reader.HasRows)
    {
      Console.WriteLine("Выборка данных не пуста. ");
    }
    else
    {
      Console.WriteLine("Выборка данных пуста.");
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
```

```
    con.Dispose();
  }
}
```

IsClosed

Свойство информирует о текущем состоянии выборки данных (активна или закрыта).

Декларация

```
public abstract bool IsClosed {get;};
```

Значение свойства

Значение типа System.Boolean:

- true – выборка данных закрыта (значение по умолчанию);
- false – выборка данных активна.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class IsClosedSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "select * from auto";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    if (!reader.IsClosed)
```

```
    {
      // Выполняем обработку данных
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

Item(Int32)

Свойство предоставляет значение столбца в текущей строке выборки данных по его порядковому номеру.

Декларация

```
public abstract Object this [int ordinal] {get;};
```

`ordinal` – порядковый номер столбца в выборке данных (отсчет начинается с `0`).

Значение по умолчанию отсутствует.

Значение свойства

Значение заданного столбца выборки данных в виде System.Object (т.е. в исходном формате).

Исключения

`IndexOutOfRangeException`	Столбец с указанным порядковым номером не существует.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Text;

class ItemInt32Sample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
```

```
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "select personid, model, make from auto";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    StringBuilder sb = new StringBuilder();
    while (reader.Read())
    {
      sb.AppendFormat("Владелец авто {0} марка авто {1}
 производитель {2}",
        reader[0], reader[1], reader[2]);
      sb.AppendLine();
    }
    Console.WriteLine(sb.ToString());
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

Item(String)

Свойство предоставляет значение столбца в текущей строке выборки данных по его имени. К первому неименованному столбцу можно обратиться по имени String.Empty. К другим неименованным столбцам обратиться нельзя.

Декларация

```
public abstract Object this [string name] {get;};
```

`name` – имя столбца в выборке данных (значение по умолчанию отсутствует).

Вначале выполняется поиск столбца по имени с учетом регистра. В случае неудачи производится повторный поиск уже без учета регистра.

Значение свойства

Значение заданного столбца выборки данных в виде System.Object (т.е. в исходном формате).

Исключения

`IndexOutOfRangeException`	Столбец с указанным именем не существует.

Пример

```
// C#
using System;
```

```
using System.Data;
using System.Data.Common;
using System.Text;

class ItemStringSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "select personid, make, model from auto";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    StringBuilder sb = new StringBuilder();
    while (reader.Read())
    {
      sb.AppendFormat("Владелец авто {0} марка авто {1}
 производитель {2}",
        reader["personid"], reader["model"], reader["make"]);
      sb.AppendLine();
    }
    Console.WriteLine(sb.ToString());
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

RecordsAffected

Свойство содержит количество реально обработанных записей в БД после выполнения последнего SQL-запроса манипулирования данными (INSERT, DELETE, UPDATE).

Декларация

```
public abstract int RecordsAffected {get;};
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Значение свойства

Значение типа System.Int32:

- количество измененных, вставленных или удаленных строк;
- количество строк в выборке данных, полученных SELECT-запросом;

Примечание
Поддерживается только при работе с СУБД ЛИНТЕР (для других СУБД возвращается `-1`).

- 0, если строки не изменены.

Примечание
Свойству RecordsAffected значение присваивается только после закрытия соответствующего объекта `DataReader`.

Примечание
В текущей версии ADO.NET-провайдера свойство RecordsAffected можно использовать только для команд, содержащих один SQL-запрос (для пакета запросов, разделенных символом ";" свойство не поддерживается).

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class RecordsAffectedSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "update auto set model='LADA KALINA' where
 make='FORD'";
```

```
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    reader.Close();
    Console.WriteLine("Кол-во измененных строк " +
 reader.RecordsAffected);
    // Освобождение ресурсов
    cmd.Dispose();
    con.Dispose();
  }
}
```

VisibleFieldCount

Свойство содержит количество видимых (не скрытых) столбцов в текущей выборке данных.

Декларация

```
public virtual int VisibleFieldCount {get;};
```

Значение свойства

Количество видимых столбцов в выборке данных (значение типа System.Int32) или 0, если запрос (типа INSERT, UPDATE, DELETE) не возвращает выборку данных.

Примечание
В текущей версии ADO.NET-провайдера механизм скрытия столбцов выборки данных не поддерживается, поэтому значение свойства VisibleFieldCount всегда совпадает со значением свойства FieldCount.

Исключения

Отсутствуют.

IsBeforeReadState

Свойство информирует о возможности работы с полями выборки данных.

Сразу после открытия выборки данных (т.е. после выполнения одного из методов `DbCommand.ExecuteReader()`…) доступ к полям (значениям) этой выборки невозможен – необходимо предварительно выполнить хотя бы один раз метод `Read()`, который устанавливает текущую строку выборки данных. Поэтому перед вызовом методов, работающих с полями выборки данных, необходимо убедиться, что метод `Read()` применительно к текущей выборке данных был вызван.

Примечание
Свойство IsBeforeReadState можно использовать только в приложениях, разработанных специально для работы с СУБД ЛИНТЕР.

Декларация

```
public bool IsBeforeReadState {get;};
```


© Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Значение свойства

Значение типа System.Boolean:

- true – значения полей выборки данных доступны, т.е. метод `Read`() отработал;
- false – значения полей выборки данных недоступны, необходимо предварительно вызвать метод `Read`() (значение по умолчанию).

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Data.LinterClient;

class IsBeforeReadStateSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8;Data
 Source=LOCAL";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "select * from auto";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    if (((LinterDbDataReader)reader).IsBeforeReadState)
    {
      reader.Read();
    }
    for (int i = 0; i < reader.FieldCount; i++)
    {
      Console.Write(reader.GetValue(i) + "\t");
    }
    // Освобождение ресурсов
```

```
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

Методы

Close

Метод закрывает объект `DbDataReader`.

Всегда нужно вызывать метод Close по окончании работы с объектом `DbDataReader`, потому что в противном случае объект `DbDataReader` будет удерживать ресурсы до тех пор, пока не будет закрыто соединение, которое используется для выполнения команды.

Для закрытия объекта DbDataReader надо использовать оператор using или блок `try...finally` (см. приложение 1).

Синтаксис

```
public abstract void Close();
```

Возвращаемое значение

Значение типа void.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CloseSample
{
  static void Main()
  {
    DbProviderFactory factory = null;
    DbConnection con = null;
    DbCommand cmd = null;
    DbDataReader reader = null;
    try
    {
      // Создание фабрики классов провайдера
      factory =
 DbProviderFactories.GetFactory("System.Data.LinterClient");
      // Соединение с БД
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
      con = factory.CreateConnection();
      con.ConnectionString =
        "User ID=SYSTEM;Password=MANAGER8;Data Source=LOCAL";
      con.Open();
      // Создание объекта DbCommand
      cmd = factory.CreateCommand();
      cmd.Connection = con;
      // Формирование текста SQL-запроса
      cmd.CommandText = "select * from some_table";
      // Выполнение SQL-запроса
      reader = cmd.ExecuteReader();
      // Обработка результатов запроса
      while (reader.Read())
      {
        for (int i = 0; i < reader.FieldCount; i++)
        {
          Console.Write(reader.GetValue(i) + "\t");
        }
        Console.WriteLine();
      }
    }
    catch (Exception ex)
    {
      // Обработка исключений
      Console.WriteLine(ex);
    }
    finally
    {
      // Освобождение ресурсов
      if (reader != null)
      {
        reader.Close();
      }
      if (cmd != null)
      {
        cmd.Dispose();
      }
      if (con != null)
      {
        con.Close();
      }
    }
  }
}
```

GetBoolean

Метод предоставляет приведенное к типу данных boolean значение указанного поля. Т.к. в текущей версии ADO.NET-провайдера приведение типов не выполняется, то корректно метод может применяться только к полям с типом данных boolean, иначе будет выдано исключение.

Синтаксис

```
public abstract bool GetBoolean(int ordinal);
```

`ordinal` – порядковый номер столбца (отсчет начинается с `0`).

Возвращаемое значение

Значение типа System.Boolean указанного поля.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных Boolean.
`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetBoolean
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText =
      "Select cast year-70 as boolean from auto limit 3";
    // Выполнение SQL-запроса
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    Boolean value;
    while (reader.Read())
    {
      value = reader.GetBoolean(0);
      Console.WriteLine(value);
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

True
False
True
```

GetByte

Метод предоставляет приведенное к типу данных byte значение указанного поля текущей строки выборки данных. Т.к. в текущей версии ADO.NET-провайдера приведение типов не выполняется, то корректно метод может применяться только к полям с типом данных byte, varbyte, иначе будет выдано исключение.

Синтаксис

```
public abstract byte GetByte(int ordinal);
```

`ordinal` – порядковый номер столбца (отсчет начинается с `0`).

Возвращаемое значение

Значение типа System.Byte указанного поля.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных byte.
`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
```

```
class GetByte
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText =
      "Select hex('134da75ff'), cast -0x23ca as byte(2)";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    Byte value1;
    Byte value2;
    while (reader.Read())
    {
      value1 = reader.GetByte(0);
      value2 = reader.GetByte(1);
      Console.WriteLine(value1);
      Console.WriteLine(value2);
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

19
54
```

GetBytes

Метод предоставляет приведенный к типу данных byte[] массив данных указанного поля текущей строки выборки данных. Т.к. в текущей версии ADO.NET-провайдера

приведение типов не выполняется, то корректно метод может применяться только к полям с типом данных byte, varbyte, blob, иначе будет выдано исключение.

Синтаксис

```
public abstract long GetBytes(
    int ordinal,
    long dataOffset,
    byte[] buffer,
    int bufferOffset,
    int length
);
```

`ordinal` – источник данных (порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`)).

`dataOffset` – местоположение извлекаемых данных в источнике данных (номер позиции в указанном поле текущей строки, начиная с которой должны извлекаться данные (отсчет начинается с `0`)).

`buffer` – приемник данных (буфер в памяти для размещения извлеченных данных).

`bufferOffset` – местоположение данных в приемнике данных (номер позиции в приемнике данных, начиная с которой должно выполняться размещение извлеченных данных (отсчет начинается с `0`)).

`length` – количество запрашиваемых данных в байтах.

Возвращаемое значение

Количество извлеченных байтов (может отличаться от количества запрошенных байтов, если источник данных не может их предоставить в полном объеме или достигнут конец буфера приемника).

Если значение `buffer` равно null, то возвращается длина запрошенного поля (независимо от заданного в нем смещения). Сами данные не предоставляются.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных byte[].
`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).
`ArgumentException`	Местоположение в источнике за пределами поля.
`ArgumentException`	Местоположение в приемнике за пределами буфера.
`ArgumentException`	Недопустимая длина.

Примеры

1)

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Text;

class GetBytes
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText =
      "Select getraw('ADO.NET-провайдер СУБД ЛИНТЕР', 18, 11)";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    Byte[] value = new Byte[11];
    while (reader.Read())
    {
      reader.GetBytes(0, 0, value, 0, 11);
    }
    Console.WriteLine(Encoding.Default.GetString(value));
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

СУБД ЛИНТЕР
```

2) Извлечение данных большого объема в файл.

```
// C#
```

```
using System;
using System.Data;
using System.Data.Common;
using System.Text;
using System.IO;

class GetBytes
{
  static void Main()
  {
    // Создание фабрики классов провайдера.
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД.
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Предположим что текстовые данные (рассказы) хранятся в
 BLOB-столбце.
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText = "SELECT publisher, story FROM biblioteka";
    // Запись извлекаемых данных в файл.
    FileStream stream;
    BinaryWriter writer;
    // Размер буфера.
    int bufferSize = 100;
    // Буфер, который будет заполнен методом GetBytes.
    byte[] outBytes = new byte[bufferSize];
    // Количество байт, полученных методом GetBytes.
    long retval;
    // Начальная позиция в буфере.
    long startIndex = 0;
    // Идентификатор издателя в имени файла.
    string pubID = "";
    // Установка соединения и получение данных в DataReader.
    con.Open();
    DbDataReader reader =
 cmd.ExecuteReader(CommandBehavior.SequentialAccess);
    while (reader.Read())
    {
      // Получение идентификатора издателя.
      pubID = reader.GetString(0);
      // Создание выходного файла.
      stream = new FileStream(
```

```
        "story" + pubID + ".txt", FileMode.OpenOrCreate,
 FileAccess.Write);
      writer = new BinaryWriter(stream);
      // Восстановление начальной позиции.
      startIndex = 0;
      // Чтение данных в буфер.
      retval = reader.GetBytes(1, startIndex, outBytes, 0,
 bufferSize);
      // Продолжаем пока есть данные за пределами буфера.
      while (retval == bufferSize)
      {
        writer.Write(outBytes);
        writer.Flush();
        // Перемещаем начальный индекс в конец последнего буфера и
 заполняем буфер.
        startIndex += bufferSize;
        retval = reader.GetBytes(1, startIndex, outBytes, 0,
 bufferSize);
      }
      // Записываем в файл оставшийся буфер.
      writer.Write(outBytes, 0, (int)retval - 1);
      writer.Flush();
      // Закрываем выходной файл.
      writer.Close();
      stream.Close();
    }
    // Закрываем DataReader и соединение.
    reader.Close();
    con.Close();
  }
}
```

GetChar

Метод предоставляет приведенный к типу данных char (в UTF-16 кодировке) первый символ указанного поля текущей строки выборки данных. Т.к. в текущей версии ADO.NET-провайдера приведение типов не выполняется, то корректно метод может применяться только к полям с типом данных char, varchar, nchar, nvarchar, иначе будет выдано исключение.

Синтаксис

```
public abstract char GetChar(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Возвращаемое значение

Первый символ указанного поля текущей выборки данных.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных char.
`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

Примеры

1)

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetChar
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "Select 'ЛИНТЕР'";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    Char value;
    while (reader.Read())
    {
      value = reader.GetChar(0);
      Console.WriteLine(value);
    }
    // Освобождение ресурсов
```

```
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

```
Результат выполнения примера:

Л
```

2) Подсчитать в выборке данных (из таблицы AUTO) количество автомобилей, название которых начинается с буквы F, и вывести их список на консоль.

Примечание
Пример приведен только для иллюстрации использования метода, т.к. данная информация может быть получена с помощью агрегатной функции `count`() в SELECT-запросе.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetChar
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "Select model from auto";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    int count = 0;
    Console.WriteLine("Автомобили, название которых начинаются с
 буквы F:");
    while (reader.Read())
    {
```

```
      if (reader.GetChar(0) == 'F')
      {
        Console.WriteLine(reader.GetString(0));
        count++;
      }
    }
    Console.WriteLine("Количество авто, название которых
 начинаются с буквы F: " + count);
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

```
Результат выполнения примера:

Автомобили, название которых начинаются с буквы F:
FULVIA SPORT 1600
FULVIA SPORT 1600
FULVIA SPORT 1600
FULVIA SPORT 1600
FULVIA SPORT 1600
FULVIA SPORT 1600
FULVIA SPORT 1600
FULVIA SPORT 1600
Количество авто, название которых начинаются с буквы F: 8
```

GetChars

Метод предоставляет приведенный к типу данных char[] (в UTF-16 кодировке) массив данных указанного поля текущей строки выборки данных. Т.к. в текущей версии ADO.NET-провайдера приведение типов не выполняется, то корректно метод может применяться только к полям с типом данных char, varchar, nchar, nvarchar, иначе будет выдано исключение.

Синтаксис

```
public abstract long GetChars(
    int ordinal,
    long dataOffset,
    char[] buffer,
    int bufferOffset,
    int length
);
```

`ordinal` – источник данных (порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`)).

`dataOffset` – местоположение извлекаемых данных в источнике данных (номер позиции в указанном поле текущей строки, начиная с которой должны извлекаться данные (отсчет начинается с `0`)).

`buffer` – приемник данных (буфер в памяти для размещения извлеченных данных).

`bufferOffset` – местоположение данных в приемнике данных (номер позиции в приемнике данных, начиная с которой должно выполняться размещение извлеченных данных (отсчет начинается с `0`)).

`length` – количество запрашиваемых данных в символах.

Возвращаемое значение

Количество извлеченных символов (может отличаться от количества запрошенных символов, если источник данных не может их предоставить в полном объеме или достигнут конец буфера приемника).

Если значение `buffer` равно null, то возвращается длина запрошенного поля (независимо от заданного в нем смещения). Сами данные не предоставляются.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных char[].
`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).
`ArgumentException`	Местоположение в источнике за пределами поля.
`ArgumentException`	Местоположение в приемнике за пределами буфера.
`ArgumentException`	Недопустимая длина.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Text;

class GetChars
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
```

```
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText =
      "Select 'ADO.NET-провайдер СУБД ЛИНТЕР'";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    Char[] value = new Char[11];
    while (reader.Read())
    {
      reader.GetChars(0, 18, value, 0, 11);
    }
    Console.WriteLine(new String(value));
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

СУБД ЛИНТЕР
```

GetData

Метод предоставляет объект `DbDataReader` указанного поля текущей строки выборки данных.

Примечание
В текущей версии ADO.NET-провайдера не поддерживается. При вызове метода генерируется исключение `NotSupportedException`.

Синтаксис

```
public DbDataReader GetData(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Объект `DbDataReader`, соответствующий запрошенному полю текущей строки выборки данных.

Исключения

`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.

`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

GetDataTypeName

Метод предоставляет тип данных указанного поля текущей строки выборки данных.

Синтаксис

```
public abstract string GetDataTypeName(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Тип данных указанного поля текущей строки выборки данных (System.String).

Возможные значения:

- Char;
- Smallint;
- Int;
- Bigint;
- Real;
- Double;
- Date;
- Numeric;
- Byte;
- Blob;
- VarChar;
- VarByte;
- Bool;
- NChar;
- NVarChar;
- ExtFile.

Исключения

`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetDataTypeName
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "Select count(*) from auto";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    string datatype = reader.GetDataTypeName(0);
    Console.WriteLine(datatype);
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

Int
```

GetDateTime

Метод предоставляет значение типа DateTime указанного поля текущей строки выборки данных.

Метод применяется только к столбцам типа DATE.

Синтаксис

```
public abstract DateTime GetDateTime(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Значение типа System.DateTime указанного поля текущей строки выборки данных.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных DateTime.
`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetDateTime
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "Select sysdate";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    Console.WriteLine(DateTime.Now);
    DateTime value;
    while (reader.Read())
    {
      value = reader.GetDateTime(0);
      Console.WriteLine(value);
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
}

Результат выполнения примера:

03.09.2012 13:41:57
03.09.2012 9:41:57
```

GetDecimal

Метод предоставляет значение типа Decimal указанного поля текущей строки выборки данных.

Метод применяется только к столбцам типа DECIMAL (NUMERIC).

Синтаксис

```
public abstract decimal GetDecimal(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Значение типа System.Decimal указанного поля текущей строки выборки данных.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных decimal.
`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetDecimal
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
```

```
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText =
      "Select distinct cast year+1900 as decimal from auto limit
 2";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    decimal value;
    while (reader.Read())
    {
      value = reader.GetDecimal(0);
      Console.WriteLine(value);
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

1970
1971
```

GetDouble

Метод предоставляет значение типа Double указанного поля текущей строки выборки данных.

Метод применяется только к полям с типом DOUBLE.

Синтаксис

```
public abstract double GetDouble(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Значение типа System.Double указанного поля текущей строки выборки данных.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных Double.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

IndexOutOfRangeException	Задан порядковый номер несуществующего столбца.
InvalidOperationException	Не установлена текущая строка выборки данных (необходимо выполнить метод Read()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetDouble
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "Select cast 100 as double";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    double value;
    while (reader.Read())
    {
      value = reader.GetDouble(0);
      Console.WriteLine(value);
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

100
```

GetEnumerator

Метод предоставляет перечислитель, используемый для перебора элементов в коллекции данных.

Синтаксис

```
public abstract IEnumerator GetEnumerator();
```

Возвращаемое значение

Перечислитель (объект типа `System.Collections.IEnumerator`), который можно использовать для перемещения по строкам выборки данных.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Collections;

class GetEnumerator
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "Select make from auto limit 2";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    IEnumerator enumerator = reader.GetEnumerator();
    while (enumerator.MoveNext())
    {
      DbDataRecord dataRecord = (DbDataRecord)enumerator.Current;
      Console.WriteLine(dataRecord.GetString(0));
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

FORD
ALPINE
```

GetFieldType

Метод предоставляет тип данных указанного поля текущей строки выборки данных в формате .NET (например, тип данных INTEGER СУБД ЛИНТЕР будет представлен как System.Int32).

Синтаксис

```
public abstract Type GetFieldType(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Значение типа System.Type указанного поля текущей строки выборки данных.

Соответствие типов данных СУБД ЛИНТЕР и .NET:

СУБД ЛИНТЕР	**Аналог в .NET**
INTEGER	System.Int32
SMALLINT	System.Int16
BIGINT	System.Int64
BYTE	System.Byte[]
VARBYTE	System.Byte[]
REAL	System.Single
DOUBLE	System.Double
BOOLEAN	System.Boolean
CHAR	System.String
VARCHAR	System.String
NCHAR	System.String
NVARCHAR	System.String
DECIMAL	System.Decimal
BLOB	System.Byte[]
FLOAT	System.Single или System.Double в зависимости от точности FLOAT (<точность>)

СУБД ЛИНТЕР	Аналог в .NET
DATE	System.DateTime
EXTFILE	System.String

Исключения

IndexOutOfRangeException	Задан порядковый номер несуществующего столбца.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetFieldType
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "Select sysdate, 45.67";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    Type dataType0 = reader.GetFieldType(0);
    Type dataType1 = reader.GetFieldType(1);
    Console.WriteLine(dataType0);
    Console.WriteLine(dataType1);
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
System.DateTime
System.Decimal
```

GetFloat

Метод предоставляет вещественное значение одиночной точности типа float указанного поля текущей строки выборки данных.

Метод применяется только к полям с типом данных real СУБД ЛИНТЕР.

Синтаксис

```
public abstract float GetFloat(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Значение типа System.Single указанного поля текущей строки выборки данных.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных float.
`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetFloat
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
```

```
    // Формирование текста SQL-запроса
    cmd.CommandText =
      "Select cast 0.5 as real, cast 0xcf4f as real, cast .57e+4
 as real";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    float value0;
    float value1;
    float value2;
    while (reader.Read())
    {
      value0 = reader.GetFloat(0);
      value1 = reader.GetFloat(1);
      value2 = reader.GetFloat(2);
      Console.WriteLine(value0);
      Console.WriteLine(value1);
      Console.WriteLine(value2);
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

System.DateTime
System.Decimal
```

GetGuid

Метод предоставляет значение в виде глобального уникального идентификатора (GUID) указанного поля текущей строки выборки данных.

Метод применяется к полю со значением GUID (в СУБД ЛИНТЕР значение GUID можно получить с помощью встроенной функции `SYS_GUID`()).



Примечание
GUID является 128-разрядным (16 байт) целым числом, которое может быть использовано во всех компьютерах и сетях, когда необходим уникальный идентификатор.

Синтаксис

```
public abstract Guid GetGuid(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Возвращаемое значение

Значение типа System.Guid указанного поля текущей строки выборки данных.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных Guid.
`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetGuid
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "Select sys_guid()";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    Guid value;
    while (reader.Read())
    {
      value = reader.GetGuid(0);
      Console.WriteLine(value);
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
```

```
    con.Dispose();
  }
}
```

```
Результат выполнения примера:

69483f8a-af2c-64aa-fd67-382b7e2cd48b
```

GetInt16

Метод предоставляет значение в виде 16-битового целого числа со знаком указанного поля текущей строки выборки данных.

Метод применяется к полю с типом данных smallint СУБД ЛИНТЕР.

Синтаксис

```
public abstract short GetInt16(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Значение типа System.Int16 указанного поля текущей строки выборки данных.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных Int16.
`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetInt16
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText =
      "Select cast personid as smallint, " +
      "cast year + 1900 as smallint, " +
      "cast 5 as smallint from auto limit 2";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    Int16 value0;
    Int16 value1;
    Int16 value2;
    while (reader.Read())
    {
      value0 = reader.GetInt16(0);
      value1 = reader.GetInt16(1);
      value2 = reader.GetInt16(2);
      Console.WriteLine("| {0} | {1} | {2} |",
        value0, value1, value2);
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

```
Результат выполнения примера:

| 1 | 1971 | 5 |
| 2 | 1970 | 5 |
```

GetInt32

Метод предоставляет значение в виде 32-битового целого числа со знаком указанного поля текущей строки выборки данных.

Метод применяется к полю с типом данных integer СУБД ЛИНТЕР.

Синтаксис

```
public abstract int GetInt32(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Значение типа System.Int32 указанного поля текущей строки выборки данных.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных Int32.
`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetInt32
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "Select 35467, 4585";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    Int32 value0;
    Int32 value1;
    while (reader.Read())
    {
      value0 = reader.GetInt32(0);
      value1 = reader.GetInt32(1);
      Console.WriteLine("| {0} | {1} |", value0, value1);
    }
    // Освобождение ресурсов
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

| 35467| 4585|
```

GetInt64

Метод предоставляет значение в виде 64-битового целого числа со знаком указанного поля текущей строки выборки данных.

Метод применяется к полю с типом данных bigint СУБД ЛИНТЕР.

Синтаксис

```
public abstract long GetInt64(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Значение типа System.Int64 указанного поля текущей строки выборки данных.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных Int64.
`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetInt64
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
```

```
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "Select 36854775808, -36854775808";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    Int64 value0;
    Int64 value1;
    while (reader.Read())
    {
      value0 = reader.GetInt64(0);
      value1 = reader.GetInt64(1);
      Console.WriteLine("| {0} | {1} |", value0, value1);
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

```
Результат выполнения примера:

| 36854775808 | -36854775808 |
```

GetLinterBlobForUpdate

Метод возвращает экземпляр класса `LinterBlob`, который позволяет добавлять данные в BLOB-значение небольшими порциями, при этом не нужно хранить в оперативной памяти BLOB-значение целиком. Он эффективен в тех случаях, когда нужно добавить к BLOB-значению порцию данных большого размера.

Алгоритм работы следующий:

- выполнить запрос на выборку данных методом `command.ExecuteReader()`;
- перейти к требуемой записи выборки методом `reader.Read()`;
- получить объект `LinterBlob` методом `reader.GetLinterBlobForUpdate()`;
- в цикле добавлять порции данных к BLOB-значению методом `linterBlob.Append()`.

Синтаксис

```
LinterBlob GetLinterBlobForUpdate(int i);
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

i – порядковый номер BLOB-поля в текущей строке (отсчет начинается с 0).

Возвращаемое значение

Значение типа LinterBlob.

Исключения

IndexOutOfRangeException	Задан порядковый номер несуществующего столбца.
InvalidOperationException	Не установлена текущая строка выборки данных (необходимо выполнить метод Read()).

Пример

```
// C#
using System;
using System.Data.LinterClient;

class GetLinterBlobForUpdateSample
{
  static void Main()
  {
    LinterDbConnection conn = null;
    try
    {
      // Соединение с БД
      conn = new
 LinterDbConnection("UserID=SYSTEM;Password=MANAGER8");
      conn.Open();
      // Создание таблицы
      LinterDbCommand cmd = conn.CreateCommand();
      cmd.CommandText =
        "create or replace table TEST_BLOB (ID int, BLOB_COLUMN
 blob)";
      cmd.ExecuteNonQuery();
      // Добавление записи в таблицу
      cmd.CommandText =
        "insert into TEST_BLOB (ID, BLOB_COLUMN) values (1,
 hex('0102030405'))";
      cmd.ExecuteNonQuery();
      // Получение первой записи
      cmd.CommandText = "select BLOB_COLUMN, ID from TEST_BLOB
 where ID = 1";
      LinterDbDataReader reader = cmd.ExecuteReader();
      reader.Read();
      // Получение объекта LinterBlob для работы с BLOB
      LinterBlob blob = reader.GetLinterBlobForUpdate(0);
      // Очистка BLOB
      blob.Clear();
```

```
      // Добавление данных в BLOB
      byte[] bytes = new byte[] { 6, 7, 8, 9 };
      for (int i = 1; i < 100; i++)
      {
        blob.Append(bytes, 0, bytes.Length);
      }
      // Освобождение DataReader
      reader.Dispose();
      Console.WriteLine("Поле BLOB успешно обновлено.");
    }
    catch (LinterSqlException ex)
    {
      Console.WriteLine(
        "Исключение ядра СУБД ЛИНТЕР \n" +
        "Текст сообщения: " + ex.Message + "\n" +
        "Код СУБД ЛИНТЕР: " + ex.Number + "\n" +
        "Код операционной системы: " + ex.LinterSysErrorCode +
 "\n");
    }
    catch (Exception ex)
    {
      Console.WriteLine(
        "Исключение ADO.NET провайдера \n" +
        "Тип ошибки: " + ex.GetType() + "\n" +
        "Сообщение: " + ex.Message + "\n");
    }
    finally
    {
      Console.WriteLine("Освобождение ресурсов.");
      if (conn != null)
      {
        conn.Close();
      }
      Console.WriteLine("Выполнение команды завершено.");
    }
  }
}
```

GetName

Метод предоставляет имя указанного поля текущей строки выборки данных.

Синтаксис

```
public abstract string GetName(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Возвращаемое значение

Имя (значение типа System.String) указанного поля текущей строки выборки данных.

Для неименованных полей возвращается пустая строка.

Для полей с алиасным именем возвращается имя алиаса.

Исключения

IndexOutOfRangeException	Задан порядковый номер несуществующего столбца.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetName
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText =
      "Select model, 100, sysdate as \"Текущая дата\" from auto
 limit 2";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    string fieldname0 = reader.GetName(0);
    string fieldname1 = reader.GetName(1);
    string fieldname2 = reader.GetName(2);
    Console.WriteLine("| {0} | {1} | {2} |",
      fieldname0, fieldname1, fieldname2);
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
```

```
  }
}

Результат выполнения примера:

| MODEL |   | Текущая дата |
```

GetOrdinal

Метод предоставляет порядковый номер указанного именованного поля текущей строки выборки данных. Поиск именованного поля выполняется сначала с учетом регистра, в случае неудачи производится повторный поиск уже без учета регистра.

Синтаксис

```
public abstract int GetOrdinal(string name);
```

`name` – имя (в том числе и алиасное) поля в текущей строке выборки данных.

Возвращаемое значение

Порядковый номер поля с указанным именем (значение типа System.Int32) текущей строки выборки данных (отсчет начинается с `0`).

Для неименованных полей узнать их порядковый номер нельзя.

Для полей с одинаковыми именами, но разными владельцами (например, auto.personid, person.personid) в одной выборке возвращается порядковый номер первого найденного поля.

Для полей с одинаковыми именами типа select 100 as aaa, 200 as aaa возвращается порядковый номер первого найденного поля.

Исключения

`IndexOutOfRangeException`	Поле с таким именем не существует.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetOrdinal
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
```

```
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText =
      "Select model, sysdate as \"Текущая дата\" from auto limit
 2";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    int fieldOrdinal0 = reader.GetOrdinal("model");
    int fieldOrdinal1 = reader.GetOrdinal("Текущая дата");
    Console.WriteLine("| {0} | {1} |", fieldOrdinal0,
 fieldOrdinal1);
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

| 0 | 1 |
```

GetProviderSpecificFieldType

Метод предоставляет тип данных указанного поля текущей строки выборки данных в терминах СУБД ЛИНТЕР.

Примечание
В текущей версии ADO.NET-провайдера данный метод не отличается от метода `GetFieldType`.

Синтаксис

```
public virtual Type GetProviderSpecificFieldType(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Тип данных поля (значение типа System.Type) указанного поля текущей строки выборки данных в терминах СУБД ЛИНТЕР (см. метод GetFieldType).

Исключения

`IndexOutOfRangeException`	Задан порядковый номер несуществующего поля.

Пример

Пример аналогичен примеру в методе GetFieldType.

GetProviderSpecificValue

Метод предоставляет значение указанного поля текущей строки выборки данных в виде экземпляра класса `Object`.

Для определения конкретного типа возвращенного объекта необходимо использовать метод `GetProviderSpecificFieldType`.

Примечание
В текущей версии ADO.NET-провайдера данный метод не отличается от метода `GetValue`.

Синтаксис

```
public virtual Object GetProviderSpecificValue(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Значение указанного поля в виде `System.Object`.

Исключения

`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод Read.

Пример

Пример аналогичен примеру в методе GetValue.

GetProviderSpecificValues

Метод предоставляет массив значений всех полей текущей строки выборки данных.

Примечание
В текущей версии ADO.NET провайдера данный метод не отличается от метода `GetValues`.

Синтаксис

```
public virtual int GetProviderSpecificValues(Object[] values);
```

`values` – массив элементов типа Object, в котором должны быть представлены значения полей.

Возвращаемое значение

Количество элементов в массиве.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Исключения

`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод Read()).

Пример

Пример аналогичен примеру в методе GetValue.

GetSchemaTable

Метод предоставляет метаданные текущей выборки данных.

Синтаксис

```
public abstract DataTable GetSchemaTable();
```

Возвращаемое значение

Объект типа `DataTable`, содержащий метаданные текущей выборки данных.

Структура записей возвращаемого объекта DataTable:

Имя столбца в объекте DataTable	Значение столбца
`ColumnName`	Имя столбца (в том числе, и алиасное).
`ColumnOrdinal`	Порядковый номер столбца (отсчет начинается с `0`).
`ColumnSize`	Размер (в байтах) столбца, для строковых типов размер вычисляется в символах.
`NumericPrecision`	Точность представления данных (для вещественных значений и чисел с фиксированной точкой).
`ProviderType`	Индикатор типа данных столбца (значение ELinterDbType, приведенное к типу int).
`BaseSchemaName`	Имя владельца.
`DataType`	Тип данных поля (в нотации .NET).
`IsExpression`	Тип значения поля: true – вычисляемое, false – загружаемое из БД.
`IsIdentity`	Тип значения поля: true – генерируется последовательностью, false – загружается из БД.
`IsAutoIncrement`	Атрибут поля: true – автоинкрементное, false – загружаемое из БД.
`IsLong`	Атрибут поля: true – в поле имеется большой двоичный объект (BLOB), содержащий очень длинные данные, false – в противном случае.
`NumericScale`	Масштаб представления данных (для вещественных значений и чисел с фиксированной точкой).
`IsUnique`	Атрибут уникальности поля: true – уникальное, false – неуникальное.
`IsKey`	Признак наличия у поля ключа: true – поле имеет ключ, false – поле без ключа.
`BaseCatalogName`	Имя папки в хранилище данных, содержащей столбец.

Имя столбца в объекте DataTable	Значение столбца
	Примечание В текущей версии ADO.NET-провайдера не поддерживается.
`BaseColumnName`	Имя столбца в хранилище данных.
	Примечание В текущей версии ADO.NET-провайдера не поддерживается.
`BaseTableName`	Имя таблицы, которой принадлежит столбец.
`IsHidden`	Видимость столбца в выборке: true – столбец видимый, false – столбец скрытый.
	Примечание В текущей версии ADO.NET-провайдера не поддерживается.
`BaseServerName`	Имя ЛИНТЕР-сервера.
	Примечание В текущей версии ADO.NET-провайдера не поддерживается.
`AllowDBNull`	Допустимость null-значений: true – столбец допускает null-значения, false – null-значения не допускаются.
`IsAliased`	Тип имени столбца: true – алиасное, false – из схемы таблицы.
	Примечание В текущей версии ADO.NET-провайдера не поддерживается.
`IsRowVersion`	Атрибут поля: true – поле содержит постоянный идентификатор строки, в который не может быть записано значение, false – в противном случае.
	Примечание В текущей версии ADO.NET-провайдера не поддерживается.
`IsReadOnly`	Тип доступа к значениям столбца: true – только для чтения, false – полный доступ.
	Примечание В текущей версии ADO.NET-провайдера не поддерживается.

Примечание
Если нет возможности определить значения атрибутов IsUnique, IsKey, AllowDBNull, то значения этих атрибутов будут равны DBNull.Value.

Исключения

`Exception`	В БД не найдено представление `DOTNET_COLUMNS` (необходимо выполнить скрипт `catalog.sql` из подкаталога `/dict` установочного каталога СУБД ЛИНТЕР).
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetSchemaTable
{
  static void Main()
  {
    DbProviderFactory factory = null;
    DbConnection con = null;
    DbCommand cmd = null;
    DbDataReader reader = null;
    try
    {
      // Соединение с БД
      factory =
 DbProviderFactories.GetFactory("System.Data.LinterClient");
      con = factory.CreateConnection();
      con.ConnectionString = "User
 ID=SYSTEM;Password=MANAGER8;DataSource=LOCAL";
      con.Open();
      cmd = factory.CreateCommand();
      cmd.Connection = con;
      cmd.CommandText = "select MAKE, MODEL, BODYTYPE from AUTO";
      reader = cmd.ExecuteReader();
      // Получение сведений о схеме базы данных
      DataTable schema = reader.GetSchemaTable();
      // Вывод полученных сведений на экран
      OutputDataTable(schema);
    }
    catch (Exception ex)
    {
      Console.WriteLine("Ошибка: " + ex.Message);
```

```
    }
    finally
    {
      // Освобождение ресурсов
      if (reader != null)
      {
        reader.Close();
      }
      if (cmd != null)
      {
        cmd.Dispose();
      }
      if (con != null)
      {
        con.Close();
      }
    }
  }
  private static void OutputDataTable(DataTable dataTable)
  {
    int columnsNumber = Math.Min(5, dataTable.Columns.Count);
    Console.WriteLine(new String('-', 60));
    for (int i = 0; i < columnsNumber; i++)
    {
      Console.Write(dataTable.Columns[i].ColumnName + " | ");
    }
    Console.WriteLine();
    Console.WriteLine(new String('-', 60));
    foreach (DataRow row in dataTable.Rows)
    {
      for (int i = 0; i < columnsNumber; i++)
      {
        if (row.IsNull(i))
        {
          Console.Write("<NULL> | ");
        }
        else
        {
          Console.Write(row[i] + " | ");
        }
      }
      Console.WriteLine();
    }
  }
}
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
Результат выполнения примера:

-------------------------------------------------------------------------------
ColumnName | ColumnOrdinal | ColumnSize | NumericPrecision |
 ProviderType |
-------------------------------------------------------------------------------
MAKE       | 0             | 20         | 0                | 1
        |
MODEL      | 1             | 20         | 0                | 1
        |
BODYTYPE   | 2             | 15         | 0                | 1
        |
```

FastGetSchemaTable

Метод аналогичен методу GetSchemaTable.

Основные отличия:

- в способе получения метаданных: метод `FastGetSchemaTable` использует команды интерфейса нижнего уровня СУБД ЛИНТЕР (`GETA` и `FCUR`), а метод `GetSchemaTable` дополнительно использует набор SQL-запросов (т.е. привлечение sql-транслятора СУБД ЛИНТЕР), поэтому метод `FastGetSchemaTable` выполняется существенно быстрее метода `GetSchemaTable`;
- не предоставляются метаданные IsUnique, IsKey, AllowDBNull;
- метод не использует представление `DOTNET_COLUMNS` СУБД ЛИНТЕР (представление `DOTNET_COLUMNS` необходимо для получения метаданных из системных таблиц СУБД ЛИНТЕР).

GetString

Метод предоставляет символьное значение указанного поля текущей строки выборки данных.

Метод применяется для типов данных CHAR, VARCHAR, NCHAR, NVARCHAR СУБД ЛИНТЕР.

Синтаксис

```
public abstract string GetString(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Значение указанного поля в виде System.String.

Исключения

`InvalidCastException`	Невозможно преобразовать тип данных указанного столбца к типу данных string.

IndexOutOfRangeException	Задан порядковый номер несуществующего столбца.
InvalidOperationException	Не установлена текущая строка выборки данных (необходимо выполнить метод Read()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetString
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText =
      "Select model, make from auto limit 2";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    string value_model;
    string value_make;
    while (reader.Read())
    {
      value_model = reader.GetString(0);
      value_make = reader.GetString(1);
      Console.WriteLine("Автомобиль " + value_model +
        " выпущен фирмой " + value_make);
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
Результат выполнения примера:

Автомобиль MERCURY COMET GT V8  выпущен фирмой FORD
Автомобиль A-310                выпущен фирмой ALPINE
```

GetValue

Метод предоставляет значение указанного поля текущей строки выборки данных в виде .NET-объекта.

Примечание
Если в БД в поле DATE хранится значение 00.00.0000:00:00:00, то метод GetValue возвращает объект DateTime, который соответствует дате 01.01.1900:00:00:00.

Синтаксис

```
public abstract Object GetValue(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Значение указанного поля в виде System.Object.

Для столбцов БД со значениями NULL возвращается объект `DBNull`.

Исключения

`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetValue
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
```

```
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText =
      "select model, personid from auto limit 2";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    while (reader.Read())
    {
      Console.WriteLine("Автомобиль марки " + reader.GetValue(0) +
        " принадлежит владельцу " + reader.GetValue(1));
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

Автомобиль марки MERCURY COMET GT V8  принадлежит владельцу 1
Автомобиль марки A-310                принадлежит владельцу 2
```

GetValues

Метод предоставляет массив значений полей текущей строки выборки данных в виде .NET-объектов.

Размерность массива для заполнения значениями может не совпадать с количеством полей текущей строки выборки данных. В этом случае заполнение массива выполняется либо в соответствии с его фактическим размером (если размер массива меньше или равен количеству полей в строке), и null-значениями в остальных элементах массива.

Для столбца БД со значениями NULL возвращается объект `DBNull`.

Синтаксис

```
public abstract int GetValues(Object[] values);
```

`values` – массив элементов типа Object, в котором должны быть представлены значения полей выборки.

Возвращаемое значение

Количество элементов в массиве.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Исключения

`ArgumentNullException`	Параметр values имеет значение null.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetValues
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "select make, model from auto limit 2";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    while (reader.Read())
    {
      object[] values = new object[reader.FieldCount];
      reader.GetValues(values);
      for (int i = 0; i < values.Length; i++)
      {
        Console.Write(values[i] + " | ");
      }
      Console.WriteLine();
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
```

```
  }
}
```

```
Результат выполнения примера:

FORD                  | MERCURY COMET GT V8   |
ALPINE                | A-310                 |
```

IsDBNull

Метод проверяет указанное поле текущей строки выборки данных на null-значение.

Примечание
Метод используется для исключения ошибки при запросе значений (с помощью метода `GetByte`, `GetInt32` и т.п.) полей выборки данных.

Синтаксис

```
public abstract bool IsDBNull(int ordinal);
```

`ordinal` – порядковый номер поля в текущей строке выборки данных (отсчет начинается с `0`).

Возвращаемое значение

Значение типа System.Bolean:

- true – проверяемое значение является null-значением;
- false – в противном случае.

Исключения

`IndexOutOfRangeException`	Задан порядковый номер несуществующего столбца.
`InvalidOperationException`	Не установлена текущая строка выборки данных (необходимо выполнить метод `Read`()).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class IsDBNull
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
```

```
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "Select sysdate, null";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    object value;
    while (reader.Read())
    {
      for (int i = 0; i < reader.FieldCount; i++)
      {
        if (!reader.IsDBNull(i))
        {
          value = reader.GetValue(i);
        }
        else
        {
          value = "Значение не определено";
        }
        Console.Write(value + " | ");
      }
      Console.WriteLine();
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}

Результат выполнения примера:

06.09.2012 13:28:23 | Значение не определено |
```

NextResult

Метод выполняет переход к следующей выборке данных (в случае пакетного выполнения SQL-запросов).

Синтаксис

```
public abstract bool NextResult();
```

Возвращаемое значение

Значение типа System.Boolean:

- true – выполнен переход к следующей выборке данных;
- false – в противном случае.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class NextResult
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "select make, model from auto limit 2;" +
      "select firstnam from person limit 2";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    while (reader.HasRows)
    {
      while (reader.Read())
      {
        for (int i = 0; i < reader.FieldCount; i++)
        {
          Console.Write(reader.GetValue(i) + " | ");
        }
        Console.WriteLine();
      }
```

```
      reader.NextResult();
      Console.WriteLine();
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

```
Результат выполнения примера:

FORD                | MERCURY COMET GT V8|
ALPINE              | A-310              |

PHIL            |
JOHN            |
```

Read

Метод предоставляет доступ к полям текущей строки выборки данных и перемещает указатель текущей строки выборки данных на следующую строку (если не достигнут конец выборки).

По умолчанию при открытии выборки данных указатель текущей строки выборки находится **перед первой** строкой выборки данных, поэтому необходимо вызвать метод `Read()`, чтобы начать получать доступ к данным.

Синтаксис

```
public abstract bool Read();
```

Возвращаемое значение

Значение типа System.Boolean:

- true – выполнен переход к следующей строке выборки данных;
- false – в противном случае.

Исключения

`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class Read
```

```
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText = "select model, make from auto limit 2";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    while (reader.Read())
    {
      Console.WriteLine("Марка авто: {0} Производитель: {1}",
        reader[0], reader[1]);
    }
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

```
Результат выполнения примера:

Марка авто: MERCURY COMET GT V8  Производитель: FORD
Марка авто: A-310                Производитель: ALPINE
```

Класс DbTransaction

Класс `DbTransaction` является базовым классом для управления транзакциями. В ADO.NET-провайдере СУБД ЛИНТЕР для последовательности SQL-запросов транзакционный режим задается с помощью метода `BeginTransaction` класса `DbConnection`.

Классы `DbCommand` и `DbTransaction` можно связать следующим способом: прописать в свойство Transaction класса `DbCommand` фактическое значение созданной транзакции.

Свойства класса `DbTransaction` приведены в таблице 15.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Таблица 15. Свойства класса DbTransaction

Свойство	Описание
Connection	Предоставляет информацию о соединении с ЛИНТЕР-сервером, связанном с данной транзакцией.
IsolationLevel	Предоставляет информацию об установленном транзакционном режиме в соединении с севером источника данных.

Методы класса приведены в таблице 16.

Таблица 16. Методы класса DbTransaction

Метод	Описание
Commit	Подтверждает текущую транзакцию и завершает её.
Commit(String)	подтверждает текущую транзакцию до указанной точки сохранения.
Rollback	Отменяет текущую транзакцию.
Rollback(String)	Отменяет текущую транзакцию до указанной точки сохранения.
Save(String)	Устанавливает точку сохранения в текущей транзакции.

Свойства

Connection

Предоставляет информацию о соединении с ЛИНТЕР-сервером, связанном с данной транзакцией.

Клиентское приложение может иметь несколько соединений с разными ЛИНТЕР-серверами, в которых с помощью разных объектов `DbTransaction` могут задаваться индивидуальные транзакционные режимы. Данное свойство позволяет определить по объекту `DbTransaction` соответствующее ему соединение.

Декларация

```
public DbConnection Connection {get;};
```

Значение свойства

Объект `DbConnection`, представляющий соединение, связанное с данной транзакцией.

Исключения

Отсутствуют.

Пример

```
DbConnection conn = tran.Connection;
```

IsolationLevel

Предоставляет информацию об установленном транзакционном режиме в соединении с ЛИНТЕР-сервером.

Значением по умолчанию является ReadCommitted.

Список возможных значений:

Значение свойства		Соответствие уровню изоляций СУБД ЛИНТЕР
`Unspecified`		Pessimistic.
`Chaos`		Не поддерживается.
`ReadUncommitted`		Не поддерживается.
`ReadCommitted`		Pessimistic.
`RepeatableRead`		Не поддерживается.
`Serializable`		Не поддерживается.
`Snapshot`		Optimistic. **Примечания** 1. Режим Snapshot можно использовать для задания режима Optimistic только в приложениях, разработанных специально для СУБД ЛИНТЕР. 2. Режим Snapshot (Optimistic) устарел. Применять не рекомендуется.

Декларация

```
public abstract IsolationLevel IsolationLevel {get;};
```

Значение свойства

Объект `IsolationLevel`, который определяет уровень изоляции транзакций.

Исключения

Отсутствуют.

Пример

```
IsolationLevel iso = tran.IsolationLevel;
```

Методы

Commit

Метод подтверждает текущую транзакцию и завершает её. Используемые ресурсы не освобождаются.

Синтаксис

```
public abstract void Commit();
```

Возвращаемое значение

Значение типа void.

Исключения

`InvalidOperationException`	Транзакция уже завершена (подтверждена/отменена/ соединение закрыто).
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommitSample
{
  static void Main()
  {
    // В примере инициируется транзакция и в таблицу auto
 добавляются две записи с
    // одинаковым значением столбца personid
    // Если столбец personid не является первичным ключом, запись
 с дубликатом
    // значения добавляется и транзакция завершается успешно.
    // Если personid является первичным ключом, то вторая запись
 нарушает
    //  целостность БД и транзакция отменяется
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "Data Source=LOCAL;User
 ID=SYSTEM;Password=MANAGER8";
    con.Open();
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Проверяем кол-во строк в таблице auto  перед началом
 транзакции
    cmd.CommandText = "SELECT COUNT(*) FROM auto";
    int AutoCount = int.Parse(cmd.ExecuteScalar().ToString());
    // отображаем кол-во строк в таблице auto
    Console.WriteLine("Кол-во строк в таблице AUTO = " +
 AutoCount);
    // Начинаем транзакцию
```

```
    DbTransaction txn =
 con.BeginTransaction(IsolationLevel.ReadCommitted);
    cmd.Transaction = txn;
    try
    {
      // Добавляем дважды одну и ту же строку
      cmd.CommandText = "INSERT INTO auto(personid) VALUES
 (2000)";
      cmd.ExecuteNonQuery();
      cmd.ExecuteNonQuery(); // При добавлении дубликата  возможно
 исключение
      txn.Commit();
    }
    catch (Exception e)
    {
      // Печать диагностического сообщения
      Console.WriteLine("Ошибка добавления записи = " +
 e.Message);
      // Отмена транзакции
      txn.Rollback();
    }
    // Проверяем кол-во строк в таблице auto после завершения
 транзакции
    cmd.CommandText = "SELECT COUNT(*) FROM auto";
    AutoCount = int.Parse(cmd.ExecuteScalar().ToString());
    // Отображаем полученное кол-во строк
    // Если столбец personid  является первичным ключом,  кол-во
 строк  не должно
    // измениться, в противном случае должно увеличиться на 2
    Console.WriteLine("Кол-во строк в таблице AUTO  = " +
 AutoCount);
    txn.Dispose();
    cmd.Dispose();
    con.Close();
    con.Dispose();
  }
}
Результат выполнения примера:
Кол-во строк в таблице AUTO = 1001
Ошибка добавления записи = [Linter error] duplicate value for
 primary or unique key
Кол-во строк в таблице AUTO  = 1001
```

Commit(String)

Метод подтверждает текущую транзакцию до указанной точки сохранения. Все установленные точки сохранения до указанной точки удаляются, последующие – сохраняются. Выполнение транзакции продолжается.

Примечание
Данный метод может использоваться в клиентских приложениях, ориентированных исключительно на работу с СУБД ЛИНТЕР.

Синтаксис

```
public  void Commit(string savePointName);
```

`savePointName` – имя точки сохранения.

Возвращаемое значение

Значение типа void.

Исключения

`InvalidOperationException`	Транзакция уже завершена (подтверждена/отменена/ соединение закрыто).
`ArgumentNullException`	Параметр savePointName имеет null-значение.
`ArgumentException`	Параметр savePointName равен пустой строке или ссылается на несуществующую точку сохранения.
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class CommitSample
{
  static void Main()
  {
    // Соединение с БД
    LinterDbConnection con = new LinterDbConnection();
    con.ConnectionString = "Data Source=LOCAL;User
 ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта LinterDbCommand, связанного с
 установленным соединением
    LinterDbCommand cmd = con.CreateCommand();
    // Создаем тестовую таблицу
```

```
    cmd.CommandText = "create or replace table test (abc
 varchar(255))";
    cmd.ExecuteNonQuery();
    // Начинаем транзакцию в режиме Exclusive
    LinterDbTransaction txn =
 con.BeginTransaction(IsolationLevel.ReadCommitted);
    cmd.Transaction = txn;
    // Делаем первый insert в тестовую таблицу
    cmd.CommandText = "insert into test (abc) values ('запись
 1')";
    cmd.ExecuteNonQuery();
    // Создаем точку сохранения SP1
    txn.Save("SP1");
    // Делаем второй insert в тестовую таблицу
    cmd.CommandText = "insert into test (abc) values ('запись
 2')";
    cmd.ExecuteNonQuery();
    // Создаем точку сохранения SP2
    txn.Save("SP2");
    // Делаем третий insert в тестовую таблицу
    cmd.CommandText = "insert into test (abc) values ('запись
 3')";
    cmd.ExecuteNonQuery();
    // Выполняем метод  Commit (SP1)
    txn.Commit("SP1");
    // Выполняем метод RollBack() для всей транзакции
    txn.Rollback();
    // Читаем записи из test
    cmd.CommandText = "select abc from test";
    LinterDbDataReader reader = cmd.ExecuteReader();
    // Должна быть только одна первая запись
    while (reader.Read())
    {
      Console.WriteLine(reader.GetValue(0));
    }
    txn.Dispose();
    cmd.Dispose();
    con.Close();
    con.Dispose();
  }
}
Результат выполнения примера:
запись 1
```

Rollback

Метод отменяет текущую транзакцию. Используемые ресурсы не освобождаются.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Синтаксис

```
public abstract void Rollback();
```

Возвращаемое значение

Значение типа void.

Исключения

`InvalidOperationException`	Транзакция уже завершена (подтверждена/отменена/ соединение закрыто).
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

См. пример в методе Commit.

Rollback(String)

Метод отменяет текущую транзакцию до указанной точки сохранения.

Установленные точки сохранения до указанной точки сохраняются, последующие – удаляются. Выполнение транзакции продолжается.

Примечание
Данный метод может использоваться в клиентских приложениях, ориентированных исключительно на работу с СУБД ЛИНТЕР.

Синтаксис

```
public  void RollBack(string savePointName);
```

`savePointName` – имя точки сохранения.

Возвращаемое значение

Значение типа void.

Исключения

`InvalidOperationException`	Транзакция уже завершена (подтверждена/отменена/ соединение закрыто).
`ArgumentNullException`	Параметр savePointName имеет значение null.
`ArgumentException`	Параметр savePointName равен пустой строке или ссылается на несуществующую точку сохранения.
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен 0.

Пример

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class RollbackSample
```

```
{
  static void Main()
  {
    // Соединение с БД
    LinterDbConnection con = new LinterDbConnection();
    con.ConnectionString = "Data Source=LOCAL;User
 ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта LinterDbCommand, связанного с
 установленным соединением
    LinterDbCommand cmd = con.CreateCommand();
    // Создаем тестовую таблицу
    cmd.CommandText = "create or replace table test (abc
 varchar(255))";
    cmd.ExecuteNonQuery();
    // Начинаем транзакцию в режиме Exclusive
    LinterDbTransaction txn =
 con.BeginTransaction(IsolationLevel.ReadCommitted);
    cmd.Transaction = txn;
    // Делаем первый insert в тестовую таблицу
    cmd.CommandText = "insert into test (abc) values ('запись
 1')";
    cmd.ExecuteNonQuery();
    // Создаем точку сохранения SP1
    txn.Save("SP1");
    // Делаем второй insert в тестовую таблицу
    cmd.CommandText = "insert into test (abc) values ('запись
 2')";
    cmd.ExecuteNonQuery();
    // Создаем точку сохранения SP2
    txn.Save("SP2");
    // Делаем третий insert в тестовую таблицу
    cmd.CommandText = "insert into test (abc) values ('запись
 3')";
    cmd.ExecuteNonQuery();
    // Создаем точку сохранения SP3
    txn.Save("SP3");
    // Выполняем метод  Rollback(SP2)
    txn.Rollback("SP2");
    // Выполняем метод Commit() для всей транзакции
    txn.Commit();
    // Читаем записи из test
    cmd.CommandText = "select abc from test";
    LinterDbDataReader reader = cmd.ExecuteReader();
    // Должны быть только две первых записи
    while (reader.Read())
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    {
      Console.WriteLine(reader.GetValue(0));
    }
    txn.Dispose();
    cmd.Dispose();
    con.Close();
    con.Dispose();
  }
}
Результат выполнения примера:
запись 1
запись 2
```

Save(String)

Метод устанавливает точку сохранения в текущей транзакции.

Примечание
Данный метод может использоваться в клиентских приложениях, ориентированных исключительно на работу с СУБД ЛИНТЕР.

Синтаксис

```
public void Save(String savePointName);
```

`savePointName` – имя точки сохранения.

Возвращаемое значение

`Значение типа void.`

Исключения

`InvalidOperationException`	Транзакция уже завершена (подтверждена/отменена/ соединение закрыто).
`ArgumentNullException`	Параметр savePointName имеет значение null.
`ArgumentException`	Параметр savePointName равен пустой строке или является дубликатом существующей точки сохранения.
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

См. пример в методе Commit(String).

Класс DbParameter

Класс `DbParameter` управляет атрибутами одного отдельно взятого параметра. Объединение всех подготовленных параметров для последующего

присоединения их к параметризованному запросу выполняется с помощью класса `DbParameterCollection`.

Конструкторы класса `DbParameter` приведены в таблице 17.

Таблица 17. Конструкторы класса DbParameter

Конструктор	*Описание*
LinterDbParameter()	Создает новый объект `DbParameter` для неопределенного параметра.
LinterDbParameter(String, Object)	Создает новый объект `DbParameter` для именованного параметра.
LinterDbParameter(String, ELinterDbType)	Создает новый объект `DbParameter` для именованного параметра указанного типа.
LinterDbParameter(String, ELinterDbType, Int)	Создает новый объект `DbParameter` для именованного параметра указанного типа заданной длины.
LinterDbParameter(String, ELinterDbType, Int, String)	Создает новый объект `DbParameter` для именованного параметра указанного типа с заданными длиной и свойством SourceColumn.
LinterDbParameter(String, ELinterDbType, Int, ParameterDirection, Bool, Byte, Byte, String, DataRowVersion, Object)	Создает новый объект `DbParameter` с указанием всех атрибутов параметра.

Свойства класса `DbParameter` приведены в таблице 18.

Таблица 18. Свойства класса DbParameter

Свойство	**Описание**
DbType	Предоставляет/устанавливает тип параметра в .NET-терминологии.
Direction	Предоставляет/устанавливает вид параметра (входной, выходной, смешанный, процедурный).
IsNullable	Индикатор возможности присваивать null-значение.
ParameterName	Предоставляет/устанавливает имя параметра.
Size	Предоставляет/устанавливает максимальную длину значения параметра.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Свойство	Описание
SourceColumn	Предоставляет/устанавливает имя столбца набора данных DataSet, к которому привязан параметр.
SourceColumnNullMapping	Предоставляет/устанавливает признак допустимости null-значений столбца в наборе данных DataSet.
SourceVersion	Предоставляет/устанавливает версию столбца, используемую при загрузке свойства Value (текущая или оригинальная версия) в наборе данных DataSet, к которому привязан параметр.
Value	Предоставляет/устанавливает значение параметра.
LinterDbType	Предоставляет/устанавливает тип параметра в терминах СУБД ЛИНТЕР.
Precision	Предоставляет/устанавливает точность представления данных для вещественных типов данных и чисел с фиксированной точкой.
Scale	Предоставляет/устанавливает масштаб представления данных для вещественных типов данных и чисел с фиксированной точкой.

Методы класса `DbParameter` приведены в таблице 19.

Таблица 19. Методы класса DbParameter

Метод	Описание
ResetDbType	Сбрасывает свойство DbType к его исходному значению.

Конструкторы

ADO.NET-провайдер СУБД ЛИНТЕР обеспечивает поддержку 6-ти конструкторов класса `DbParameter`.

LinterDbParameter()

Синтаксис

```
public LinterDbParameter();
```

Возвращаемое значение

Конструктор создает новый объект `DbParameter` со следующими атрибутами:

- имя параметра – пустая строка;
- тип данных параметра – string;
- вид параметра – input;
- свойство SourceVersion параметра – DataRowVersion.Current;
- свойство SourceColumn параметра – пустая строка.

LinterDbParameter(String, Object)

Синтаксис

```
public LinterDbParameter(string strName, object objValue);
```

`strName` – имя параметра.

`objValue` – значение параметра.

Возвращаемое значение

Конструктор создает новый объект `DbParameter` со следующими атрибутами:

- имя параметра – значение аргумента strName;
- свойство Value (собственно значение параметра) – значение аргумента objValue (по умолчанию null);
- тип данных параметра – string;
- вид параметра – input;
- свойство SourceVersion параметра – DataRowVersion.Current;
- свойство SourceColumn параметра – пустая строка.

LinterDbParameter(String, ELinterDbType)

Синтаксис

```
public LinterDbParameter(string strName, ELinterDbType type);
```

`strName` – имя параметра.

`type` – тип данных параметра.

Возвращаемое значение

Конструктор создает новый объект `DbParameter` со следующими атрибутами:

- имя параметра – значение аргумента strName;
- свойство LinterDbType параметра – значение аргумента type;
- тип данных параметра – устанавливается в соответствии с аргументом type;
- вид параметра – input;
- свойство SourceVersion параметра – DataRowVersion.Current.

LinterDbParameter(String, ELinterDbType, Int)

Синтаксис

```
public LinterDbParameter(string strName, ELinterDbType type,  int
 iSize);
```

`strName` – имя параметра.

`type` – тип данных параметра.

`iSize` – длина значения параметра.

Возвращаемое значение

Конструктор создает новый объект `DbParameter` со следующими атрибутами:

- имя параметра – значение аргумента strName;

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

- свойство LinterDbType параметра – значение аргумента type;
- тип данных параметра – устанавливается в соответствии с аргументом type;
- вид параметра – input;
- свойство SourceVersion параметра – DataRowVersion.Current;
- свойство SourceColumn параметра – пустая строка;
- буфер указанного в аргументе iSize размера для хранения значения параметра указанного типа.

Примечание
Данный конструктор используется обычно при работе с параметрами с переменной длиной (например, CHAR[20]).

LinterDbParameter(String, ELinterDbType, Int, String)

Синтаксис

```
public LinterDbParameter(string strName, ELinterDbType type,  int
 iSize, string strSourceColumn);
```

`strName` – имя параметра.

`type` – тип данных параметра.

`iSize` – длина значения параметра.

`strSourceColumn` – значение свойства параметра.

Возвращаемое значение

Конструктор создает новый объект `DbParameter` со следующими атрибутами:

- имя параметра – значение аргумента strName;
- свойство LinterDbType параметра – значение аргумента type;
- тип данных параметра – устанавливается в соответствии с аргументом type;
- вид параметра – input;
- свойство SourceVersion параметра – DataRowVersion.Current.
- буфер указанного в аргументе iSize размера для хранения значения параметра указанного типа;
- свойство SourceColumn параметра – значение аргумента strSourceColumn (по умолчанию пустая строка).

LinterDbParameter(String, ELinterDbType, Int, ParameterDirection, Bool, Byte, Byte, String, DataRowVersion, Object)

Синтаксис

```
public LinterDbParameter(string strName, ELinterDbType type,
 int iSize, ParameterDirection direction, bool nullable, byte
```

```
bPrecision, byte bScale, string strSourceColumn, DataRowVersion
sourceVer, object objValue);
```

`strName` – имя параметра.

`type` – тип данных параметра.

`iSize` – длина значения параметра.

`direction` – тип параметра.

`nullable` – признак допустимости null-значений.

`bPrecision` – точность значений параметра.

`bScale` – масштаб значений параметра.

`strSourceColumn` – значение свойства параметра.

`sourceVer` – версия параметра.

`objValue` – значение параметра.

Возвращаемое значение

Конструктор создает новый объект `DbParameter` со следующими атрибутами:

- имя параметра – значение аргумента strName;
- свойство LinterDbType параметра – значение аргумента type;
- тип данных параметра – устанавливается в соответствии с аргументом type;
- вид параметра – значение аргумента direction;
- буфер указанного в аргументе iSize размера для хранения значения параметра указанного типа;
- свойство IsNullable параметра – значение аргумента nullable (по умолчанию false);
- свойство Precision (точность) параметра – значение аргумента bPrecision (по умолчанию 0);
- свойство Scale (масштаб) параметра – значение аргумента bScale (по умолчанию 0);
- свойство SourceColumn параметра – значение аргумента strSourceColumn (по умолчанию пустая строка);
- свойство SourceVersion параметра – DataRowVersion.Current;
- свойство Value (собственно значение параметра) – значение аргумента objValue (по умолчанию null).

Свойства

DbType

Свойство предоставляет или устанавливает тип параметра в .NET-терминологии.

Значение по умолчанию – String.

Декларация

```
[BrowsableAttribute(false)] public abstract DbType DbType {get;
 set;};
```

Типы параметров:

Тип параметра	Описание
AnsiString	Символьные строки в кодировке ANSI переменной длины от 1 до 8000 символов.
Binary	Двоичные данные длиной от 1 до 8000 байт.
Byte	8-битовое целое число без знака, которое может принимать значения от 0 до 255.
Boolean	Простой тип для представления логических значений true и false.
Currency	Значение типа currency, лежащее в диапазоне от -2^{63} (или -922,337,203,685,477.5808) до 2^{63}-1 (или +922,337,203,685,477.5807) и имеющее точность до одной десятитысячной денежной единицы (таблица 20).
Date	Тип для представления значений даты.
DateTime	Тип для представления значений даты и времени (таблица 20).
Decimal	Простой тип для представления значений, лежащих в диапазоне от 1,0 x 10^-28 до приблизительно 7,9 x 10^28 с 28-29 значимыми цифрами.
Double	Простой тип для представления значений с плавающей запятой, лежащих в диапазоне от 5,0 x 10^-324 до приблизительно 1,7 x 10^308 с точностью до 15-16 знаков.
Guid	Глобальный уникальный идентификатор (GUID).
Int16	Целочисленный тип для представления 16-разрядных целых чисел со знаком, лежащих в диапазоне от -32767 до 32767.
Int32	Целочисленный тип для представления 32-разрядных целых чисел со знаком, лежащих в диапазоне от -2147483647 до 2147483647.
Int64	Целочисленный тип для представления 64-разрядных целых чисел со знаком, лежащих в диапазоне от -9223372036854775807 до 9223372036854775807.
Object	Общий тип для представления всех значений и ссылок, которые не могут быть представлены ни одним другим значением DbType.
SByte	Целочисленный тип для представления 8-разрядных целых чисел со знаком, лежащих в диапазоне от -128 до 127 (таблица 20).
Single	Простой тип для представления значений с плавающей запятой, лежащих в диапазоне от 1,5 x 10^-45 до 3,4 x 10^38 с точностью до 15-16 знаков.

Тип параметра	Описание
String	Тип для представления символьных строк UNICODE.
Time	Тип для представления значений времени (таблица 20)
UInt16	Целочисленный тип для представления 16-разрядных целых чисел без знака, лежащих в диапазоне от 0 до 65535 (таблица 20).
UInt32	Целочисленный тип для представления 32-разрядных целых чисел без знака, лежащих в диапазоне от 0 до 4294967295 (таблица 20).
UInt64	Целочисленный тип для представления 64-разрядных целых чисел без знака, лежащих в диапазоне от 0 до 18446744073709551615 (таблица 20).
VarNumeric	Числовое значение переменной длины (таблица 20).
AnsiStringFixedLength	Символьные строки в кодировке ANSI фиксированной длины.
StringFixedLength	Строка фиксированной длины из символов UNICODE.
Xml	Проанализированное представление фрагмента или документа XML (таблица 20).
DateTime2	Данные даты и времени. Значение даты может находиться в диапазоне от 1 января 1 г. н.э. до 31 декабря 9999 года н. э. Значение времени может находиться в диапазоне от 00:00:00 до 23:59:59.9999999 с точностью до 100 наносекунд (таблица 20).
DateTimeOffset	Тип даты и времени, поддерживающий часовые пояса. Значение даты может находиться в диапазоне от 1 января 1 г. н.э. до 31 декабря 9999 года н.э. Значение времени может находиться в диапазоне от 00:00:00 до 23:59:59.9999999 с точностью до 100 наносекунд. Часовые пояса могут находиться в диапазоне от -14:00 до +14:00 (таблица 20).

Таблица 20. Соответствие типов DbType типам данных СУБД ЛИНТЕР

Тип данных DbType	Тип данных СУБД ЛИНТЕР
AnsiString	NCHAR
Binary	BYTE
Byte	BYTE(1)
Boolean	BOOLEAN
Currency	DECIMAL
Date	NCHAR(44)
DateTime	NCHAR(44)
Decimal	DECIMAL
Double	DOUBLE

Тип данных DbType	Тип данных СУБД ЛИНТЕР
Guid	BYTE(16)
Int16	SMALLINT
Int32	INTEGER
Int64	BIGINT
Object	BLOB
SByte	SMALLINT
Single	REAL
String	NCHAR
Time	NCHAR(44)
UInt16	INTEGER
UInt32	BIGINT
UInt64	DECIMAL
VarNumeric	DECIMAL
AnsiStringFixedLength	NCHAR
StringFixedLength	NCHAR
Xml	В текущей версии ADO.NET провайдера не поддерживается
DateTime2	В текущей версии ADO.NET провайдера не поддерживается
DateTimeOffset	В текущей версии ADO.NET провайдера не поддерживается

Значение свойства

Тип параметра.

Исключения

ArgumentException	Неизвестный тип параметра.

Примеры

1) Получение свойства.

```
DbType dbType = parameter.DbType;
```

2) Установка свойства.

```
parameter.DbType = DbType.Int16;
```

Direction

Свойство предоставляет или устанавливает вид параметра:

- входной (Input) (значение по умолчанию). Значение параметра используется при выполнении запроса. Измененное запросом входное значение этого вида параметра не возвращается;
- выходной (Output). Параметр используется для размещения результата выполнения запроса. Входное значение игнорируется;

- смешанный (InputOutput). Входное значение используется при выполнении запроса, после чего заменяется результатом выполнения запроса;
- процедурный (ReturnValue). Содержит результат выполнения хранимой процедуры.

Если вид параметра Output, а запрос, связанный с DbCommand, не возвращает значение, то данное свойство параметра будет иметь null-значение.

Параметры вида Output, InputOutput и ReturnValue, возвращаемые методом `ExecuteReader`, становятся доступными только после вызова метода `Close` или `Dispose` в объекте `DbDataReader`.

Декларация

```
public abstract ParameterDirection Direction {get; set;};
```

Значение свойства

Вид параметра.

Исключения

`ArgumentException`	Неизвестный вид параметра.

Примеры

1) Получение свойства.

```
ParameterDirection dir = parameter.Direction;
```

2) Установка свойства.

```
parameter.Direction = ParameterDirection.InputOutput;
```

IsNullable

Свойство возвращает или устанавливает значение, показывающее, может ли параметр принимать null-значения:

- true – null-значения допускаются;
- false – null-значения запрещены.

Свойство доступно только для чтения и устанавливается ADO.NET-провайдером автоматически при привязке параметра к какому-либо столбцу выборки данных (т.е. для этого столбца проверяется ограничение NOT NULL).

Примечание
В текущей версии ADO.NET-провайдера автоматическая установка свойства IsNullable не поддерживается.

Работа с null-значениями выполнятся с помощью класса `DBNull`.

Класс `DBNull` представляет несуществующее значение. Например, в поле строки таблицы БД могут отсутствовать данные. В этом случае, поле считается несуществующим, а не просто не имеющим значения. Объект `DBNull` представляет несуществующее поле (значение).

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Тип DBNull является одноэлементным классом, т.е. допускается существование только одного объекта DBNull. Член DBNull.Value представляет единственный объект DBNull. DBNull.Value можно использовать для явного присвоения несуществующего значения полю таблицы БД, хотя большинство поставщиков данных ADO.NET обеспечивает автоматическое присвоение значений DBNull при отсутствии в поле допустимого значения. Чтобы определить, является ли значение, извлеченное из поля таблицы БД, значением DBNull, можно передать значение этого поля методу DBNull.Value.Equals. Однако в некоторых языках программирования и объектах БД предусмотрены методы, с помощью которых можно намного проще определить, содержится ли в поле таблицы БД значение DBNull.Value. К их числу относится метод DbDataReader.IsDBNull.

Не следует путать объект DBNull с понятием null в объектно-ориентированных языках программирования. В объектно-ориентированных языках программирования null-значение означает отсутствие ссылки на объект. Объект же DBNull представляет неинициализированный вариант или несуществующее значение поля таблицы БД.

Декларация

```
public bool IsNullable {get; set;};
```

Значение свойства

Объект типа bool, определяющий допустимость null-значений.

Исключения

Отсутствуют.

Примеры

1) Получение свойства.

```
bool isNullable = parameter.IsNullable;
```

2) Установка свойства.

```
parameter.IsNullable = true;
```

ParameterName

Свойство предоставляет или устанавливает имя параметра. Имя не зависит от регистра.

Примечание
В текущей версии ADO.NET-провайдера русскоязычные имена не поддерживаются.

Максимальная длина имени именованного параметра 66 символов. Имя неименованного параметра – пустая строка.

Значение по умолчанию для имени параметра – пустая строка.

В SQL-запросах именованные параметры должны задаваться в виде :<имя параметра>,

а неименованные параметры в виде знака ?,

например,

```
select * from auto where make = :brand and color=?;
```

Декларация

```
public abstract string ParameterName {get; set;};
```

Значение свойства

Объект типа string, представляющий имя параметра.

Исключения

`ArgumentException`	Недопустимая длина имени параметра.

Примеры

1) Получение свойства.

```
string paramName = parameter.ParameterName;
```

2) Установка свойства (именование параметра).

```
parameter.ParameterName = "AutoMake";
```

Size

Свойство предоставляет или устанавливает максимальную длину значения параметра в байтах. Если свойство не установлено явно, то оно наследуется из значения параметра.

Свойство Size используется для числовых и строковых типов данных.

Для типов данных переменной длины свойство Size устанавливает максимальное количество данных, передаваемых на ЛИНТЕР-сервер. Например, для UTF-16 строкового значения свойство Size может быть использовано для того, чтобы ограничить объем данных, отправляемых на ЛИНТЕР-сервер, до первых ста символов.



Примечание
Для всех параметров переменной длины необходимо явно установить ненулевое значение Size.

Для смешанных, выходных и возвращаемых процедурных параметров необходимо задавать значение свойства Size. Это не обязательно для входных параметров. Если размеры не заданы в явном виде, то они берутся из фактического размера указанного параметра при выполнении параметризованного оператора.

Хотя значения свойства DbType и Size параметра могут быть получены из свойства Value (т.е. задавать их не обязательно), но если DbType и Size не установлены явно, то автоматически вычисляемые значения этих свойств ADO.NET-провайдером не устанавливаются. Например, если на основании свойства Value был определен размер параметра, то свойство Size не будет содержать вычисленное значение после выполнения оператора.

Для типов данных с фиксированной длиной установка значения свойства Size игнорируется. Его можно извлечь лишь в информационных целях. Свойство возвращает наибольшее количество байтов, используемых ADO.NET-провайдером при передаче значения параметра на ЛИНТЕР-сервер.

Если размер значения, предоставленного для `DbParameter`, превышает указанную величину для свойства Size, то свойство Value объекта `DbParameter` будет содержать указанное значение, обрезанное до размера Size объекта `DbParameter`.

Для параметра типа DbType.String значение Size должно задаваться в UTF-16 символах (а не в байтах).

Примечание
В текущей версии ADO.NET-провайдера значение Size всегда должно задаваться в байтах.

Длина -1 устанавливается внутри ADO.NET-провайдера и указывает на то, что параметр содержит null-значение. Клиентское приложение не должно устанавливать длину -1. Вместо этого, оно должен установить Value=null или Value=DBNull.Value.

Декларация

```
public abstract int Size {get; set;};
```

Значение свойства

Объект типа int, представляющий длину параметра.

Исключения

`ArgumentException`	Недопустимая длина параметра.

Примеры

1) Получение свойства.

```
int size = parameter.Size;
```

2) Установка свойства.

```
parameter.Size = 255;
```

SourceColumn

Свойство предоставляет или устанавливает имя столбца набора данных DataSet, к которому привязан данный параметр. Используется ADO.NET-провайдером для загрузки в параметр значения из указанного столбца (для присвоения значения свойству Value параметра).

Значение по умолчанию – пустая строка.

Если значение, установленное для свойства SourceColumn, не является пустой строкой, значит, оно получено из столбца с именем SourceColumn.

Если для свойства Direction задано значение Input, то значение параметра берется из набора данных DataSet.

Если для свойства Direction задано значение Output, то значение параметра берется из БД ЛИНТЕР.

Свойство Direction для InputOutput представляет собой комбинацию обоих значений.

Декларация

```
public abstract string SourceColumn {get; set;};
```

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ParameterSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText = "create or replace table UserInfo" +
      "(UserID int, UserName Char(256), Age int)";
    cmd.ExecuteNonQuery();
    // Создание объекта DataTable
    DataTable users = new DataTable("Пользователи");
    users.Columns.Add("Номер", typeof(int));
    users.Columns.Add("Имя пользователя", typeof(string));
    users.Columns.Add("Возраст", typeof(int));
    // Создание параметра для колонки "Номер"
    DbParameter paramUserID = factory.CreateParameter();
    paramUserID.SourceColumn = "Номер";
    paramUserID.ParameterName = ":pUserID";
    paramUserID.DbType = DbType.Int32;
    // Создание параметра для колонки "Имя пользователя"
    DbParameter paramUserName = factory.CreateParameter();
    paramUserName.SourceColumn = "Имя пользователя";
    paramUserName.ParameterName = ":pUserName";
    paramUserName.DbType = DbType.String;
    // Создание параметра для колонки "Возраст"
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    DbParameter paramAge = factory.CreateParameter();
    paramAge.SourceColumn = "Возраст";
    paramAge.ParameterName = ":pAge";
    paramAge.DbType = DbType.Int32;
    // Создание команды для выборки данных из таблицы
    DbCommand selectCommand = factory.CreateCommand();
    selectCommand.CommandText = "select * from UserInfo";
    selectCommand.Connection = con;
    // Создание команды для вставки данных в таблицу
    DbCommand insertCommand = factory.CreateCommand();
    insertCommand.CommandText = "insert into UserInfo " +
      "(UserID, UserName, Age) values
 (:pUserID, :pUserName, :pAge)";
    insertCommand.Connection = con;
    insertCommand.Parameters.Add(paramUserID);
    insertCommand.Parameters.Add(paramUserName);
    insertCommand.Parameters.Add(paramAge);
    // Создание объекта DataAdapter
    DbDataAdapter dataAdapter = factory.CreateDataAdapter();
    dataAdapter.SelectCommand = selectCommand;
    dataAdapter.InsertCommand = insertCommand;
    // Изменение объекта DataTable
    DataRow newUser = users.NewRow();
    newUser["Номер"] = 1;
    newUser["Имя пользователя"] = "Первый пользователь";
    newUser["Возраст"] = 25;
    users.Rows.Add(newUser);
    // Синхронизация объекта DataTable с БД
    try
    {
      int rowsInserted = dataAdapter.Update(users);
      Console.WriteLine("Обработано строк: " + rowsInserted);
    }
    catch (Exception ex)
    {
      Console.WriteLine("При обновлении БД возникла ошибка: ");
      Console.WriteLine(ex.Message);
    }
    // Освобождение ресурсов
    con.Close();
  }
}
```

SourceColumnNullMapping

Свойство предоставляет или устанавливает признак допустимости null-значений столбца в наборе данных DataSet. Это позволяет объекту `DbCommandBuilder`

правильно генерировать Update-операторы для столбцов, которые могут содержать null-значения.

Если исходный столбец может содержать null-значение, то значением свойства будет true, в противном случае – значение false.

Декларация

```
public abstract bool SourceColumnNullMapping {get; set;};
```

Значение свойства

Объект типа bool – признак допустимости null-значений столбца в наборе данных DataSet.

Исключения

Отсутствуют.

SourceVersion

Свойство предоставляет или устанавливает версию столбца, используемую при загрузке свойства Value (текущая или оригинальная версия) в наборе данных DataSet, к которому привязан данный параметр.

Возможные значения свойства:

- `Original` – строка содержит исходные значения. Это свойство не существует для строк со статусом Added;
- `Current` – строка содержит текущие значения. Это свойство не существует для строк со статусом Delete;
- `Proposed` – строка содержит предложенное значение. Это свойство существует только во время редактирования строки набора данных DataSet, или для строк, не являющихся частью DataRowCollection;
- `Default` – версия по умолчанию объекта `DataRowState`. Версией по умолчанию для строк со статусом Added, Modified или Unchanged является Current, для строк со статусом Deleted – Original, для строк со статусом Detached – Proposed.

Когда метод `AcceptChanges` применяется к объекту `DataSet`, `DataTable` или `DataRow`, то все строки со статусом Deleted удаляются. Оставшиеся строки получают статус Unchanged, и их версии строк Original заменяются на Current.

При вызове метода `RejectChanges` удаляются все строки со статусом Added. Оставшиеся строки получают статус Unchanged, и их версии строк Current заменяются на Original.

Значение по умолчанию – current.

Значения DataRowVersion используются при получении значения, найденного в `DataRow` с помощью свойства Item или метода `GetChildRows` объекта `DataRow`.

DataRowVersion сообщает, какая версия объекта `DataRow` существует. Версии изменяются в следующих обстоятельствах:

- после вызова метода `BeginEdit` объекта `DataRow` при изменении значения становятся доступны значения Current и Proposed;
- после вызова метода `CancelEdit` объекта `DataRow` значение Proposed удаляется;

- после вызова метода `EndEdit` объекта `DataRow` значение Proposed становится Current;
- после вызова метода `AcceptChanges` объекта `DataRow` значение Original становится идентичным значению Current;
- после вызова метода `AcceptChanges` объекта `DataTable` значение Original становится идентичным значению Current;
- после вызова метода `RejectChanges` объекта `DataRow` значение Proposed удаляется, и версия получает значение Current.



Примечание
Свойство используется методом `UpdateCommand` во время операции Update, чтобы определить, равно ли значение параметра Current или Original (это позволяет обновить первичные ключи).

Декларация

```
public abstract DataRowVersion SourceVersion {get; set;};
```

Значение свойства

Объект `DataRowVersion`, представляющий версию строки.

Исключения

`ArgumentException`	Неизвестное имя версии строки.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ParameterSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание таблицы
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText = "Create Or Replace Table UserInfo " +
      "(UserID int, UserName Char(256), Age int)";
    cmd.ExecuteNonQuery();
```

```
// Добавление записи
cmd.CommandText = "Insert Into UserInfo " +
  "(UserID, UserName, Age) Values (1, 'Administrator', 25)";
cmd.ExecuteNonQuery();
// Создание параметра для старого значения столбца UserID
DbParameter paramOldUserID = factory.CreateParameter();
paramOldUserID.SourceColumn = "UserID";
paramOldUserID.ParameterName = ":OldUserID";
paramOldUserID.DbType = DbType.Int32;
paramOldUserID.SourceVersion = DataRowVersion.Original;
// Создание параметра для нового значения столбца UserID
DbParameter paramUserID = factory.CreateParameter();
paramUserID.SourceColumn = "UserID";
paramUserID.ParameterName = ":UserID";
paramUserID.DbType = DbType.Int32;
paramUserID.SourceVersion = DataRowVersion.Current;
// Создание параметра для столбца UserName
DbParameter paramUserName = factory.CreateParameter();
paramUserName.SourceColumn = "UserName";
paramUserName.ParameterName = ":UserName";
paramUserName.DbType = DbType.String;
// Создание параметра для столбца Age
DbParameter paramAge = factory.CreateParameter();
paramAge.SourceColumn = "Age";
paramAge.ParameterName = ":Age";
paramAge.DbType = DbType.Int32;
// Создание команды для выборки данных из таблицы
DbCommand selectCommand = factory.CreateCommand();
selectCommand.CommandText = "Select * From UserInfo";
selectCommand.Connection = con;
// Создание команды для обновления данных в таблице
DbCommand updateCommand = factory.CreateCommand();
updateCommand.CommandText = "Update UserInfo " +
  "Set UserID=:UserID, UserName=:UserName, Age=:Age " +
  "Where UserID=:OldUserID";
updateCommand.Connection = con;
updateCommand.Parameters.Add(paramOldUserID);
updateCommand.Parameters.Add(paramUserID);
updateCommand.Parameters.Add(paramUserName);
updateCommand.Parameters.Add(paramAge);
// Создание объекта DataAdapter
DbDataAdapter dataAdapter = factory.CreateDataAdapter();
dataAdapter.SelectCommand = selectCommand;
dataAdapter.UpdateCommand = updateCommand;
// Создание объекта DataSet и получение данных из БД
DataSet ds = new DataSet("UserInfo");
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    dataAdapter.Fill(ds);
    // Изменение объекта DataSet
    DataRow dataRow = ds.Tables[0].Rows[0];
    dataRow["UserID"] = 2;
    dataRow["UserName"] = "New Administrator";
    dataRow["Age"] = 50;
    // Синхронизация объекта DataSet с БД
    try
    {
      int rowsUpdated = dataAdapter.Update(ds);
      Console.WriteLine("Обработано строк: " + rowsUpdated);
    }
    catch (Exception ex)
    {
      Console.WriteLine("При обновлении БД возникла ошибка: ");
      Console.WriteLine(ex.Message);
    }
    // Освобождение ресурсов
    con.Close();
  }
}
```

Value

Свойство предоставляет или устанавливает значение параметра типа object.

Значение по умолчанию – null.

Для входных параметров это значение привязывается к параметрическому SQL-запросу (т.е. к объекту `DbCommand`), посылаемому на ЛИНТЕР-сервер.

Для выходных и процедурных параметров значение устанавливается по завершении выполнения SQL-запроса после закрытия объекта `DbDataReader`.

Если значением параметра должно быть null-значение, то при передаче его на ЛИНТЕР-сервер необходимо использовать объект `DBNull`, а не null-значение (null-значение ADO.NET-провайдером воспринимается как пустой объект, не имеющий значения).

Свойство Value перезаписывается при выполнении метода `DbDataAdapter.Update`.

Декларация

```
public abstract Object Value {get; set;};
```

Значение свойства

Объект типа Object, представляющий значение параметра.

Исключения

Отсутствуют.

Примеры

1) Получение свойства.

```
Object value = parameter.Value;
```

2) Установка свойства.

```
parameter.Value = "FORD";
```

LinterDbType

Свойство предоставляет или устанавливает тип параметра в терминах СУБД ЛИНТЕР (таблица 21).

Таблица 21. Типы параметров в терминах СУБД ЛИНТЕР (перечисление `ELinterDbType`)

Имя константы	Значение константы, приведенное к типу byte	Тип СУБД ЛИНТЕР
Bigint	28	BIGINT
Blob	7	BLOB
Bool	10	BOOLEAN
Byte	6	BYTE
Char	1	CHAR
Cursor	101	
Date	4	DATE
Double	38	DOUBLE
ExtFile	13	EXTFILE
Int	24	INTEGER
NChar	11	NCHAR
Numeric	5	NUMERIC
NVarChar	12	NVARCHAR
Real	3	REAL
Smallint	2	SMALLINT
VarByte	9	VARBYTE
VarChar	8	VARCHAR

В текущей версии ADO.NET-провайдера тип значения по умолчанию равен `0`.

Свойства LinterDbType и DbType взаимосвязаны, т.е. если задать значение DbType, то оно будет заменено на соответствующее ему значение LinterDbType.

Примечание
В текущей версии ADO.NET-провайдера свойства LinterDbType и DbType не взаимосвязаны.

Декларация

```
public System.Data.LinterClient.ELinterDbType
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
LinterDbType {set; get;};
```

Значение свойства

Объект типа `ELinterDbType`, представляющий тип параметра.

Исключения

Отсутствуют.

Примеры

1) Получение свойства параметра.

```
ELinterDbType linterDbType = parameter.LinterDbType;
```

2) Установка свойства параметра.

```
parameter.LinterDbType = ELinterDbType.Numeric;
```

Precision

Свойство предоставляет или устанавливает точность представления данных для вещественных типов данных и чисел с фиксированной точкой, т.е. максимальное количество цифр, используемое ADO.NET-провайдером для представления значения.

Для СУБД ЛИНТЕР точность задается в диапазоне от 1 до 30.

Значение по умолчанию 0. Это означает, что точность представления данных устанавливается самим ADO.NET-провайдером.

Декларация

```
public byte Precision {set; get;};
```

Значение свойства

Объект типа byte, представляющий точность.

Исключения

`ArgumentException`	Недопустимая точность.

Примеры

1) Получение свойства параметра.

```
byte precision = parameter.Precision;
```

2) Установка свойства параметра.

```
parameter.Precision = 30;
```

Scale

Свойство предоставляет или устанавливает масштаб представления данных для вещественных типов данных и чисел с фиксированной точкой, т.е. максимальное

количество цифр после десятичной точки, используемое ADO.NET-провайдером для представления значения.

Для СУБД ЛИНТЕР точность задается в диапазоне от 0 до 10.

Значение по умолчанию 0. Это означает, что масштаб представления данных устанавливается самим ADO.NET-провайдером.

Декларация

```
public byte Scale {set; get;};
```

Значение свойства

Объект типа byte, представляющий масштаб.

Исключения

Отсутствуют.

Примеры

1) Получение свойства параметра.

```
byte scale = parameter.Scale;
```

2) Установка свойства параметра.

```
parameter.Scale = 10;
```

Методы

ResetDbType

При подготовке к привязке параметров клиентское приложение может само устанавливать свойства этого параметра либо предоставить это ADO.NET-провайдеру. Чтобы указать ADO.NET-провайдеру, что именно он должен определять свойства параметра, используется метод ResetDbType, который отменяет установленный ранее вручную тип параметра и сообщает ADO.NET-провайдеру, что с этого момента определение типа параметра и всех его свойств должно выполняться самим ADO.NET-провайдером автоматически.

В результате вызова этого метода у параметра будут изменены свойства DbType и LinterDbType.

Примечание
В текущей версии ADO.NET-провайдера метод не поддерживается.

Синтаксис

```
public abstract void ResetDbType();
```

Возвращаемое значение

Значение типа void.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ParameterSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создаем параметр
    DbParameter parameter = factory.CreateParameter();
    // Присваиваем ему значение
    parameter.Value = 25;
    // Вручную задаем свойства параметра
    parameter.DbType = DbType.Int32;
    // Выводим на консоль значение параметра с установленными
 свойствами
    Console.WriteLine("Значение параметра: " + parameter.Value);
    Console.WriteLine("Тип параметра: " + parameter.DbType);
    // Делаем Reset
    parameter.ResetDbType();
    // Выводим на консоль значение параметра с новыми,
 автоматически установленными
    // свойствами
    Console.WriteLine("Значение параметра: " + parameter.Value);
    Console.WriteLine("Тип параметра: " + parameter.DbType);
  }
}
```

Класс DbParameterCollection

Класс `DbParameterCollection` является базовым классом для управления списком (коллекцией) параметров параметризованного SQL-запроса или хранимой процедуры.

Свойства класса `DbParameterCollection` приведены в таблице 22.

Таблица 22. Свойства класса DbParameterCollection

Свойство	Описание
Count	Предоставляет информацию о текущем количестве параметров в коллекции.

Свойство	Описание
IsFixedSize	Указывает, имеет ли коллекция фиксированный размер.
IsReadOnly	Предоставляет информацию о доступности коллекции параметров (удаление/добавление/модификация/ чтение параметров коллекции).
IsSynchronized	Предоставляет информацию о синхронизации коллекции параметров при многопоточной работе.
Item(Int32)	Предоставляет/устанавливает объект `DbParameter` для указанного по порядковому номеру параметра в коллекции параметров.
Item(String)	Предоставляет/устанавливает объект `DbParameter` для указанного по имени параметра в коллекции параметров.
SyncRoot	Задает объект Object, который может быть использован для синхронизации доступа к коллекции.

Методы класса `DbParameterCollection` приведены в таблице 23.

Таблица 23. Методы класса DbParameterCollection

Метод	Описание
Add(Object)	Добавляет параметр в коллекцию параметров.
Add(String, Object)	Добавляет в коллекцию параметров именованный параметр с указанным значением.
Add(String, ELinterDbType)	Добавляет в коллекцию параметров именованный параметр с типом данных в терминах СУБД ЛИНТЕР.
Add(String, ELinterDbType, Int32)	Добавляет в коллекцию параметров именованный параметр с выделенным буфером заданного размера и с типом данных в терминах СУБД ЛИНТЕР.
Add(String, ELinterDbType, Int32, String)	Добавляет в коллекцию параметров с привязкой к указанному столбцу именованный параметр с выделенным буфером заданного размера и с типом данных в терминах СУБД ЛИНТЕР.
Add(LinterDbParameter)	Добавляет в коллекцию параметров параметр типа LinterDbParameter.
AddRange	Добавляет в конец коллекции параметров массив параметров.
Clear	Удаляет все параметры из коллекции параметров.
Contains(String)	Проверяет наличие в коллекции параметров указанного именованного параметра.
Contains(Object)	Проверяет наличие указанного параметра в коллекции параметров.
CopyTo	Копирует все параметры из текущей коллекции параметров (источник) в заданный целевой массив (приемник), начиная с указанного индекса в коллекции источника.
GetEnumerator	Предоставляет перечислитель, осуществляющий перебор элементов коллекции параметров.

Метод	Описание
IndexOf(String)	Предоставляет порядковый номер заданного именованного параметра в коллекции параметров.
IndexOf(Object)	Предоставляет порядковый номер указанного объекта `DbParameter` в коллекции параметров.
Insert	Вставляет указанный параметр в коллекцию параметров на заданную позицию.
Remove	Удаляет заданный параметр из коллекции параметров.
RemoveAt(Int32)	Удаляет из коллекции параметров параметр с указанным порядковым номером.
RemoveAt(String)	Удаляет из коллекции параметров параметр с указанным именем.

Свойства

Count

Свойство предоставляет информацию о текущем количестве параметров в коллекции.

Значение по умолчанию 0.

Декларация

```
public override int Count {get;};
```

Значение свойства

Размерность коллекции параметров (значение типа System.Int32).

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CountSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание команды
    DbCommand myCommand = factory.CreateCommand();
    // Создание параметров
    DbParameter param1 = factory.CreateParameter();
```

```
    param1.ParameterName = "MAKE";
    param1.DbType = DbType.String;
    myCommand.Parameters.Add(param1);
    DbParameter param2 = factory.CreateParameter();
    param2.ParameterName = "MODEL";
    param2.DbType = DbType.String;
    myCommand.Parameters.Add(param2);
    DbParameter param3 = factory.CreateParameter();
    param3.ParameterName = "YEAR";
    param3.DbType = DbType.Int32;
    myCommand.Parameters.Add(param3);
    // Получение коллекции параметров
    DbParameterCollection myParamCollection =
 myCommand.Parameters;
    for (int i = 0; i < myParamCollection.Count; i++)
    {
      Console.WriteLine(myParamCollection[i].ParameterName);
    }
  }
}

Результат выполнения примера:

MAKE
MODEL
YEAR
```

IsFixedSize

Свойство предоставляет информацию о размерности коллекции параметров.

Размер коллекции автоматически изменяется при добавлении новых элементов и не может быть установлен в клиентском приложении. Значение по умолчанию зависит от версии .NET Framework.

Декларация

```
public override bool IsFixedSize  {get;};
```

Значение свойства

Статус размерности коллекции:

- true – фиксированный размер;
- false – переменный размер (максимальный размер определяется объемом доступной памяти и не может быть больше значения Int32.MaxValue).

Примечание
В ADO.NET-провайдере СУБД ЛИНТЕР коллекции параметров всегда переменного размера.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Исключения

Отсутствуют.

IsReadOnly

Свойство предоставляет информацию о доступности коллекции параметров.

Декларация

```
public override bool IsReadOnly  {get;};
```

Значение свойства

Статус доступности коллекции параметров:

- true – только для чтения;
- false – полный доступ (удаление/добавление/модификация/чтение параметров коллекции).

Значение по умолчанию false.

Примечание
В ADO.NET-провайдере СУБД ЛИНТЕР коллекции параметров всегда с полным доступом.

Исключения

Отсутствуют.

IsSynchronized

Свойство предоставляет информацию о синхронизации коллекции параметров при многопоточной работе.

С точки зрения работы с потоками коллекция объектов может относиться к безопасной или небезопасной.

Если несколько потоков могут вызывать свойства и методы коллекции, эти вызовы необходимо синхронизировать. В противном случае один поток может прервать операцию другого потока, и коллекция будет иметь неверное состояние. Коллекция, элементы которой защищены от подобных прерываний, называется потокобезопасной.

Свойство IsSynchronized позволяет проверить, безопасна ли текущая версия коллекции параметров.

Если коллекция параметров не синхронизирована, необходимо в свойстве SyncRoot коллекции параметров получить объект, который может использоваться для синхронизации доступа к коллекции. Это позволяет синхронизировать потоки, которые могут использовать коллекцию параметров, с помощью одного и того же объекта.

Примечание
В ADO.NET-провайдере СУБД ЛИНТЕР в коллекции параметров отсутствует метод `Synchronized`, но есть свойство SyncRoot, поэтому коллекцию параметров можно сделать потокобезопасной только с помощью механизма блокировки.

Декларация

```
public override bool IsSynchronized {get;};
```

Значение свойства

Статус коллекции параметров:

- true – коллекция синхронизирована;
- false – коллекция не синхронизирована.

Значение по умолчанию false.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class IsSynchronizedSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание команды
    DbCommand myCommand = factory.CreateCommand();
    // Создание параметров
    DbParameter param1 = factory.CreateParameter();
    param1.ParameterName = "MAKE";
    param1.DbType = DbType.String;
    myCommand.Parameters.Add(param1);
    DbParameter param2 = factory.CreateParameter();
    param2.ParameterName = "MODEL";
    param2.DbType = DbType.String;
    myCommand.Parameters.Add(param2);
    // Получение коллекции параметров
    DbParameterCollection myParamCollection =
 myCommand.Parameters;
    if (!myParamCollection.IsSynchronized)
    {
      // Блокировка
      lock (myParamCollection.SyncRoot)
      {
```

```
        foreach (DbParameter myParam in myParamCollection)
        {
          Console.WriteLine(myParam.ParameterName);
        }
      }
    }
  }
}
```

Item(Int32)

Свойство предоставляет или устанавливает объект `DbParameter` для указанного по порядковому номеру параметра в коллекции параметров.

Декларация

```
public DbParameter this [int index] {get; set;};
```

`index` – порядковый номер параметра в коллекции (отсчет начинается с `0`).

Значение свойства

Объект `DbParameter` с указанным порядковым номером.

Исключения

`IndexOutOfRangeException`	Неправильный порядковый номер параметра.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ItemSample
{
  static void Main()
  {
    // Создать фабрику классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создать команду
    DbCommand myCommand = factory.CreateCommand();
    // Создать параметр
    DbParameter myParam = factory.CreateParameter();
    myParam.ParameterName = "MAKE";
    myParam.DbType = DbType.String;
    myParam.Size = 20;
    // Вставить в коллекцию созданный параметр
```

```
    myCommand.Parameters.Add(myParam);
    // Изменить атрибуты параметра
    myCommand.Parameters[0].Size = 40;
  }
}
```

Item(String)

Свойство предоставляет или устанавливает объект `DbParameter` для указанного по имени параметра в коллекции параметров.

Декларация

```
public DbParameter this[string parameterName]{get; set;};
```

`parameterName` – имя параметра в коллекции (параметр с таким именем должен быть предварительно добавлен в коллекцию).

Примечание
В текущей версии ADO.NET-провайдера русскоязычные имена запрещены.

Значение свойства

Объект `DbParameter` с указанным именем.

Исключения

`IndexOutOfRangeException`	Неизвестное имя параметра в коллекции.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ItemSample
{
  static void Main()
  {
    // Создать фабрику классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создать команду
    DbCommand myCommand = factory.CreateCommand();
    // Создать параметр
    DbParameter myParam = factory.CreateParameter();
    myParam.ParameterName = "MAKE";
    myParam.DbType = DbType.String;
    myParam.Size = 20;
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    // Вставить в коллекцию созданный параметр
    myCommand.Parameters.Add(myParam);
    // Изменить атрибуты параметра
    myCommand.Parameters["MAKE"].Size = 40;
  }
}
```

SyncRoot

Задает объект `Object`, который может быть использован для синхронизации доступа к коллекции.

Декларация

```
[BrowsableAttribute(false)] public abstract Object SyncRoot
 {get;};
```

Значение свойства

Объект `Object`, который может быть использован для синхронизации доступа к DbParameterCollection.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class SyncRootSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание команды
    DbCommand myCommand = factory.CreateCommand();
    // Создание параметров
    DbParameter param1 = factory.CreateParameter();
    param1.ParameterName = "MAKE";
    param1.DbType = DbType.String;
    myCommand.Parameters.Add(param1);
    DbParameter param2 = factory.CreateParameter();
    param2.ParameterName = "MODEL";
    param2.DbType = DbType.String;
```

```
    myCommand.Parameters.Add(param2);
    // Получение коллекции параметров
    DbParameterCollection myParamCollection =
 myCommand.Parameters;
    // Блокировка
    lock (myParamCollection.SyncRoot)
    {
      foreach (DbParameter myParam in myParamCollection)
      {
        Console.WriteLine(myParam.ParameterName);
      }
    }
  }
}
```

Методы

Add(Object)

Метод добавляет параметр в коллекцию параметров.

Добавление выполняется в конец коллекции. В коллекции при каждом добавлении выделяется ресурс для размещения параметра. Максимальное количество параметров в коллекции ограничено объемом доступной памяти и не может быть больше значения Int32.MaxValue.

При добавлении параметра в коллекцию его свойства не изменяются.

Синтаксис

```
public override int Add(Object value);
```

`value` – добавляемый параметр (объект `DbParameter`).

Возвращаемое значение

Порядковый номер (значение типа System.Int32) добавленного параметра в коллекции. Отсчет начинается с `0`.

Исключения

`ArgumentException`	Добавляемый параметр уже существует в коллекции.
`InvalidCastException`	Переданный параметр не является LinterDbParameter.
`ArgumentNullException`	Аргумент value содержит null-значение.
`OutOfMemoryException`, `OverflowException`	Превышено допустимое количество параметров в коллекции (недостаточно ресурсов).

Пример

```
// C#
using System;
using System.Data;
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
using System.Data.Common;

class AddSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание команды
    DbCommand myCommand = factory.CreateCommand();
    // Создание параметра
    DbParameter myParam = factory.CreateParameter();
    myParam.ParameterName = "MAKE";
    myParam.DbType = DbType.String;
    myParam.Size = 40;
    // Добавление параметра в коллекцию
    myCommand.Parameters.Add(myParam);
  }
}
```

Add(String, Object)

Метод добавляет в коллекцию параметров именованный параметр с указанным значением.

Добавление выполняется в конец коллекции, при каждом добавлении выделяется ресурс для размещения параметра.

Максимальное количество параметров в коллекции ограничено объемом доступной памяти и не может быть больше значения Int32.MaxValue.

Если в коллекции уже есть параметр с таким же именем, то будет создан еще один параметр с этим же именем.

Примечание
Данный метод может использоваться в клиентских приложениях, ориентированных исключительно на работу с СУБД ЛИНТЕР.

Синтаксис

```
public LinterDbParameter Add(string parameterName, object value);
```

`parameterName` – имя добавляемого параметра (строка длиной не более 66 символов) (свойство ParameterName объекта `DbParameter`).

`value` – значение добавляемого параметра (свойство Value объекта `DbParameter`).

Возвращаемое значение

Параметр, добавленный в коллекцию (значение типа LinterDbParameter).

По умолчанию добавленный параметр имеет следующие свойства:

- имя параметра (ParameterName) – значение аргумента parameterName;
- тип данных параметра (DbType) – тип данных аргумента value;
- значение параметра (Value) – значение аргумента value;
- вид параметра (Direction) – ParameterDirection.Input;
- допустимость null-значений (IsNullable) – false;
- длина значения параметра (Size) – длина аргумента value;
- имя столбца, к которому привязан параметр (SourceColumn) – пустая строка;
- признак допустимости null-значений в наборе данных DataSet (SourceColumnNullMapping) – false;
- версия столбца в наборе данных DataSet (SourceVersion) – DataRowVersion.Current;
- тип данных параметра в терминах СУБД ЛИНТЕР (LinterDbType) – тип аргумента value;
- точность значений параметра (Precision) – 0;
- масштаб значений параметра (Scale) – 0.

Исключения

`OutOfMemoryException`, `OverflowException`	Превышено допустимое количество параметров в коллекции (недостаточно ресурсов).

Пример

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class AddSample
{
  static void Main()
  {
    // Создание команды
    LinterDbCommand myCommand = new LinterDbCommand();
    // Создание и добавление параметра в коллекцию
    LinterDbParameter myParam = myCommand.Parameters.Add("MODEL",
      "CADILLAC FLEETWOOD");
  }
}
```

Add(String, ELinterDbType)

Метод добавляет в коллекцию параметров именованный параметр с типом данных в терминах СУБД ЛИНТЕР.

Добавление выполняется в конец коллекции, при каждом добавлении выделяется ресурс для размещения параметра. Максимальное количество параметров в коллекции

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

ограничено объемом доступной памяти и не может быть больше значения Int32.MaxValue.

Если в коллекции уже есть параметр с таким же именем, то будет создан еще один параметр с этим же именем.

Примечание
Данный метод может использоваться в клиентских приложениях, ориентированных исключительно на работу с СУБД ЛИНТЕР.

Синтаксис

```
public LinterDbParameter Add(string parameterName, ELinterDbType
 type);
```

`parameterName` – имя добавляемого параметра (строка длиной не более 66 символов) (свойство ParameterName объекта `DbParameter`).

`type` – тип добавляемого параметра в терминах СУБД ЛИНТЕР (см. таблицу 21).

Возвращаемое значение

Параметр, добавленный в коллекцию (значение типа LinterDbParameter).

По умолчанию добавленный параметр имеет следующие свойства:

- имя параметра (ParameterName) – значение аргумента parameterName;
- тип данных параметра (DbType) – соответствует аргументу type;
- значение параметра (Value) – null-значение;
- вид параметра (Direction) – ParameterDirection.Input;
- допустимость null-значений (IsNullable) – false;
- длина значения параметра (Size) – соответствует аргументу type;
- имя столбца, к которому привязан параметр (SourceColumn) – пустая строка;
- признак допустимости null-значений в наборе данных DataSet (SourceColumnNullMapping) – false;
- версия столбца в наборе данных DataSet (SourceVersion) – DataRowVersion.Current;
- тип данных параметра в терминах СУБД ЛИНТЕР (LinterDbType) – значение аргумента type;
- точность значений параметра (Precision) – 0;
- масштаб значений параметра (Scale) – 0.

Исключения

`OutOfMemoryException`, `OverflowException`	Превышено допустимое количество параметров в коллекции (недостаточно ресурсов)

Пример

```
// C#
```

```
using System;
using System.Data;
using System.Data.LinterClient;

class AddSample
{
  static void Main()
  {
    // Создание команды
    LinterDbCommand myCommand = new LinterDbCommand();
    // Создание и добавление параметра в коллекцию
    LinterDbParameter myParam = myCommand.Parameters.Add("MODEL",
      ELinterDbType.NChar);
  }
}
```

Add(String, ELinterDbType, Int32)

Метод добавляет в коллекцию параметров именованный параметр с выделенным буфером заданного размера и с типом данных в терминах СУБД ЛИНТЕР.

Добавление выполняется в конец коллекции, при каждом добавлении выделяется ресурс для размещения параметра.

Максимальное количество параметров в коллекции ограничено объемом доступной памяти и не может быть больше значения Int32.MaxValue.

Максимально допустимый размер буфера параметра – 4000 байт.

Если в коллекции уже есть параметр с таким же именем, то будет создан еще один параметр с этим же именем.

Примечание
Данный метод может использоваться в клиентских приложениях, ориентированных исключительно на работу с СУБД ЛИНТЕР.

Синтаксис

```
public LinterDbParameter Add(string parameterName, ELinterDbType
 type, int size);
```

`parameterName` – имя добавляемого параметра (строка длиной не более 66 символов) (свойство ParameterName объекта `DbParameter`).

`type` – тип добавляемого параметра в терминах СУБД ЛИНТЕР (см. таблицу 21).

`size` – размер выделяемого параметру буфера (в байтах).

Возвращаемое значение

Параметр, добавленный в коллекцию (значение типа LinterDbParameter).

По умолчанию добавленный параметр имеет следующие свойства:

- имя параметра (ParameterName) – значение аргумента parameterName;
- тип данных параметра (DbType) – соответствует аргументу type;
- значение параметра (Value) – null-значение;
- вид параметра (Direction) – ParameterDirection.Input;
- допустимость null-значений (IsNullable) – false;
- длина значения параметра (Size) – значение аргумента size;
- имя столбца, к которому привязан параметр (SourceColumn) – пустая строка;
- признак допустимости null-значений в наборе данных DataSet (SourceColumnNullMapping) – false;
- версия столбца в наборе данных DataSet (SourceVersion) – DataRowVersion.Current;
- тип данных параметра в терминах СУБД ЛИНТЕР (LinterDbType) – значение аргумента type;
- точность значений параметра (Precision) – 0;
- масштаб значений параметра (Scale) – 0.

Исключения

`OutOfMemoryException`	Невозможно выделить память под буфер параметра.
`OutOfMemoryException`, `OverflowException`	Превышено допустимое количество параметров в коллекции (недостаточно ресурсов).



Примечание
Если задан недопустимый размер буфера параметра (больше 4000 байтов) или параметр не имеет тип данных BLOB, то исключение будет сгенерировано при выполнении запроса, а не при выполнении данного метода.

Пример

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class AddSample
{
  static void Main()
  {
    // Создание команды
    LinterDbCommand myCommand = new LinterDbCommand();
    // Создание и добавление параметра в коллекцию
    LinterDbParameter myParam = myCommand.Parameters.Add("MODEL",
      ELinterDbType.NChar, 40);
```

```
    }
}
```

Add(String, ELinterDbType, Int32, String)

Метод добавляет в коллекцию параметров с привязкой к указанному столбцу именованный параметр с выделенным буфером заданного размера и с типом данных в терминах СУБД ЛИНТЕР.

Добавление выполняется в конец коллекции, при каждом добавлении выделяется ресурс для размещения параметра.

Максимальное кол-во параметров в коллекции ограничено объемом доступной памяти и не может быть больше значения Int32.MaxValue.

Максимально допустимый размер буфера параметра – 4000 байт.

Если в коллекции уже есть параметр с таким же именем, то будет создан еще один параметр с этим же именем.

Примечание
Данный метод может использоваться в клиентских приложениях, ориентированных исключительно на работу с СУБД ЛИНТЕР.

Синтаксис

```
public LinterDbParameter Add(string parameterName, ELinterDbType
 type, int size, string sourceColumn);
```

`parameterName` – имя добавляемого параметра (строка длиной не более 66 символов) (свойство ParameterName объекта `DbParameter`).

`type` – тип добавляемого параметра в терминах СУБД ЛИНТЕР (см. таблицу 21).

`size` – размер выделяемого параметру буфера (в байтах).

`sourceColumn` – имя столбца в наборе DataSet, к которому должен быть привязан добавляемый параметр.

Возвращаемое значение

Параметр, добавленный в коллекцию (значение типа LinterDbParameter).

По умолчанию добавленный параметр имеет следующие свойства:

- имя параметра (ParameterName) – значение аргумента parameterName;
- тип данных параметра (DbType) – соответствует аргументу type;
- значение параметра (Value) – null-значение;
- вид параметра (Direction) – ParameterDirection.Input;
- допустимость null-значений (IsNullable) – false;
- длина значения параметра (Size) – значение аргумента size;

- имя столбца, к которому привязан параметр (SourceColumn) – значение аргумента sourceColumn;
- признак допустимости null-значений в наборе данных DataSet (SourceColumnNullMapping) – false;
- версия столбца в наборе данных DataSet (SourceVersion) – DataRowVersion.Current;
- тип данных параметра в терминах СУБД ЛИНТЕР (LinterDbType) – значение аргумента type;
- точность значений параметра (Precision) – 0;
- масштаб значений параметра (Scale) – 0.

Исключения

`OutOfMemoryException`	Невозможно выделить память под буфер параметра.
`OutOfMemoryException`, `OverflowException`	Превышено допустимое количество параметров в коллекции (недостаточно ресурсов).

Примечание
Если задан недопустимый размер буфера параметра (больше 4000 байтов) или параметр не имеет тип данных BLOB, то исключение будет сгенерировано при выполнении запроса, а не при выполнении данного метода.

Пример

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class AddSample
{
  static void Main()
  {
    // Создание команды
    LinterDbCommand myCommand = new LinterDbCommand();
    // Создание и добавление параметра в коллекцию
    LinterDbParameter myParam = myCommand.Parameters.Add("MODEL",
      ELinterDbType.NChar, 40, "Model");
  }
}
```

Add(LinterDbParameter)

Метод добавляет в коллекцию параметров параметр типа LinterDbParameter.

Добавление выполняется в конец коллекции, при каждом добавлении выделяется ресурс для размещения параметра.

Максимальное количество параметров в коллекции ограничено объемом доступной памяти и не может быть больше значения Int32.MaxValue.

Примечание
Данный метод может использоваться в клиентских приложениях, ориентированных исключительно на работу с СУБД ЛИНТЕР.

Синтаксис

```
public LinterDbParameter Add(LinterDbParameter object);
```

`object` – типизированное значение добавляемого параметра.

Возвращаемое значение

Параметр, добавленный в коллекцию (значение типа LinterDbParameter).

Свойства параметра при добавлении в коллекцию не изменяются.

Исключения

`ArgumentException`	LinterDbParameter, заданный в параметре object, уже добавлен в коллекцию.
`ArgumentNullException`	Параметр object содержит null-значение.
`OutOfMemoryException`, `OverflowException`	Превышено допустимое количество параметров в коллекции (недостаточно ресурсов).

Пример

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class AddSample
{
  static void Main()
  {
    // Создание команды
    LinterDbCommand myCommand = new LinterDbCommand();
    // Создание параметра
    LinterDbParameter myParam = new LinterDbParameter("MODEL",
      ELinterDbType.NChar, 40, "Model");
    // Добавление параметра в коллекцию
    myCommand.Parameters.Add(myParam);
  }
}
```

AddRange

Метод добавляет в конец коллекции параметров массив параметров.

В коллекции параметров при каждом добавлении выделяется необходимый ресурс для размещения добавляемого массива параметров.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Максимальное количество параметров коллекции с учетом добавленного массива параметров и максимальный размер массива добавляемых параметров ограничены объемом доступной оперативной памяти и не могут быть больше значения Int32.MaxValue.

Синтаксис

```
public abstract void AddRange(Array values);
```

`values` – массив добавляемых параметров.

Элементы массива должны иметь тип данных DbParameter.

Возвращаемое значение

Значение типа void.

Исключения

`ArgumentException`	Добавляемый параметр уже существует в коллекции.
`InvalidCastException`	Переданный параметр не является LinterDbParameter.
`ArgumentNullException`	Аргумент values содержит null-значение.
`OutOfMemoryException`, `OverflowException`	Превышено допустимое количество параметров в коллекции (недостаточно ресурсов).

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class AddRangeSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание команды
    DbCommand myCommand = factory.CreateCommand();
    // Создание параметров
    DbParameter param1 = factory.CreateParameter();
    param1.ParameterName = "MAKE";
    param1.DbType = DbType.String;
    DbParameter param2 = factory.CreateParameter();
    param2.ParameterName = "MODEL";
    param2.DbType = DbType.String;
    DbParameter param3 = factory.CreateParameter();
    param3.ParameterName = "YEAR";
```

```
    param3.DbType = DbType.Int32;
    // Создание массива параметров
    Array myArray = Array.CreateInstance(typeof(DbParameter), 3);
    myArray.SetValue(param1, 0);
    myArray.SetValue(param2, 1);
    myArray.SetValue(param3, 2);
    // Добавление массива параметров в коллекцию
    myCommand.Parameters.AddRange(myArray);
    // Отображение полученной коллекции
    for (int i = 0; i < myCommand.Parameters.Count; i++)
    {
      Console.WriteLine(myCommand.Parameters[i].ParameterName);
    }
  }
}
```

Clear

Метод удаляет все параметры из коллекции параметров.

Синтаксис

```
public abstract void Clear();
```

Возвращаемое значение

Значение типа void.

Исключения

Отсутствуют.

Пример

```
// Удаление всех параметров из DbParameterCollection
cmd.Parameters.Clear();
```

Contains(String)

Метод проверяет наличие в коллекции параметров указанного именованного параметра.

Синтаксис

```
public abstract bool Contains(string value);
```

`value` – имя интересующего параметра

Возвращаемое значение

Результат проверки:

- true – параметр в коллекции присутствует;

- false – параметр в коллекции отсутствует.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ContainsSample
{
  static void Main()
  {
    // Создать фабрику классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создать команду
    DbCommand cmd = factory.CreateCommand();
    // Создать параметр
    DbParameter prm = factory.CreateParameter();
    prm.ParameterName = "MyParam";
    prm.DbType = DbType.Decimal;
    // Добавить параметр в коллекцию параметров
    cmd.Parameters.Add(prm);
    // Проверить наличие в коллекции именованного параметра
 "MyParam"
    bool bContains = cmd.Parameters.Contains("MyParam");
    // Напечатать  "bContains = True"
    Console.WriteLine("bContains = " + bContains);
    // Проверить наличие в коллекции именованного параметра
 "NoParam"
    bContains = cmd.Parameters.Contains("NoParam");
    // Напечатать"bContains = False"
    Console.WriteLine("bContains = " + bContains);
  }
}
```

Contains(Object)

Метод проверяет наличие указанного параметра в коллекции параметров.

Примечание
В соответствии с документацией MSDN для класса DbParameterCollection, методу Contains(Object) необходимо передавать значение DbParameter.Value, но большинство ADO.NET-провайдеров реализованы таким образом, что методу Contains(Object)

необходимо передавать значение типа DbParameter. Для совместимости с этими провайдерами, ADO.NET-провайдер СУБД ЛИНТЕР также принимает значение типа DbParameter в методе LinterDbParameterCollection.Contains(Object).

Синтаксис

```
public abstract bool Contains( Object value);
```

`value` – интересующий параметр (значение типа DbParameter).

Возвращаемое значение

Результат проверки:

- true – параметр в коллекции присутствует;
- false – параметр в коллекции отсутствует.

Исключения

`InvalidCastException`	Проверяемый объект не является объектом типа DbParameter.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ContainsSample
{
  static void Main()
  {
    // Создать фабрику классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создать команду
    DbCommand cmd = factory.CreateCommand();
    // Создать параметр
    DbParameter prm1 = factory.CreateParameter();
    prm1.ParameterName = "MyParam";
    prm1.DbType = DbType.Decimal;
    // Добавить параметр в коллекцию параметров
    cmd.Parameters.Add(prm1);
    // Проверить наличие в коллекции параметра prm1
    bool bContains = cmd.Parameters.Contains(prm1);
    // Напечатать "bContains = True"
    Console.WriteLine("bContains = " + bContains);
    DbParameter prm2 = factory.CreateParameter();
    // Проверить наличие в коллекции параметра prm2
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    bContains = cmd.Parameters.Contains(prm2);
    // Напечатать "bContains = False"
    Console.WriteLine("bContains = " + bContains);
    // Освобождение ресурсов
    prm1.Dispose();
    prm2.Dispose();
    cmd.Dispose();
  }
}
```

CopyTo

Метод копирует все параметры из текущей коллекции параметров (источник) в заданный целевой массив (приемник), начиная с указанного индекса в коллекции источника.

Синтаксис

```
public override void CopyTo(Array array, int index);
```

`array` – одномерный массив Array, в который копируются параметры из коллекции (приемник).

`index` – индекс в массиве-источнике, начиная с которого следует выполнять копирование (отсчет начинается с `0`).

Возвращаемое значение

Значение типа void.

Исключения

`ArgumentNullException`	Аргумент array имеет null-значение.
`ArgumentOutOfRangeException`	Значение аргумента index меньше нуля.
`ArgumentException`	Возможные причины: • массив array является многомерным; • значение индекса массива index больше или равно длине массива array; • количество извлекаемых из массива-источника элементов превышает размер массива-приемника array; • тип данных массива-источника нельзя автоматически привести к типу массива-приемника array.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
```

```
class CopyToSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание команды
    DbCommand cmd = factory.CreateCommand();
    // Создание параметров
    DbParameter prm1 = factory.CreateParameter();
    prm1.ParameterName = "MAKE";
    prm1.DbType = DbType.String;
    prm1.Size = 40;
    cmd.Parameters.Add(prm1);
    DbParameter prm2 = factory.CreateParameter();
    prm2.ParameterName = "MODEL";
    prm2.DbType = DbType.String;
    prm2.Size = 40;
    cmd.Parameters.Add(prm2);
    DbParameter prm3 = factory.CreateParameter();
    prm3.ParameterName = "YEAR";
    prm3.DbType = DbType.Int32;
    cmd.Parameters.Add(prm3);
    // Копирование параметров в массив
    DbParameter[] prms = new DbParameter[3];
    cmd.Parameters.CopyTo(prms, 0);
    // Освобождение ресурсов
    cmd.Dispose();
  }
}
```

GetEnumerator

Метод предоставляет перечислитель, осуществляющий перебор элементов коллекции параметров.

Синтаксис

```
public override IEnumerator GetEnumerator();
```

Возвращаемое значение

Объект `IEnumerator`, который может использоваться для итерации элементов коллекции параметров.

Исключения

Отсутствуют.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Collections;

class GetEnumeratorSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание команды
    DbCommand cmd = factory.CreateCommand();
    // Создание параметров
    DbParameter prm1 = factory.CreateParameter();
    prm1.ParameterName = "MAKE";
    prm1.DbType = DbType.String;
    prm1.Size = 40;
    cmd.Parameters.Add(prm1);
    DbParameter prm2 = factory.CreateParameter();
    prm2.ParameterName = "MODEL";
    prm2.DbType = DbType.String;
    prm2.Size = 40;
    cmd.Parameters.Add(prm2);
    DbParameter prm3 = factory.CreateParameter();
    prm3.ParameterName = "YEAR";
    prm3.DbType = DbType.Int32;
    cmd.Parameters.Add(prm3);
    // Итерация коллекции параметров
    IEnumerator enumerator = cmd.Parameters.GetEnumerator();
    while (enumerator.MoveNext())
    {
      DbParameter prm = (DbParameter)enumerator.Current;
      Console.WriteLine(prm.ParameterName);
    }
    // Освобождение ресурсов
    cmd.Dispose();
  }
}
```

IndexOf(String)

Метод предоставляет порядковый номер заданного именованного параметра в коллекции параметров.

Синтаксис

```
public override int IndexOf(String parametername);
```

`parametername` – имя интересуемого параметра.

Возвращаемое значение

Порядковый номер указанного именованного параметра в коллекции параметров. Отсчет начинается с `0`.

Если параметр с указанным именем не существует, то возвращается `-1`.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class IndexOfSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание команды
    DbCommand cmd = factory.CreateCommand();
    // Создание параметров
    DbParameter prm1 = factory.CreateParameter();
    prm1.ParameterName = "MAKE";
    prm1.DbType = DbType.String;
    prm1.Size = 40;
    cmd.Parameters.Add(prm1);
    DbParameter prm2 = factory.CreateParameter();
    prm2.ParameterName = "";
    prm2.DbType = DbType.String;
    prm2.Size = 40;
    cmd.Parameters.Add(prm2);
    DbParameter prm3 = factory.CreateParameter();
    prm3.ParameterName = "YEAR";
    prm3.DbType = DbType.Int32;
    cmd.Parameters.Add(prm3);
    // Получение порядкового номера параметра
    int index = cmd.Parameters.IndexOf("YEAR");
```

```
    Console.WriteLine(index);
    // Освобождение ресурсов
    cmd.Dispose();
  }
}
```

IndexOf(Object)

Метод предоставляет порядковый номер указанного объекта DbParameter в коллекции параметров.

Синтаксис

```
public override int IndexOf(object obj);
```

`obj` – объект типа DbParameter.

Возвращаемое значение

Порядковый номер указанного объекта в коллекции параметров. Отсчет начинается с `0`.

Если объект отсутствует в коллекции, то возвращается `-1`.

Исключения

`InvalidCastException`	Заданный объект не является объектом типа DbParameter.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class IndexOfSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание команды
    DbCommand cmd = factory.CreateCommand();
    // Создание параметров
    DbParameter prm1 = factory.CreateParameter();
    prm1.ParameterName = "MAKE";
    prm1.DbType = DbType.String;
    prm1.Size = 40;
    cmd.Parameters.Add(prm1);
    DbParameter prm2 = factory.CreateParameter();
```

```
    prm2.ParameterName = "";
    prm2.DbType = DbType.String;
    prm2.Size = 40;
    cmd.Parameters.Add(prm2);
    DbParameter prm3 = factory.CreateParameter();
    prm3.ParameterName = "YEAR";
    prm3.DbType = DbType.Int32;
    cmd.Parameters.Add(prm3);
    // Получение порядкового номера параметра
    int index = cmd.Parameters.IndexOf(prm3);
    Console.WriteLine(index);
    // Освобождение ресурсов
    cmd.Dispose();
  }
}
```

Insert

Метод вставляет указанный параметр в коллекцию параметров на заданную позицию (освобождая, при необходимости, место путем сдвига существующих параметров).

Синтаксис

```
public override void Insert(int index, object obj);
```

`index` – порядковый номер вставляемого в коллекцию параметра.

Отсчет начинается с `0`. Если необходимо добавить параметр в конец коллекции, то нужно указать порядковый номер, равный количеству параметров в коллекции. Например, если в коллекции 10 параметров, то для добавления параметра в конец коллекции (на 11-ю позицию) нужно указать порядковый номер 10.

`obj` – значение System.Object вставляемого параметра.

Возвращаемое значение

Значение типа void.

Исключения

`InvalidCastException`	Тип данных добавляемого объекта не DbParameter.
`ArgumentOutOfRangeException`	Порядковый номер вставляемого параметра меньше нуля или больше количества параметров в коллекции.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
class InsertSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание команды
    DbCommand cmd = factory.CreateCommand();
    // Создание параметров
    DbParameter prm1 = factory.CreateParameter();
    prm1.ParameterName = "MAKE";
    prm1.DbType = DbType.String;
    prm1.Size = 40;
    cmd.Parameters.Add(prm1);
    DbParameter prm2 = factory.CreateParameter();
    prm2.ParameterName = "";
    prm2.DbType = DbType.String;
    prm2.Size = 40;
    cmd.Parameters.Add(prm2);
    // Создание параметра для вставки
    DbParameter prm3 = factory.CreateParameter();
    prm3.ParameterName = "YEAR";
    prm3.DbType = DbType.Int32;
    // Вставка параметра
    cmd.Parameters.Insert(1, prm3);
    // Освобождение ресурсов
    cmd.Dispose();
  }
}
```

Remove

Метод удаляет заданный параметр из коллекции параметров.

Синтаксис

```
public override void Remove(object obj);
```

`obj` – значение типа System.Object удаляемого параметра.

Возвращаемое значение

Значение типа void.

Исключения

`InvalidCastException`	Тип данных удаляемого объекта не DbParameter.
`ArgumentException`	Удаляемый объект отсутствует в коллекции параметров.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class RemoveSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание команды
    DbCommand cmd = factory.CreateCommand();
    // Добавить 2 параметра в коллекцию параметров
    DbParameter prm1 = factory.CreateParameter();
    prm1.ParameterName = "MyParam1";
    prm1.DbType = DbType.Decimal;
    cmd.Parameters.Add(prm1);
    DbParameter prm2 = factory.CreateParameter();
    prm2.ParameterName = "MyParam2";
    prm2.DbType = DbType.Decimal;
    cmd.Parameters.Add(prm2);
    // Напечатать  "cmd.Parameters.Count = 2"
    Console.WriteLine("cmd.Parameters.Count = " +
 cmd.Parameters.Count);
    // Удалить из коллекции первый параметр
    cmd.Parameters.Remove(prm1);
    // Напечатать  "cmd.Parameters.Count = 1"
    Console.WriteLine("cmd.Parameters.Count = " +
 cmd.Parameters.Count);
    // Напечатать  "cmd.Parameters[0].ParameterName = MyParam2"
    Console.WriteLine("cmd.Parameters[0].ParameterName = " +
      cmd.Parameters[0].ParameterName);
    // Освобождение ресурсов
    cmd.Dispose();
  }
}
```

RemoveAt(Int32)

Метод удаляет из коллекции параметров параметр с указанным порядковым номером.

Синтаксис

```
public override void RemoveAt(int index);
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

index – порядковый номер удаляемого из коллекции параметра.

Отсчет начинается с 0.

Возвращаемое значение

Значение типа void.

Исключения

IndexOutOfRangeException	Задан порядковый номер несуществующего в коллекции параметра.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class RemoveAtSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание команды
    DbCommand cmd = factory.CreateCommand();
    // Добавить 2 параметра в коллекцию параметров
    DbParameter prm1 = factory.CreateParameter();
    prm1.ParameterName = "MyParam1";
    prm1.DbType = DbType.Decimal;
    cmd.Parameters.Add(prm1);
    DbParameter prm2 = factory.CreateParameter();
    prm2.ParameterName = "MyParam2";
    prm2.DbType = DbType.Decimal;
    cmd.Parameters.Add(prm2);
    // Напечатать  "cmd.Parameters.Count = 2"
    Console.WriteLine("cmd.Parameters.Count = " +
 cmd.Parameters.Count);
    // Удалить из коллекции первый параметр
    cmd.Parameters.RemoveAt(0);
    // Напечатать  "cmd.Parameters.Count = 1"
    Console.WriteLine("cmd.Parameters.Count = " +
 cmd.Parameters.Count);
    // Напечатать  "cmd.Parameters[0].ParameterName = MyParam2"
    Console.WriteLine("cmd.Parameters[0].ParameterName = " +
      cmd.Parameters[0].ParameterName);
```

```
    // Освобождение ресурсов
    cmd.Dispose();
  }
}
```

RemoveAt(String)

Метод удаляет из коллекции параметров заданный именованный параметр.

Синтаксис

```
public override void RemoveAt(string  parameterName);
```

`parameterName` – имя удаляемого из коллекции параметра.

Возвращаемое значение

Значение типа void.

Исключения

`IndexOutOfRangeException`	Неизвестное имя параметра.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class RemoveAtSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание команды
    DbCommand cmd = factory.CreateCommand();
    // Добавить 2 параметра в коллекцию параметров
    DbParameter prm1 = factory.CreateParameter();
    prm1.ParameterName = "MyParam1";
    prm1.DbType = DbType.Decimal;
    cmd.Parameters.Add(prm1);
    DbParameter prm2 = factory.CreateParameter();
    prm2.ParameterName = "MyParam2";
    prm2.DbType = DbType.Decimal;
    cmd.Parameters.Add(prm2);
    // Напечатать  "cmd.Parameters.Count = 2"
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    Console.WriteLine("cmd.Parameters.Count = " +
 cmd.Parameters.Count);
    // Удалить из коллекции первый параметр
    cmd.Parameters.RemoveAt("MyParam1");
    // Напечатать  "cmd.Parameters.Count = 1"
    Console.WriteLine("cmd.Parameters.Count = " +
 cmd.Parameters.Count);
    // Напечатать  "cmd.Parameters[0].ParameterName = MyParam2"
    Console.WriteLine("cmd.Parameters[0].ParameterName = " +
      cmd.Parameters[0].ParameterName);
    // Освобождение ресурсов
    cmd.Dispose();
  }
}
```

Класс DbDataAdapter

Класс `DbDataAdapter` – это своеобразный мост между реляционной БД и отсоединенными объектами модели ADO.NET.

Взаимодействие клиентского приложения с БД возможно с помощью ранее описанных классов `DbCommand` и, если запрос возвращает значения, `DbDataReader`. При работе с этими классами каждая команда обработки данных клиентского приложения вызывает обращение к ЛИНТЕР-серверу.

Например, при добавлении в БД 100000 записей с помощью метода `DbCommand` будет выполнено 100000 обращений к ЛИНТЕР-серверу, т.к. записи добавляются по одной. При этом соединение с БД должно быть всегда открыто. Такой уровень взаимодействия с СУБД в ADO.NET называется **связным**. Во многих случаях этот уровень не является оптимальным с точки зрения производительности.

Альтернативный уровень взаимодействия с СУБД, называемый **несвязный**, предоставляет класс `DbDataAdapter`.

В отличие от связного уровня, данные, полученные с помощью адаптера данных, не обрабатываются с помощью объекта чтения данных. Вместо этого для обмена данными между клиентским приложением и ЛИНТЕР-сервером используется так называемый отсоединенный объект `DataSet`. `DataSet` – это контейнер, используемый для любого числа объектов `DataTable`, каждый из которых содержит коллекцию объектов `DataRow` и `DataColumn`.

Т.е. с помощью класса `DbDataAdapter` данные, извлеченные из БД, помещаются в объект `DataSet` и становятся доступными клиентскому приложению, и наоборот, кэшированные в `DataSet` обновления передаются в БД.

Класс `DbDataAdapter` устанавливает соединение с ЛИНТЕР-сервером автоматически и сохраняют его открытым на минимально необходимое время. Как только клиентское приложение получает запрошенные данные (т.е. объект `DataSet`), соединение с ЛИНТЕР-сервером разрывается. Клиентское приложение работает с локальной копией загруженных с БД ЛИНТЕР-сервера данных в автономном режиме, без взаимодействия с ЛИНТЕР-сервером. Оно может вставлять, удалять и модифицировать данные `DataTable`, но физически БД не будет обновлена до тех пор, пока клиентское приложение не передаст **явно** объект `DataSet` адаптеру данных для обновления.

Конструкторы класса `DbDataAdapter` приведены в таблице 24.

Таблица 24. Конструкторы класса DbDataAdapter

Конструктор	Описание
LinterDbDataAdapter()	Создает новый объект LinterDbDataAdapter со свойствами по умолчанию.
LinterDbDataAdapter(LinterDbCommand)	Создает новый объект LinterDbDataAdapter с заданным SQL-оператором.
LinterDbDataAdapter(String, LinterDbConnection)	Создает новый объект LinterDbDataAdapter с заданным SQL-оператором для указанного соединения с источником данных.
LinterDbDataAdapter(String, String)	Создает новый объект LinterDbDataAdapter с заданными SQL-оператором и строкой подключения к источнику данных.

Свойства класса `DbDataAdapter` приведены в таблице 25.

Таблица 25. Свойства класса DbDataAdapter

Свойство	Описание
AcceptChangesDuringFill	Предоставляет/устанавливает признак необходимости вызова метода `AcceptChanges` при добавлении новой строки в DataTable при выполнении любой из операций методом `Fill`.
AcceptChangesDuringUpdate	Предоставляет/устанавливает признак необходимости вызова метода `AcceptChanges` при модификации строки с помощью метода `Update`().
ContinueUpdateOnError	Предоставляет/устанавливает признак необходимости генерировать исключение при обнаружении ошибки во время обновления строки.
DeleteCommand	Предоставляет/устанавливает текст SQL-оператора, используемого для удаления записей из источника данных.
FillLoadOption	Предоставляет/устанавливает значение перечисления типа LoadOption, определяющее, как адаптер заполняет объект `DataTable` из объекта `DbDataReader`.
InsertCommand	Предоставляет/устанавливает текст SQL-оператора, используемого для добавления записей в источник данных.
MissingMappingAction	Предоставляет/устанавливает реакцию объекта `DbDataAdapter` в ситуации, когда загружаемые таблицы (или столбцы) отсутствуют в коллекции TableMappings объекта `DataSet`.
MissingSchemaAction	Предоставляет или устанавливает реакцию объекта `DbDataAdapter` в ситуации, когда схема

Свойство	Описание
	загружаемых данных не соответствует схеме в текущем объекте `DataSet`.
ReturnProviderSpecificTypes	Предоставляет/устанавливает тип данных, возвращаемых ADO.NET-провайдером при выполнении метода `Fill`.
SelectCommand	Предоставляет/устанавливает текст SQL-оператора, используемого для выборки данных в источнике данных.
TableMappings	Предоставляет коллекцию столбцов таблицы из источника данных, используемую для сопоставления со столбцами соответствующего объекта `DataTable`.
UpdateBatchSize	Задает/отменяет режим пакетного обновления записей и предоставляет/устанавливает размерность пакета обновлений.
UpdateCommand	Предоставляет/устанавливает текст SQL-оператора, используемого для обновления записей в источнике данных.

Методы класса `DbDataAdapter` приведены в таблице 26.

Таблица 26. Методы класса DbDataAdapter

Метод	Описание
Fill(DataSet)	Добавляет/обновляет строки в объекте `DataSet`.
Fill(DataTable)	Добавляет/обновляет строки в объекте `DataTable`.
Fill(DataSet, String)	Добавляет/обновляет строки в указанной таблице объекта `DataSet` для получения соответствия строкам, полученный из источника данных.
Fill(Int32, Int32, DataTable)	Добавляет/обновляет строки в объекте `DataTable` из указанной части таблицы в источнике данных.
Fill(DataSet, Int32, Int32, String)	Добавляет/обновляет строки в объекте `DataSet` из указанной части таблицы в источнике данных.
FillSchema(DataSet, SchemaType, String)	Настраивает схему объекта `DataSet` в соответствие со схемой в источнике данных.
FillSchema(DataSet, SchemaType)	Добавляет объект `DataTable` с именем "Table" в объект `DataSet` и настраивает его схему в соответствии с источником данных.
FillSchema(DataTable, SchemaType)	Настраивает схему данных для указанного объекта `DataTable`.
GetFillParameters	Предоставляет информацию о параметрах параметризованного SELECT-оператора в виде массива объектов `IDataParameter`.
Update(DataRow[])	Выполняет необходимые операторы (INSERT, UPDATE или DELETE) для изменения строк в указанном массиве объектов `DataRow`.
Update(DataSet)	Выполняет необходимые операторы (INSERT, UPDATE или DELETE) для изменения строк в первой таблице объекта `DataSet`.

Метод	Описание
Update(DataTable)	Выполняет необходимые операторы (INSERT, UPDATE или DELETE) для изменения строк в указанном объекте `DataTable`.
Update(DataSet, String)	Выполняет необходимые операторы (INSERT, UPDATE или DELETE) для изменения строк в указанной таблице объекта DataSet.

События класса `DbDataAdapter` приведены в таблице 27.

Таблица 27. События класса DbDataAdapter

Событие	Описание
FillError	Генерируется при возникновении ошибки в методе `Fill`.
RowUpdating	Генерируется перед обновлением любой строки из `DataSet` в источнике данных.
RowUpdated	Генерируется после обновления в источнике данных строки из `DataSet`.

Конструкторы

ADO.NET-провайдер СУБД ЛИНТЕР обеспечивает поддержку четырех конструкторов класса `LinterDbDataAdapter`.

Если клиентское приложение должно работать с ADO.NET-провайдерами разных СУБД, вместо конструкторов нужно использовать метод `CreateDataAdapter` класса `DbProviderFactory`.

LinterDbDataAdapter()

Синтаксис

```
public LinterDbDataAdapter();
```

Возвращаемое значение

Конструктор создает новый объект `LinterDbDataAdapter` со следующими свойствами:

- AcceptChangesDuringFill – true;
- AcceptChangesDuringUpdate – true;
- ContinueUpdateOnError – false;
- DeleteCommand – null;
- FillLoadOption – OverwriteChanges;
- InsertCommand – null;
- MissingMappingAction – Passthrough;
- MissingSchemaAction – Add;
- ReturnProviderSpecificTypes – false;
- SelectCommand – null;
- TableMappings – DataTableMappingCollection;
- UpdateBatchSize – 1;
- UpdateCommand – null.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Примечание
Данный конструктор может использоваться в клиентских приложениях, ориентированных исключительно на работу с СУБД ЛИНТЕР.

LinterDbDataAdapter(LinterDbCommand)

Синтаксис

```
public LinterDbDataAdapter(LinterDbCommand cmd);
```

`cmd` – SQL-запрос выборки данных.

Возвращаемое значение

Конструктор создает новый объект `LinterDbDataAdapter` с подготовленным `LinterDbCommand`-объектом со следующими свойствами:

- AcceptChangesDuringFill – true;
- AcceptChangesDuringUpdate – true;
- ContinueUpdateOnError – false;
- DeleteCommand – null;
- FillLoadOption – OverwriteChanges;
- InsertCommand – null;
- MissingMappingAction – Passthrough;
- MissingSchemaAction – Add;
- ReturnProviderSpecificTypes – false;
- SelectCommand – значение аргумента cmd;
- TableMappings – DataTableMappingCollection;
- UpdateBatchSize – 1;
- UpdateCommand – null.

Примечание
Данный конструктор может использоваться в клиентских приложениях, ориентированных исключительно на работу с СУБД ЛИНТЕР.

LinterDbDataAdapter(String, LinterDbConnection)

Синтаксис

```
public LinterDbDataAdapter(string commandText,
        LinterDbConnection  connection);
```

`commandText` – текст SQL-запроса.

`connection` – объект `LinterDbConnection`, используемый для соединения с СУБД.

Возвращаемое значение

Конструктор создает новый `LinterDbCommand`-объект с заданной командой и существующим `LinterDbConnection` соединением и добавляет его в новый объект `LinterDbDataAdapter` со следующими свойствами:

- AcceptChangesDuringFill – true;

- AcceptChangesDuringUpdate – true;
- ContinueUpdateOnError – false;
- DeleteCommand – null;
- FillLoadOption – OverwriteChanges;
- InsertCommand – null;
- MissingMappingAction – Passthrough;
- MissingSchemaAction – Add;
- ReturnProviderSpecificTypes – false;
- SelectCommand – объект `LinterDbCommand` с установленными свойствами (CommandText равно значению аргумента commandText, Connection равно значению аргумента connection);
- TableMappings – DataTableMappingCollection;
- UpdateBatchSize – 1;
- UpdateCommand – null.

Примечание
Данный конструктор может использоваться в клиентских приложениях, ориентированных исключительно на работу с СУБД ЛИНТЕР.

LinterDbDataAdapter(String, String)

Синтаксис

```
public LinterDbDataAdapter(string commandText,
string  connectionString);
```

`commandText` – текст SQL-запроса.

`Connection` – строка подключения к СУБД.

Возвращаемое значение

Конструктор создает новый `LinterDbCommand`-объект с заданной командой и новое LinterDbConnection соединение с заданной строкой подключения и добавляет их в новый объект `LinterDbDataAdapter` со следующими свойствами:

- AcceptChangesDuringFill – true;
- AcceptChangesDuringUpdate – true;
- ContinueUpdateOnError – false;
- DeleteCommand – null;
- FillLoadOption – OverwriteChanges;
- InsertCommand – null;
- MissingMappingAction – Passthrough;
- MissingSchemaAction – Add;
- ReturnProviderSpecificTypes – false;
- SelectCommand – объект `LinterDbCommand` с установленными свойствами (CommandText равно значению аргумента commandText, Connection равно объекту `LinterDbConnection`, у которого свойству ConnectionString установлено значению аргумента connectionString);
- TableMappings – DataTableMappingCollection;
- UpdateBatchSize – 1;
- UpdateCommand – null.

Примечание
Данный конструктор может использоваться в клиентских приложениях, ориентированных исключительно на работу с СУБД ЛИНТЕР.

Свойства

AcceptChangesDuringFill

Свойство предоставляет или устанавливает признак необходимости вызова метода `AcceptChanges` при добавлении новой строки в `DataTable` при выполнении любой из операций методом `Fill`.

Если значение свойства AcceptChangesDuringFill равно true, то добавленные при выполнении метода `Fill` в `DataSet` строки будут иметь статус (RowStatus) Unchanged (неизменённые) в противном случае (false) добавленные строки будут иметь статус Added (добавленные).

Т.к. загруженные из БД записи существуют в ней, то они не должны рассматриваться в качестве новых записей, если объект `DataSet` впоследствии откорректирует их в БД.

Присвоение свойству AcceptChangesDuringFill значения false может быть полезно, например, при копировании данных из одного источника данных в другой. Записи, загружаемые из источника данных, помечаются как New, и `DataSet` может затем вставить эти записи в другой источник данных с помощью метода `Update`() из класса `DbDataAdapter`.

Значение по умолчанию true.

Декларация

```
public bool AcceptChangesDuringFill {get; set;};
```

Значение свойства

Значение true, если метод `AcceptChanges` вызывается в объекте `DataRow`; в противном случае – значение false.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class AcceptChangesDuringFillSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
```

```
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText = "select * from auto";
    // Создание нового объекта DataSet для загрузки в него данных
    DataSet ds = new DataSet();
    // Создание объекта DbDataAdapter
    DbDataAdapter da = factory.CreateDataAdapter();
    da.SelectCommand = cmd;
    // Загрузка данных в DataSet
    da.Fill(ds, "auto");
    // Статус всех загруженных строк таблицы auto будет Unchanged
    ds.Tables["auto"].Clear();
    da.AcceptChangesDuringFill = false;
    da.Fill(ds, "auto");
    // Все добавляемые строки будут иметь статус Added
    // Меняем вручную с помощью вызова метода AcceptChanges
    // статус строк на Unchanged
    ds.AcceptChanges();
    // Освобождение ресурсов
    cmd.Dispose();
    con.Dispose();
  }
}
```

AcceptChangesDuringUpdate

Свойство предоставляет или устанавливает признак необходимости вызова метода `AcceptChanges` при модификации строки с помощью метода `Update`().

При вызове метода `Update` объекта `DataAdapter` БД может возвращать данные обратно в ADO.NET-приложение в качестве входных параметров или как первую возвращенную запись результирующего набора. ADO.NET-провайдер может извлечь эти значения и обновить соответствующие столбцы в DataRow. По умолчанию ADO.NET-провайдер вызывает метод `AcceptChanges` объекта `DataRow` после выполнения обновления. Однако, если требуется выполнять слияние обновленной строки в другом объекте `DataTable`, можно сохранить исходное значение столбца первичного ключа. Например, столбец первичного ключа, соответствующий столбцу с атрибутом AUTO INCREMENT, может содержать значения, назначенные БД, которые не соответствуют исходным значениям, назначенным в `DataRow`. По умолчанию метод `AcceptChanges` неявно вызывается после обновления записи, и исходные значения в строке, которые могут являться значениями AutoIncrement, назначенными ADO.NET-

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

провайдером, теряются. Можно сохранить исходные значения в DataRow, запрещая в ADO.NET-провайдере возможность вызова метода AcceptChanges после обновления в строке, задав для свойства AcceptChangesDuringUpdate значение false, что приведет к сохранению исходных значений.

Декларация

```
public bool AcceptChangesDuringUpdate {get; set;};
```

Значение свойства

Значение true, если метод AcceptChanges вызывается при вызове метода Update; в противном случае – значение false.

Значением по умолчанию являются true.

Исключения

Отсутствуют.

Пример

Пример демонстрирует извлечение измененных строк из объекта DataTable, использование DbDataAdapter для модификации источника данных и загрузки нового значения соответствующего столбца. Присвоив свойству AcceptChangesDuringUpdate объекта значение false, сохраняем исходное автоинкрементное значение. Новые данные могут быть затем объединены с исходными из DataTable даже если новое соответствующее значение не совпадает с исходным автоинкрементным значением в объекте DataTable.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class AcceptChangesDuringUpdateSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText =
      "create if not exists table policy ( " +
      "policy_id integer autoinc primary key, " +
```

```
  "policy_name varchar(70))";
cmd.ExecuteNonQuery();
// Создание объекта DbParameter
DbParameter par = factory.CreateParameter();
par.ParameterName = ":policy_name";
par.SourceColumn = "policy_name";
par.Direction = ParameterDirection.Input;
par.DbType = DbType.String;
par.Size = 70;
// Создание объекта DbDataAdapter
DbDataAdapter adapter = factory.CreateDataAdapter();
adapter.SelectCommand = factory.CreateCommand();
adapter.SelectCommand.Connection = con;
adapter.SelectCommand.CommandText =
  "select policy_id, policy_name from policy";
adapter.InsertCommand = factory.CreateCommand();
adapter.InsertCommand.Connection = con;
adapter.InsertCommand.CommandText =
  "insert into policy (policy_name) values (:policy_name); " +
  "select last_autoinc as policy_id";
adapter.InsertCommand.Parameters.Add(par);
adapter.InsertCommand.UpdatedRowSource = UpdateRowSource.Both;
adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;
adapter.AcceptChangesDuringUpdate = false;
// Заполнение объекта DataTable данными из таблицы БД
DataTable policy = new DataTable();
adapter.Fill(policy);
// Добавление записи в таблицу DataTable
DataRow newRow = policy.NewRow();
newRow["policy_name"] = "Политика " + DateTime.Now.ToString();
policy.Rows.Add(newRow);
// Получение таблицы, содержащей все изменения
DataTable dataChanges = policy.GetChanges();
// Обновление БД
adapter.Update(dataChanges);
// Освобождение ресурсов
con.Close();
// Отображение исходной таблицы
Console.WriteLine("Строки до объединения");
foreach (DataRow rowBefore in policy.Rows)
{
  Console.WriteLine("{0}: {1}", rowBefore[0], rowBefore[1]);
}
// Объединение исходной таблицы и таблицы с изменениями
policy.Merge(dataChanges);
// Фиксация изменений
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    policy.AcceptChanges();
    // Отображение таблицы после объединения
    Console.WriteLine("Строки после объединения");
    foreach (DataRow rowAfter in policy.Rows)
    {
      Console.WriteLine("{0}: {1}", rowAfter[0], rowAfter[1]);
    }
  }
}
```

ContinueUpdateOnError

Свойство предоставляет или устанавливает признак необходимости генерировать исключение при обнаружении ошибки во время обновления строки.

Если для свойства ContinueUpdateOnError установлено значение true, то при обнаружении ошибки во время обновления строки исключение не выдается. Обновление строки пропускается, и информация об ошибке помещается в свойство RowError строки с ошибкой. Объект `DataAdapter` продолжает обновление последующих строк.

Если для свойства ContinueUpdateOnError установлено значение false, то при обнаружении ошибки во время обновления строки исключение выдается.

Значением по умолчанию является false.

Декларация

```
public bool ContinueUpdateOnError {get; set;};
```

Значение свойства

Значение true для продолжения обновления данных без генерации исключения; в противном случае – значение false.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ContinueUpdateOnErrorSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
```

```
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText =
      "create or replace table users ( " +
      "id integer primary key, name varchar(70));" +
      "insert into users (id, name) values (0, 'Пользователь A');"
+
      "insert into users (id, name) values (1, 'Пользователь
B');";
    cmd.ExecuteNonQuery();
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = factory.CreateCommand();
    adapter.SelectCommand.Connection = con;
    adapter.SelectCommand.CommandText =
      "select id, name from users";
    adapter.ContinueUpdateOnError = true;
    // Создание объекта DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Заполнение объекта DataTable данными из таблицы БД
    DataTable users = new DataTable();
    adapter.Fill(users);
    // Изменяем записи в таблице DataTable
    users.Rows[0]["name"] = "Новый пользователь A";
    users.Rows[1]["name"] = "Новый пользователь B";
    // Имитируем несогласованное изменение первой записи в БД
    cmd.CommandText =
      "update users set name = 'Новый пользователь X' where id =
0";
    cmd.ExecuteNonQuery();
    // Теперь обновление БД должно завершиться ошибкой
    adapter.Update(users);
    // Отображение таблицы после обновления
    Console.WriteLine("Строки таблицы после обновления:");
    foreach (DataRow row in users.Rows)
    {
      Console.Write("{0}: '{1}' ", row[0], row[1]);
      if (row.HasErrors)
      {
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
        Console.WriteLine("(при обновлении строки произошла
 ошибка)");
      }
      else
      {
        Console.WriteLine("(обновление строки выполнено
 успешно)");
      }
    }
    // Освобождение ресурсов
    con.Close();
  }
}
```

DeleteCommand

Свойство предоставляет или устанавливает текст команды (SQL-запроса), используемой для удаления записей из источника данных.

При использовании метода `Update` в случае, когда это свойство не задано, и данные первичного ключа имеются в объекте `DataSet`, свойство DeleteCommand создается автоматически, если объект `DbDataAdapter` связан с объектом `DbCommandBuilder`.

Декларация

```
[BrowsableAttribute(false)]
public DbCommand DeleteCommand {get; set;};
```

Значение свойства

Интерфейс IDbCommand, используемый во время применения метода `Update` для удаления записей в источнике данных, соответствующих удаленным строкам в наборе данных.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class DeleteCommandSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
```

```
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText =
      "create or replace table users ( " +
      "id integer primary key, name varchar(70));" +
      "insert into users (id, name) values (0, 'Пользователь A');"
+
      "insert into users (id, name) values (1, 'Пользователь
B');";
    cmd.ExecuteNonQuery();
    // Создание команды для выборки записей
    DbCommand selectCommand = factory.CreateCommand();
    selectCommand.Connection = con;
    selectCommand.CommandText =
      "select id, name from users";
    // Создание параметра для удаления записи
    DbParameter par = factory.CreateParameter();
    par.ParameterName = ":id";
    par.SourceColumn = "id";
    par.Direction = ParameterDirection.Input;
    par.DbType = DbType.Int32;
    // Создание команды для удаления записи
    DbCommand deleteCommand = factory.CreateCommand();
    deleteCommand.Connection = con;
    deleteCommand.CommandText =
      "delete from users where id = :id";
    deleteCommand.Parameters.Add(par);
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = selectCommand;
    adapter.DeleteCommand = deleteCommand;
    // Заполнение объекта DataTable данными из таблицы БД
    DataTable users = new DataTable();
    adapter.Fill(users);
    // Удаление записей из таблицы DataTable
    users.Rows[0].Delete();
    // Обновление БД
    adapter.Update(users);
    // Отображение таблицы после обновления
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    Console.WriteLine("Строки таблицы после обновления:");
    foreach (DataRow row in users.Rows)
    {
      Console.Write("{0}: '{1}' ", row[0], row[1]);
    }
    // Освобождение ресурсов
    con.Close();
  }
}
```

FillLoadOption

Свойство предоставляет или устанавливает значение перечисления типа LoadOption, определяющее, как адаптер заполняет объект `DataTable` из объекта `DbDataReader`.

Свойство устанавливает правило обработки записей в текущем объекте `DataTable` с совпадающими загружаемыми записями из источника данных.

Для правильной работы данного свойства в таблице `DataTable` должен быть задан первичный ключ. Если первичный ключ не задан, то записи `DataTable` не будут изменены, а записи из источника данных будут добавлены в конец таблицы `DataTable`.

Декларация

```
public LoadOption FillLoadOption {get; set;};
```

Значение свойства

Значение LoadOption:

- OverwriteChanges (значение по умолчанию). Значение OverwriteChanges заставляет `DataTable` заменять любые изменения, внесенные в обе версии: «текущая» (current) и «исходная» (original) соответствующего `DataRow` текущего объекта `DataTable`, значениями, загружаемыми объектом `DbDataAdapter` из источника данных. Т.е. в `DataTable` будут содержаться только новые значения;
- PreserveChanges. Все внесенные текущие изменения в `DataRow` будут сохраняться с версией «текущая» (current), а все вновь загружаемые объектом `DbDataAdaper` значения будут помещаться в соответствующий `DataRow` с версией «исходная» (original). Т.е. в `DataTable` будут содержаться как старые значения, так и новые;
- Upsert. Свойство Upsert является комбинацией двух операций: UPdate и inSERT. Если загружаемая из источника данных запись в `DataTable` уже существует, то она будет заменена с указанием версии «текущая» (current), иначе запись будет добавлена с пометкой «представлена к обработке».

Исходная версия данных в столбце не изменяется.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
```

```
using System.Data.Common;

class FillLoadOptionSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText =
      "create or replace table users ( " +
      "id integer primary key, name varchar(70));" +
      "insert into users (id, name) values (0, 'Пользователь A');"
 +
      "insert into users (id, name) values (1, 'Пользователь
 B');";
    cmd.ExecuteNonQuery();
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = factory.CreateCommand();
    adapter.SelectCommand.Connection = con;
    adapter.SelectCommand.CommandText =
      "select id, name from users";
    // Создание объекта DataTable
    DataTable users = CreateDataTable();
    Console.WriteLine("Исходная таблица:");
    Output(users);
    // Исходное и текущее значения будут заменены значениями,
    // загруженными из БД
    adapter.FillLoadOption = LoadOption.OverwriteChanges;
    adapter.Fill(users);
    Console.WriteLine("LoadOption.OverwriteChanges:");
    Output(users);
    // Исходные значения будут заменены на значения из БД.
    // Текущие значения строки не будут изменены.
    users = CreateDataTable();
    adapter.FillLoadOption = LoadOption.PreserveChanges;
    adapter.Fill(users);
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    Console.WriteLine("LoadOption.PreserveChanges:");
    Output(users);
    // Исходные значения не будут изменены.
    // Текущие значения будут заменены на значения из БД.
    users = CreateDataTable();
    adapter.FillLoadOption = LoadOption.Upsert;
    adapter.Fill(users);
    Console.WriteLine("LoadOption.Upsert:");
    Output(users);
    // Освобождение ресурсов
    con.Close();
  }
  private static DataTable CreateDataTable()
  {
    DataTable users = new DataTable();
    users.Columns.Add("id", typeof(int));
    users.Columns.Add("name", typeof(string));
    users.PrimaryKey = new DataColumn[] { users.Columns[0] };
    users.Rows.Add(0, "Пользователь X");
    users.Rows.Add(1, "Пользователь Y");
    users.AcceptChanges();
    users.Rows[0]["name"] = "Новый Пользователь X";
    users.Rows[1]["name"] = "Новый Пользователь Y";
    return users;
  }
  private static void Output(DataTable users)
  {
    Console.WriteLine("ID | Исходное значение | Текущее
 значение");
    foreach (DataRow row in users.Rows)
    {
      Console.WriteLine("{0,-2} | {1,-17} | {2}", row[0],
        row.HasVersion(DataRowVersion.Original) ? row[1,
 DataRowVersion.Original] : "",
        row.HasVersion(DataRowVersion.Current) ? row[1,
 DataRowVersion.Current] : "");
    }
  }
}
```

Результат выполнения примера:

```
Исходная таблица:
ID | Исходное значение | Текущее значение
0  | Пользователь X    | Новый Пользователь X
1  | Пользователь Y    | Новый Пользователь Y
```

```
LoadOption.OverwriteChanges:
ID | Исходное значение | Текущее значение
0  | Пользователь A    | Пользователь A
1  | Пользователь B    | Пользователь B
LoadOption.PreserveChanges:
ID | Исходное значение | Текущее значение
0  | Пользователь A    | Новый Пользователь X
1  | Пользователь B    | Новый Пользователь Y
LoadOption.Upsert:
ID | Исходное значение | Текущее значение
0  | Пользователь X    | Пользователь A
1  | Пользователь Y    | Пользователь B
```

InsertCommand

Свойство предоставляет или устанавливает текст команды (SQL-запроса), используемой для добавления записей в источник данных.

При использовании метода `Update` в случае, когда это свойство не задано, и данные первичного ключа имеются в объекте `DataSet`, свойство InsertCommand будет создаваться автоматически, если объект `DbDataAdapter` связан с объектом `DbCommandBuilder`.

Декларация

```
[BrowsableAttribute(false)]
public DbCommand InsertCommand {get; set;};
```

Значение свойства

Интерфейс IDbCommand, используемый во время применения метода `Update` для вставки записей, соответствующих новым строкам в наборе данных, в источник данных.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class InsertCommandSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
```

```
DbConnection con = factory.CreateConnection();
con.ConnectionString =
  "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
con.Open();
// Создание таблицы БД
DbCommand cmd = factory.CreateCommand();
cmd.Connection = con;
cmd.CommandText =
  "create or replace table users ( " +
  "id integer primary key, name varchar(70))";
cmd.ExecuteNonQuery();
// Создание команды для выборки записей
DbCommand selectCommand = factory.CreateCommand();
selectCommand.Connection = con;
selectCommand.CommandText =
  "select id, name from users";
// Создание параметров для добавления записей
DbParameter parId = factory.CreateParameter();
parId.ParameterName = ":id";
parId.SourceColumn = "id";
parId.Direction = ParameterDirection.Input;
parId.DbType = DbType.Int32;
DbParameter parName = factory.CreateParameter();
parName.ParameterName = ":name";
parName.SourceColumn = "name";
parName.Direction = ParameterDirection.Input;
parName.DbType = DbType.String;
parName.Size = 70;
// Создание команды для добавления записи
DbCommand insertCommand = factory.CreateCommand();
insertCommand.Connection = con;
insertCommand.CommandText =
  "insert into users (id,name) values (:id,:name)";
insertCommand.Parameters.Add(parId);
insertCommand.Parameters.Add(parName);
// Создание объекта DbDataAdapter
DbDataAdapter adapter = factory.CreateDataAdapter();
adapter.SelectCommand = selectCommand;
adapter.InsertCommand = insertCommand;
// Заполнение объекта DataTable данными из таблицы БД
DataTable users = new DataTable();
adapter.Fill(users);
// Добавление записей в таблицу DataTable
users.Rows.Add(0, "Пользователь A");
users.Rows.Add(1, "Пользователь B");
// Обновление БД
```

```
    adapter.Update(users);
    // Отображение таблицы после обновления
    Console.WriteLine("Строки таблицы после обновления:");
    foreach (DataRow row in users.Rows)
    {
      Console.WriteLine("{0}: '{1}' ", row[0], row[1]);
    }
    // Освобождение ресурсов
    con.Close();
  }
}
```

MissingMappingAction

Свойство предоставляет или устанавливает реакцию объекта `DbDataAdapter` в ситуации, когда загружаемые таблицы (или столбцы) отсутствуют в коллекции TableMappings объекта `DataSet`.

Декларация

```
public MissingMappingAction MissingMappingAction {get; set;};
```

Значение свойства

Значение MissingMappingAction:

- Passthrough (значение по умолчанию). Создается новый объект (столбец или таблица), который добавляется к `DataSet` с использованием исходного имени;
- Ignore. Данные (столбец или таблица), для которых сопоставление отсутствует, игнорируются;
- Error. Отсутствует сопоставление указанного столбца со значениями из коллекции TableMappings объекта `DataSet`.

Исключения

`ArgumentException`	Устанавливаемое значение не является одним из значений MissingMappingAction.
`InvalidOperationException`	Отсутствует возможность сопоставления загружаемых данных с данными из коллекции TableMappings объекта `DataSet`.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class MissingMappingActionSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
```

```
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText =
      "create or replace table users ( " +
      "id integer primary key, name varchar(70));" +
      "insert into users (id, name) values (0, 'Пользователь A');"
+
      "insert into users (id, name) values (1, 'Пользователь
B');";
    cmd.ExecuteNonQuery();
    // Создание объекта DataSet
    DataSet ds = new DataSet();
    DataTable users = ds.Tables.Add("Пользователи");
    users.Columns.Add("Номер", typeof(int));
    users.Columns.Add("Имя пользователя", typeof(string));
    // Отображение колонок таблицы DataTable на поля БД
    DataTableMapping mapping = new DataTableMapping("Table",
"Пользователи");
    mapping.ColumnMappings.Add("ID", "Номер");
    mapping.ColumnMappings.Add("NAME", "Имя пользователя");
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = factory.CreateCommand();
    adapter.SelectCommand.Connection = con;
    adapter.SelectCommand.CommandText =
      "select id, name, rowtime from users";
    adapter.TableMappings.Add(mapping);
    adapter.MissingMappingAction =
MissingMappingAction.Passthrough;
    // Заполнение объекта DataSet данными из таблицы БД
    adapter.Fill(ds);
    // Отображение столбцов таблицы
    Console.WriteLine("Столбцы таблицы:");
    foreach (DataColumn column in users.Columns)
    {
      Console.Write("{0} | ", column.ColumnName);
    }
    Console.WriteLine();
```

```
    // Отображение строк таблицы
    Console.WriteLine("Строки таблицы:");
    foreach (DataRow row in users.Rows)
    {
      foreach (DataColumn column in users.Columns)
      {
        Console.Write("{0} | ", row[column.ColumnName]);
      }
      Console.WriteLine();
    }
    // Освобождение ресурсов
    con.Close();
  }
}

Результат выполнения примера:

Столбцы таблицы:
Номер | Имя пользователя | ROWTIME |
Строки таблицы:
0 | Пользователь A | 03.12.2012 10:44:26 |
1 | Пользователь B | 03.12.2012 10:44:26 |
```

MissingSchemaAction

Свойство предоставляет или устанавливает реакцию объекта `DbDataAdapter` в ситуации, когда схема загружаемых данных не соответствует схеме в текущем объекте `DataSet`.

Установка значений этого свойства требуется в том случае, если объект `DataSet` не содержит таблицу (столбец) с именем, заданным при привязке таблицы (столбца) к выборке данных. Значения свойства указывают клиентскому приложению, как поступать в этом случае.

Декларация

```
public MissingSchemaAction MissingSchemaAction {get; set;};
```

Значение свойства

Значение MissingSchemaAction:

- Add (значение по умолчанию). Добавляет необходимые таблицы или столбцы для завершения схемы (ограничения целостности не учитываются);
- Ignore. Игнорирует лишние столбцы;
- Error. Отсутствует сопоставление указанного столбца, создается исключение InvalidOperationException;
- AddWithKey. Добавление информации о схеме в объект `DataSet` перед его заполнением данными предполагает, что ограничения целостности (первичный ключ, unique-значения, not null-значения и др.) включается с `DataTable` в объект `DataSet`. В результате, когда делается дополнительный вызов для заполнения `DataSet`, информация об ограничениях целостности используется в случае

совпадения добавляемых строк из источника данных с текущими строками в DataTable, и текущие данные в таблицах заменяются данными из источника данных.

Без информации о схеме данных (например, некоторый столбец является первичным ключом и, следовательно, в DataSet не должно быть строк с одинаковыми значениями первичного ключа) новые строки из источника данных добавляются в DataSet, порождая дублирование строк.

Для исключения такой ситуации необходимо, чтобы DbDataAdapter принимал во внимание схему загружаемых данных, что и указывается в свойстве MissingSchemaAction.

Использование метода FillSchema или присвоение свойству MissingSchemaAction значения AddWithKey требует дополнительных затрат в источнике данных для получения схемы (метаданных) о столбцах с ограничением целостности, что может снизить производительность, поэтому, если клиентское приложение знает об ограничениях целостности загружаемых данных, рекомендуется указывать их в явном виде.



Примечание

Использование значения AddWithKey данного свойства обязательно в случае необходимости обрабатывать в клиентском приложении исключение «Constraint Exception», которое может быть сгенерировано при вызове метода Add(DataRow) или Add(Object[]) класса DataRowCollection.

Исключения

ArgumentException	Устанавливаемое значение не является одним из значений MissingSchemaAction.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class MissingSchemaActionSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
```

```
    cmd.Connection = con;
    cmd.CommandText =
      "create or replace table users ( " +
      "id integer primary key, name varchar(70));" +
      "insert into users (id, name) values (0, 'Пользователь A');"
+
      "insert into users (id, name) values (1, 'Пользователь
B');";
    cmd.ExecuteNonQuery();
    // Создание команды для выборки записей
    DbCommand selectCommand = factory.CreateCommand();
    selectCommand.Connection = con;
    selectCommand.CommandText =
      "select id, name from users";
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = selectCommand;
    adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;
    // Создание объекта DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Заполнение объекта DataTable данными из таблицы БД
    DataTable users = new DataTable();
    adapter.Fill(users);
    try
    {
      // При добавление записи, нарушающей уникальность первичного
      // ключа, генерируется исключение
      users.Rows.Add(0, "Пользователь X");
    }
    catch (ConstraintException ex)
    {
      Console.WriteLine(ex.Message);
      // Добавление корректной записи
      users.Rows.Add(2, "Пользователь Y");
    }
    // Обновление БД
    adapter.Update(users);
    // Отображение таблицы после обновления
    Console.WriteLine("Строки таблицы после обновления:");
    foreach (DataRow row in users.Rows)
    {
      Console.WriteLine("{0}: '{1}' ", row[0], row[1]);
    }
    // Освобождение ресурсов
    con.Close();
```

```
  }
}

Результат выполнения примера:

Column 'ID' is constrained to be unique.  Value '0' is already
 present.
Строки таблицы после обновления:
0: 'Пользователь A'
1: 'Пользователь B'
2: 'Пользователь Y'
```

ReturnProviderSpecificTypes

Свойство предоставляет или устанавливает тип данных, возвращаемых ADO.NET-провайдером при выполнении метода `Fill`.

Декларация

```
public virtual bool ReturnProviderSpecificTypes {get; set;};
```

Значение свойства

Тип возвращаемых ADO.NET-провайдером данных:

- true – определяется ADO.NET-провайдером;
- false – согласно CLS-спецификации (Common Language Specification) (значение по умолчанию).

Примечание
В текущей версии ADO.NET-провайдер всегда возвращает типы данных CLS.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ReturnProviderSpecificTypesSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
```

```
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание команды для выборки записей
    DbCommand selectCommand = factory.CreateCommand();
    selectCommand.Connection = con;
    selectCommand.CommandText =
      "select cast 12.56 as decimal, cast 12345 as bigint";
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = selectCommand;
    adapter.ReturnProviderSpecificTypes = true;
    // Заполнение объекта DataTable данными из таблицы БД
    DataTable users = new DataTable();
    adapter.Fill(users);
    // Отображение строк таблицы
    Console.WriteLine("Строки таблицы:");
    foreach (DataRow row in users.Rows)
    {
      Console.WriteLine("{0} | {1} ", row[0], row[1]);
    }
    // Освобождение ресурсов
    con.Close();
  }
}
```

SelectCommand

Свойство предоставляет или устанавливает текст команды (SQL-запроса), применяемой для выборки данных в источнике данных (используется в методах `Fill` и `FillSchema` для заполнения данными объектов `DataSet`, `DataTable` или `DataRow[]`).

Декларация

```
[BrowsableAttribute(false)]
public DbCommand SelectCommand {get; set;};
```

Значение свойства

Интерфейс IDbCommand, используемый во время применения метода `Fill` для выбора записей из источника данных для размещения в наборе данных.

Исключения

Отсутствуют.

Примеры

1) Явно установить свойство SelectCommand объекта `DbDataAdapter`.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class SelectCommandSample
{
  static void Main()
  {
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    DbCommand selectCommand = factory.CreateCommand();
    selectCommand.CommandText = "select personid, model from
 auto";
    selectCommand.Connection = con;
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = selectCommand;
    DataTable auto = new DataTable();
    adapter.Fill(auto);
    Console.WriteLine("Строки таблицы:");
    foreach (DataRow row in auto.Rows)
    {
      Console.WriteLine("{0}: '{1}' ", row[0], row[1]);
    }
  }
}
```

2) Вместо установки свойства SelectCommand передать объект `Command` конструктору класса `LinterDbDataAdapter`. ADO.NET-провайдер свяжет его со свойством SelectCommand объекта `DataAdapter`.

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class SelectCommandSample
{
  static void Main()
  {
    LinterDbConnection con = new LinterDbConnection(
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8");
    LinterDbCommand selectCommand = new LinterDbCommand(
      "select id, name from users", con);
```

```
    LinterDbDataAdapter adapter = new
 LinterDbDataAdapter(selectCommand);
    DataTable auto = new DataTable();
    adapter.Fill(auto);
    Console.WriteLine("Строки таблицы:");
    foreach (DataRow row in auto.Rows)
    {
      Console.WriteLine("{0}: '{1}' ", row[0], row[1]);
    }
  }
}
```

3) Если конструктору LinterDbDataAdapter передан текст SQL-запроса для выборки данных и объект `LinterDbConnection`, то на основе этих значений будет создан объект `LinterDbCommand` и установлено свойство SelectCommand объекта `LinterDbDataAdapter`.

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class SelectCommandSample
{
  static void Main()
  {
    LinterDbConnection con = new LinterDbConnection(
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8");
    string selectCommandText =
      "select personid, model from auto";
    LinterDbDataAdapter adapter = new LinterDbDataAdapter(
      selectCommandText, con);
    DataTable auto = new DataTable();
    adapter.Fill(auto);
    Console.WriteLine("Строки таблицы:");
    foreach (DataRow row in auto.Rows)
    {
      Console.WriteLine("{0}: '{1}' ", row[0], row[1]);
    }
  }
}
```

4) Если конструктору LinterDbDataAdapter передан текст SQL-запроса для выборки данных и строка подключения, то на основе этих значений будет создан объект `LinterDbCommand` и установлено свойство SelectCommand объекта `LinterDbDataAdapter`.

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class SelectCommandSample
{
  static void Main()
  {
    string connectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    string selectCommandText =
      "select personid, model from auto";
    LinterDbDataAdapter adapter = new LinterDbDataAdapter(
      selectCommandText, connectionString);
    DataTable auto = new DataTable();
    adapter.Fill(auto);
    Console.WriteLine("Строки таблицы:");
    foreach (DataRow row in auto.Rows)
    {
      Console.WriteLine("{0}: '{1}' ", row[0], row[1]);
    }
  }
}
```

TableMappings

Свойство предоставляет коллекцию столбцов таблицы из источника данных, используемую для сопоставления со столбцами соответствующего объекта `DataTable`.

При согласовании изменений объект `DataAdapter` применяет коллекцию DataTableMappingCollection для связи имен столбцов, использованных источником данных, с именами столбцов, использованных `DataTable`.

При изменении набора данных `DataSet` метод `Fill` должен первым делом определить, существует ли объект `DataTable`, в который надо будет размещать выборку данных, формируемую SelectCommand. Он делает это путем просмотра одновременно имен существующих таблиц и любой отображаемой объектом `DbDataAdapter` таблицы и столбца. По умолчанию, если таблица не существует в `DataSet` или нет таблицы с именем «Table», то создается новая таблица с именем «Table» и её столбцы создаются с использованием имен и типов данных из источника данных. Как альтернатива, имя этой таблицы может быть передано во втором аргументе метода `Fill` и она будет использована для размещения в ней результатов выборки данных. Если результатом команды является несколько выборок данных, то дополнительные таблицы именуются как «Table1» «Table2» и т.д.

Если метод `Fill` выявляет дубликаты имен столбцов, то они будут именоваться как «имя столбца»1, «имя столбца»2 и т.д. Не именованные столбцы (такие, как результат агрегатных функций) будут иметь имена «столбец1», «столбец2».

Если клиентское приложение не устраивает такое именование, то оно всегда может использовать явные имена для столбцов.

Когда используется режим перезаписи и передается объект `DataTable`, то метод `Fill` сначала просматривает отображение таблицы, и если она не найдена, просто добавляет таблицу независимо от её имени. Свойства MissingSchemaAction и MissingMappingAction оказывают влияние на этот процесс. Конечным результатом является то, что данные могут быть добавлены в `DataSet` или `DataTable` без отображения любой таблицы или столбца, независимо от того, какие таблицы или столбцы уже существуют.

Декларация

```
public DataTableMappingCollection TableMappings {get;};
```

Значение свойства

Коллекция, обеспечивающая основное сопоставление между возвращенными записями и записями в объекте `DataSet`.

По умолчанию используется пустая коллекция.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class TableMappingsSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText =
      "create or replace table users ( " +
      "id integer primary key, name varchar(70));" +
      "insert into users (id, name) values (0, 'Пользователь A');"
 +
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
      "insert into users (id, name) values (1, 'Пользователь
 B');";
    con.Open();
    cmd.ExecuteNonQuery();
    con.Close();
    // Создание объекта DataSet
    DataSet ds = new DataSet();
    DataTable users = ds.Tables.Add("Пользователи");
    users.Columns.Add("Номер", typeof(int));
    users.Columns.Add("Имя пользователя", typeof(string));
    // Отображение столбцов таблицы DataTable на поля БД
    DataTableMapping mapping = new DataTableMapping("Table",
 "Пользователи");
    mapping.ColumnMappings.Add("ID", "Номер");
    mapping.ColumnMappings.Add("NAME", "Имя пользователя");
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = factory.CreateCommand();
    adapter.SelectCommand.Connection = con;
    adapter.SelectCommand.CommandText =
      "select id, name from users";
    adapter.TableMappings.Add(mapping);
    // Заполнение объекта DataSet данными из таблицы БД
    adapter.Fill(ds);
    // Отображение столбцов таблицы
    Console.WriteLine("Столбцы таблицы:");
    foreach (DataColumn column in users.Columns)
    {
      Console.Write("{0} | ", column.ColumnName);
    }
    Console.WriteLine();
    // Отображение строк таблицы
    Console.WriteLine("Строки таблицы:");
    foreach (DataRow row in users.Rows)
    {
      foreach (DataColumn column in users.Columns)
      {
        Console.Write("{0} | ", row[column.ColumnName]);
      }
      Console.WriteLine();
    }
  }
}
```

Результат выполнения примера:

```
Столбцы таблицы:
Номер | Имя пользователя |
Строки таблицы:
0 | Пользователь A |
1 | Пользователь B |
```

UpdateBatchSize

Свойство задает/отменяет режим пакетного обновления записей и предоставляет или устанавливает размерность пакета обновлений.

Свойство UpdateBatchSize используется для обновления источника данных с учетом изменений, полученных из объекта `DataSet`. Пакетная обработка позволяет увеличить производительность клиентского приложения за счет сокращения количества циклов приема-передачи для обработки данных на ЛИНТЕР-сервере.

Выполнение слишком большого пакета может привести к снижению производительности. Поэтому перед реализацией клиентского приложения рекомендуется выполнить тестирование с целью определить оптимальный размер пакета.

При пакетной вставке тип поля в объекте DataTable должен соответствовать типу поля в таблице СУБД ЛИНТЕР. Допустимые типы приведены в таблице 28. Если при конвертации данных происходит ошибка, то метод DbDataAdapter.Update() генерирует исключение. Если конвертация не выполняется, то пакетная вставка работает быстрее.

Таблица 28. Соответствие типов данных при пакетной вставке

Тип колонки в объекте DataTable	Допустимый тип СУБД ЛИНТЕР	Конвертации данных
System.Int16 или System.SByte	SMALLINT	Конвертация не выполняется
System.Int16 или System.SByte	INT	Конвертация выполняется
System.Int16 или System.SByte	BIGINT	Конвертация выполняется
System.Int32 или System.UInt16	SMALLINT	Конвертация выполняется и генерируется исключение типа OverflowException если значение превышает допустимые пределы
System.Int32 или System.UInt16	INT	Конвертация не выполняется
System.Int32 или System.UInt16	BIGINT	Конвертация выполняется
System.Int64 или System.UInt32	SMALLINT	Конвертация выполняется и генерируется исключение типа OverflowException если значение превышает допустимые пределы
System.Int64 или System.UInt32	INT	Конвертация выполняется и генерируется исключение типа OverflowException если значение превышает допустимые пределы

Тип колонки в объекте DataTable	Допустимый тип СУБД ЛИНТЕР	Конвертации данных
System.Int64 или System.UInt32	BIGINT	Конвертация не выполняется
System.Single	REAL	Конвертация не выполняется
System.Single	DOUBLE	Конвертация выполняется
System.Double	REAL	Конвертация выполняется и генерируется исключение типа Exception если значение превышает допустимые пределы
System.Double	DOUBLE	Конвертация не выполняется
System.Decimal или System.UInt64	DECIMAL	Конвертация не выполняется
System.Decimal или System.UInt64	REAL	Конвертация выполняется и генерируется исключение типа Exception если значение превышает допустимые пределы
System.Decimal или System.UInt64	DOUBLE	Конвертация выполняется
System.DateTime	DATE	Конвертация не выполняется
System.Byte	BYTE(1)	Конвертация не выполняется
System.Byte	VARBYTE(1)	Конвертация не выполняется
System.Byte[]	BYTE(<длина>)	Конвертация не выполняется
System.Byte[]	VARBYTE(<длина>)	Конвертация не выполняется
System.Boolean	BOOLEAN	Конвертация не выполняется
System.Char	NCHAR(1)	Конвертация не выполняется
System.Char	NVARCHAR(1)	Конвертация не выполняется
System.Char	CHAR(1)	Конвертация выполняется
System.Char	VARCHAR(1)	Конвертация выполняется
System.String	NCHAR(<длина>)	Конвертация не выполняется
System.String	NVARCHAR(<длина>)	Конвертация не выполняется
System.String	CHAR(<длина>)	Конвертация выполняется
System.String	VARCHAR(<длина>)	Конвертация выполняется
System.Guid	BYTE(16)	Конвертация не выполняется

Значение по умолчанию 1.

Примечание
В текущей версии ADO.NET-провайдера пакетная обработка данных поддерживается только для вставки записей (INSERT). Обновление (UPDATE) и удаление (DELETE) записей выполняется отдельно для каждой записи. Также отдельно обрабатываются записи, содержащие BLOB-данные и команды, использующие хранимые процедуры.

Декларация

```
public virtual int UpdateBatchSize {get; set;};
```

Значение свойства

Количество строк, которые необходимо обработать с помощью одного пакета:

- 0 – включает пакетный режим и устанавливает неограниченный размер пакета обновлений;
- 1 – отключает режим пакетного обновления (значение по умолчанию);
- больше 1 – включает пакетный режим и устанавливает размерность пакета обновлений. В этом случае во всех командах, связанных с объектом `DbDataAdapter`, для свойства `UpdatedRowSource` должно быть установлено значение None или OutputParameters. В противном случае будет выдаваться исключение.

Исключения

`ArgumentOutOfRangeException`	Размерность пакета обновления меньше 0.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class UpdateBatchSizeSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText =
      "create or replace table users ( " +
      "id integer primary key, name varchar(70))";
    con.Open();
    cmd.ExecuteNonQuery();
    con.Close();
    // Создание команды для выборки записей
    DbCommand selectCommand = factory.CreateCommand();
    selectCommand.Connection = con;
    selectCommand.CommandText =
      "select id, name from users";
    // Создание параметров для добавления записей
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    DbParameter parId = factory.CreateParameter();
    parId.ParameterName = ":id";
    parId.SourceColumn = "id";
    parId.Direction = ParameterDirection.Input;
    parId.DbType = DbType.Int32;
    DbParameter parName = factory.CreateParameter();
    parName.ParameterName = ":name";
    parName.SourceColumn = "name";
    parName.Direction = ParameterDirection.Input;
    parName.DbType = DbType.String;
    parName.Size = 70;
    // Создание команды для добавления записи
    DbCommand insertCommand = factory.CreateCommand();
    insertCommand.Connection = con;
    insertCommand.CommandText =
      "insert into users (id,name) values (:id,:name)";
    insertCommand.Parameters.Add(parId);
    insertCommand.Parameters.Add(parName);
    insertCommand.UpdatedRowSource = UpdateRowSource.None;
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = selectCommand;
    adapter.InsertCommand = insertCommand;
    adapter.UpdateBatchSize = 2;
    // Заполнение объекта DataTable данными из таблицы БД
    DataTable users = new DataTable();
    adapter.Fill(users);
    // Добавление записей в таблицу DataTable
    users.Rows.Add(0, "Пользователь A");
    users.Rows.Add(1, "Пользователь B");
    // Обновление БД
    int recordsAffected = adapter.Update(users);
    Console.WriteLine("Обработано строк: " + recordsAffected);
  }
}
```

UpdateCommand

Свойство предоставляет или устанавливает текст команды (SQL-запроса), используемой для обновления записей в источнике данных.

При использовании метода `Update` в случае, когда это свойство не задано и данные первичного ключа имеются в объекте `DataSet`, свойство UpdateCommand будет создаваться автоматически, если объект `DbDataAdapter` связан с объектом `DbCommandBuilder`.

Декларация

```
[BrowsableAttribute(false)]
```

```
public DbCommand UpdateCommand {get; set;};
```

Значение свойства

Интерфейс IDbCommand, используемый во время применения метода `Update` для обновления записей в источнике данных, соответствующих измененным строкам в наборе данных.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class UpdateCommandSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText =
      "create or replace table users ( " +
      "id integer primary key, name varchar(70));" +
      "insert into users (id, name) values (0, 'Пользователь A');"
 +
      "insert into users (id, name) values (1, 'Пользователь
 B');";
    cmd.ExecuteNonQuery();
    // Создание команды для выборки записей
    DbCommand selectCommand = factory.CreateCommand();
    selectCommand.Connection = con;
    selectCommand.CommandText =
      "select id, name from users";
    // Создание параметров для обновления записей
    DbParameter parOldId = factory.CreateParameter();
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    parOldId.ParameterName = ":oldId";
    parOldId.SourceColumn = "id";
    parOldId.Direction = ParameterDirection.Input;
    parOldId.DbType = DbType.Int32;
    parOldId.SourceVersion = DataRowVersion.Original;
    DbParameter parId = factory.CreateParameter();
    parId.ParameterName = ":id";
    parId.SourceColumn = "id";
    parId.Direction = ParameterDirection.Input;
    parId.DbType = DbType.Int32;
    parId.SourceVersion = DataRowVersion.Current;
    DbParameter parName = factory.CreateParameter();
    parName.ParameterName = ":name";
    parName.SourceColumn = "name";
    parName.Direction = ParameterDirection.Input;
    parName.DbType = DbType.String;
    parName.Size = 70;
    // Создание команды для обновления записи
    DbCommand updateCommand = factory.CreateCommand();
    updateCommand.Connection = con;
    updateCommand.CommandText =
      "update users set id = :id, name = :name where id = :oldId";
    updateCommand.Parameters.Add(parId);
    updateCommand.Parameters.Add(parName);
    updateCommand.Parameters.Add(parOldId);
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = selectCommand;
    adapter.UpdateCommand = updateCommand;
    // Заполнение объекта DataTable данными из таблицы БД
    DataTable users = new DataTable();
    adapter.Fill(users);
    // Изменение записей в таблице DataTable
    users.Rows[0]["name"] = "Новый Пользователь A";
    users.Rows[1]["name"] = "Новый Пользователь B";
    // Обновление БД
    adapter.Update(users);
    // Отображение таблицы после обновления
    Console.WriteLine("Строки таблицы после обновления:");
    foreach (DataRow row in users.Rows)
    {
      Console.WriteLine("{0}: '{1}' ", row[0], row[1]);
    }
    // Освобождение ресурсов
    con.Close();
  }
```

```
}
```

Методы

Fill(DataSet)

Метод загружает данные из источника данных в объект `DataSet`.

Метод `Fill` получает данные из источника данных с использованием оператора SELECT. Объект `IDbConnection`, связанный с командой SELECT, должен быть допустимым, но открывать его не требуется. Если интерфейс `IDbConnection` был закрыт до вызова метода `Fill`, он автоматически открывается для извлечения данных, а затем закрывается. Если подключение было открыто до вызова метода `Fill`, оно остается открытым.

Если во время заполнения таблицы данными происходит ошибка или исключение, то дальнейшее заполнение таблицы прекращается (при этом ранее загруженные данные остаются в таблице).

Если команда не возвращает строк, в объект `DataSet` никакая таблица не добавляется и исключение не происходит.

Если объект `DbDataAdapter` обнаруживает дублированные столбцы при заполнении объекта `DataTable`, он создает имена для последующих столбцов, используя шаблон «имя столбца»1, «имя столбца»2, «имя столбца»3 и т.д. Если входные данные содержат неименованные столбцы, они помещаются в объект `DataSet` в соответствии с шаблоном «столбец1», «столбец2» и т.д.

Если запрос возвращает несколько результатов, то набор результатов для каждого запроса, возвращающего строку, помещается в отдельную таблицу. Дополнительным наборам результатов присваиваются имена с добавлением целого числа к заданному имени таблицы, например «таблица»1, «таблица»2 и т.д. Так как для запросов, не возвращающих строк, таблица не создается, то в случаях, когда обрабатывается запрос INSERT, а затем запрос SELECT, таблица, созданная для запроса SELECT, будет иметь имя «таблица», поскольку это первая созданная таблица. Приложения, использующие имена столбцов и таблиц, должны обеспечивать отсутствие конфликтов шаблонов именования.

Когда оператор SELECT, используемый для заполнения объекта `DataSet`, возвращает несколько результатов, например, пакетные операторы SQL, то, если один из результатов содержит ошибку, все последующие результаты пропускаются и не добавляются в объект `DataSet`.

При использовании последующих вызовов метода `Fill` для обновления содержимого объекта `DataSet` должны быть соблюдены два условия:

1) оператор SQL должен соответствовать инструкции, первоначально использованной для заполнения объекта `DataSet`;
2) должны быть представлены сведения о столбце Key.

Если имеются данные первичного ключа, любые дублирующие строки согласовываются и отображаются только один раз в объекте `DataTable`, который соответствует объекту `DataSet`. Данные первичного ключа можно задать либо с помощью метода `FillSchema`, указав свойство PrimaryKey объекта `DataTable`, либо задав для свойства MissingSchemaAction значение AddWithKey.

Если свойство SelectCommand возвращает результаты соединения таблиц (OUTER JOIN), то объект `DataAdapter` не устанавливает значение свойства PrimaryKey для результирующего объекта `DataTable`. Необходимо явно определить первичный ключ, чтобы убедиться в том, что обработка дубликатов строк выполняется правильно.

Примечание
При обработке пакетных SQL-операторов, возвращающих несколько результатов, метод `FillSchema` ADO.NET-провайдера СУБД ЛИНТЕР получает сведения схемы только для первого результата. Чтобы извлечь информацию схемы для остальных результатов, необходимо использовать метод `Fill` со значением AddWithKey свойства MissingSchemaAction.

Синтаксис

```
public override int Fill( DataSet dataSet);
```

`dataSet` – объект `DataSet`, который должен быть заполнен данными и, при необходимости, метаданными (схемой данных).

Возвращаемое значение

Количество строк, реально добавленных или обновленных в объекте `DataSet`.

Исключения

`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Примеры

1) Код, показанный в данном примере, не открывает и не закрывает Connection явным образом. Если соединение еще не открыто, то метод `Fill` неявно открывает Connection, которое используется `DataAdapter`. Если операция `Fill` открыла соединение, она также закрывает его при завершении `Fill`. Это позволяет упростить код при использовании отдельной операции, такой, как `Fill` или `Update`. Однако при выполнении нескольких операций, требующих открытого соединения, можно повысить производительность приложения путем явного вызова метода `Open` объекта `Connection`, выполнения операций с источником данных и последующего вызова метода `Close` объекта `Connection`. Необходимо сохранять соединения с источником данных лишь на такое короткое время, насколько это возможно, освобождая ресурсы для использования другими клиентскими приложениями.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class FillDataSetSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
```

```
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = factory.CreateCommand();
    adapter.SelectCommand.Connection = con;
    adapter.SelectCommand.CommandText =
      "select personid, model from auto";
    // Создание объекта DataSet
    DataSet dataset = new DataSet();
    // Заполнение объекта DataSet данными из таблицы БД
    adapter.Fill(dataset);
    // Отображение полученных данных
    foreach (DataTable table in dataset.Tables)
    {
      Console.WriteLine("Имя таблицы: " + table.TableName);
      Console.WriteLine("Строки таблицы:");
      foreach (DataRow row in table.Rows)
      {
        foreach (DataColumn column in table.Columns)
        {
          Console.Write("{0} | ", row[column.ColumnName]);
        }
        Console.WriteLine();
      }
    }
  }
}
```

Каждый объект `DataSet` содержит коллекцию из одного или более объектов `DataTable`. Каждый объект `DataTable` соответствует одной таблице. С помощью свойства SelectCommand, в котором содержится операция соединения, можно производить выборку из нескольких таблиц БД в один объект `DataTable`. При необходимости обновить содержимое множественных таблиц, достаточно определить лишь команду обновления, так как информация о связях между таблицами БД уже известна.

В этом примере `DataSet` содержит один объект `DataTable`, представляющий таблицу Motorist («Автовладельцы»).

```
// C#
using System;
using System.Data;
using System.Data.Common;

class FillDataSetSample
{
  static void Main()
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = factory.CreateCommand();
    adapter.SelectCommand.Connection = con;
    adapter.SelectCommand.CommandText =
      "select auto.personid, auto.model, auto.color, person.city "
+
      "from auto, person " +
      "where auto.personid=person.personid and year=70 limit
 50,10";
    // Создание объекта DataSet
    DataSet dataset = new DataSet();
    // Заполнение объекта DataSet данными из таблицы БД
    adapter.Fill(dataset, "Motorist");
    // Отображение полученных данных
    foreach (DataTable table in dataset.Tables)
    {
      Console.WriteLine("Имя таблицы: " + table.TableName);
      Console.WriteLine("Строки таблицы:");
      foreach (DataRow row in table.Rows)
      {
        foreach (DataColumn column in table.Columns)
        {
          Console.Write("{0} | ", row[column.ColumnName]);
        }
        Console.WriteLine();
      }
    }
  }
}
```

Fill(DataTable)

Метод `Fill(DataTable)` получает строки из источника данных с помощью оператора SELECT, указанного в свойстве SelectCommand. Объект соединения, связанный с оператором SELECT, должен быть допустимым, но открывать его не требуется. Если соединение с ЛИНТЕР-сервером было закрыто до вызова метода `Fill(DataTable)`, то оно автоматически открывается для извлечения данных, а затем также автоматически закрывается. Если соединение было открыто до вызова метода `Fill(DataTable)`, то оно остается открытым.

Метод Fill(DataTable) добавляет строки в целевые объекты DataTable в DataSet, создавая объекты DataTable, если они еще не существуют. При создании объектов DataTable метод Fill(DataTable) обычно загружает метаданные только об имени столбца. Однако если свойство MissingSchemaAction имеет значение AddWithKey, то также загружаются метаданные о первичных ключах и ограничениях целостности.

Если объект DbDataAdapter обнаруживает дублированные столбцы при заполнении объекта DataTable, то он создает имена для последующих столбцов, используя шаблон «имя столбца»1, «имя столбца»2, «имя столбца»3 и т.д., где «имя столбца» – имя дублируемого столбца. Если входные данные содержат неименованные столбцы, они помещаются в объект DataSet в соответствии с шаблоном «столбец1», «столбец2» и т.д. При добавлении нескольких наборов результатов в объект DataSet каждый из них помещается в отдельную таблицу.

Метод Fill(DataTable) может использоваться несколько раз для одного и того же объекта DataTable. Если существует первичный ключ, входящие строки объединяются с соответствующими строками, которые уже существуют.

Если первичный ключ отсутствует, входящие строки добавляются в конец объекта DataTable.

Если свойство SelectCommand возвращает результаты соединения таблиц (OUTER JOIN), то объект DataAdapter не задает значение свойства PrimaryKey для результирующего объекта DataTable. Необходимо явно определить первичный ключ, чтобы убедиться в том, что дублированные строки разрешены правильно.

Примечание
Если обрабатываемый пакет SQL-операторов возвращает несколько наборов результатов, то метод Fill(DataTable) и FillSchema ADO.NET-провайдера СУБД ЛИНТЕР извлекает сведения о схеме только для первого набора результатов.

Синтаксис

```
public int Fill(DataTable dataTable);
```

dataTable – объект DataTable, в который необходимо загрузить данные.

Возвращаемое значение

Количество строк, реально добавленных или обновленных в объекте DataSet.

Исключения

LinterSqlException	Код завершения СУБД ЛИНТЕР не равен 0.

Примеры

Метод Fill поддерживает сценарии, где объект DataSet содержит несколько объектов DataTable, имена которых отличаются только регистром. В таких ситуациях метод Fill выполняет сравнение с учетом регистра для поиска соответствующей таблицы и создает новую таблицу, если не существует полного соответствия.

Если при вызове метода Fill в объекте DataSet содержит только один объект DataTable, имя которого отличается только регистром, этот объект DataTable обновляется.

1) В этом примере создается новая таблица, т.к. в DataSet уже есть несколько таблиц с похожими именами и трудно выбрать, какая из них может быть обновлена (т.е. регистр учитывается)

```
DataSet dataset = new DataSet();
 dataset.Tables.Add("aaa");
 dataset.Tables.Add("AAA");
//Заполнение таблицы "aaa", которая уже существует в DataSet.
 adapter.Fill(dataset, "aaa");
// Создание в DataSet новой таблицы с последующим заполнением её
 данными
 adapter.Fill(dataset, "Aaa");
```

2) В этом примере новая таблица не создается, т.к. в DataSet есть только одна таблица с похожим именем и проблемы с выбором таблицы обновления нет (т.е. регистр не учитывается)

```
DataSet dataset = new DataSet();
dataset.Tables.Add("aaa");
// Заполняется таблица "aaa", потому что в DataSet есть только
 одна таблица с похожим именем
adapter.Fill(dataset, "AAA");
```

Fill(DataSet, String)

Метод добавляет или обновляет строки в указанной таблице объекта `DataSet` для получения соответствия строкам, полученный из источника данных.

Описание работы метода см. в пункте Fill(DataTable).

Синтаксис

```
public int Fill(DataSet dataSet,string srcTable);
```

`dataSet` – объект `DataSet`, который должен быть заполнен данными и, при необходимости, метаданными (схемой данных).

`srcTable` – имя таблицы в источнике данных, используемой для загрузки данных, изменения которой будут переданы на сервер для обновления базы данных.

Возвращаемое значение

Количество строк, реально добавленных или обновленных в объекте `DataSet`.

Исключения

`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

```
// C#
using System;
using System.Data;
```

```
using System.Data.Common;

class FillDataSetSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = factory.CreateCommand();
    adapter.SelectCommand.Connection = con;
    adapter.SelectCommand.CommandText =
      "select personid, model from auto";
    // Создание объекта DataSet
    DataSet dataset = new DataSet();
    // Заполнение объекта DataSet данными из таблицы БД
    adapter.Fill(dataset, "Auto");
    // Отображение полученных данных
    foreach (DataTable table in dataset.Tables)
    {
      Console.WriteLine("Имя таблицы: " + table.TableName);
      Console.WriteLine("Строки таблицы:");
      foreach (DataRow row in table.Rows)
      {
        foreach (DataColumn column in table.Columns)
        {
          Console.Write("{0} | ", row[column.ColumnName]);
        }
        Console.WriteLine();
      }
    }
  }
}
```

Fill(Int32, Int32, DataTable)

Метод реализует добавление или обновление строк в объекте `DataTable` из части таблицы в источнике данных.

Например, таблица X содержит 1 млн. записей. Данный метод позволяет выполнить частичную загрузку этой таблицы в `DataTable`, например, 100 записей, начиная с 500001.

Примечание
В клиентских приложениях, ориентированных на работу исключительно с СУБД ЛИНТЕР, вместо данного метода можно использовать конструкцию «ограничение выборки» select-запроса:

```
<ограничение выборки>::=
FETCH FIRST <объем выборки> [PERCENT] [WITH TIES]
| LIMIT  [<начало выборки>,] <количество строк>
```

Описание работы метода см. в пункте Fill(DataTable).

Синтаксис

```
  public int Fill(
    int startRecord,
    int maxRecords,
    params DataTable[] dataTables
);
```

`startRecord` – начальный номер строки в источнике данных, с которой необходимо начать добавление (обновление) строк в DataTable.

Отсчет начинается с `0`.

`maxRecords` – максимальное число извлекаемых из источника данных строк.

Если значение maxRecords равно 0, то извлекаются все строки, найденные после начальной строки. Если maxRecords больше, чем число оставшихся строк, возвращаются только оставшиеся строки и ошибка не выдается.

`dataTables` – массив объектов `DataTable`, который необходимо заполнить данными.

При обработке пакетных запросов данные, полученные каждым запросом, загружаются в отдельный объект `DataTable`. Если методу передано недостаточно объектов `DataTable`, то получение данных завершается после заполнения всех объектов `DataTable` и ошибка не возникает. Если указаны ненулевые значения параметров startRecord или maxRecords, то в массиве dataTables должен быть только один элемент.

Возвращаемое значение

Количество строк, реально добавленных или обновленных в объекте `DataTable`.

Примечание
Хотя вся выборка данных, создаваемая SQL-запросом, передается с ЛИНТЕР-сервера на клиентский компьютер, тем не менее, объект `DataTable` не будет содержать больше записей, чем указано в maxRecords.

Исключения

`ArgumentNullException`	Параметр dataTables содержит null-значение.

ArgumentException	Значение параметра startRecord или параметра maxRecords меньше 0.
InvalidOperationException	Свойство SelectCommand не инициализировано или массив dataTables частично заполнен при ненулевых значениях параметра startRecord или maxRecords.
LinterSqlException	Код завершения СУБД ЛИНТЕР не равен 0.

Пример

Загрузить в DataTable семь записей выборки из двух таблиц, начиная с 25.

1) Стандартный метод.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class FillSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = factory.CreateCommand();
    adapter.SelectCommand.Connection = con;
    adapter.SelectCommand.CommandText =
      "select model, make " +
      "from auto, person " +
      "where auto.personid=person.personid and name like 'A%'";
    // Создание объекта DataTable
    DataTable table = new DataTable();
    // Заполнение объекта DataTable данными из таблицы БД
    adapter.Fill(25, 7, table);
    // Отображение полученных данных
    Console.WriteLine("Строки таблицы:");
    foreach (DataRow row in table.Rows)
    {
      foreach (DataColumn column in table.Columns)
      {
        Console.Write("{0} | ", row[column.ColumnName]);
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
      }
      Console.WriteLine();
    }
  }
}
```

2) Только при работе с ЛИНТЕР-сервером.

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class FillSample
{
  static void Main()
  {
    // Создание объекта LinterDbDataAdapter
    LinterDbDataAdapter adapter = new LinterDbDataAdapter(
      "select model, make from auto, person " +
      "where auto.personid=person.personid and name like 'A%'
 limit 25, 7",
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8");
    // Создание объекта DataTable
    DataTable table = new DataTable();
    // Заполнение объекта DataTable данными из таблицы БД
    adapter.Fill(table);
    // Отображение полученных данных
    Console.WriteLine("Строки таблицы:");
    foreach (DataRow row in table.Rows)
    {
      foreach (DataColumn column in table.Columns)
      {
        Console.Write("{0} | ", row[column.ColumnName]);
      }
      Console.WriteLine();
    }
  }
}
```

Fill(DataSet, Int32, Int32, String)

Метод реализует добавление или обновление строк в объект `DataSet` из части таблицы в источнике данных.

Например, таблица X содержит 1 млн. записей. Данный метод позволяет выполнить частичную загрузку этой таблицы в `DataSet`, например, 100 записей, начиная с 500001.

Примечание
В клиентских приложениях, ориентированных на работу исключительно с СУБД ЛИНТЕР, вместо данного метода можно использовать конструкцию «ограничение выборки» select-запроса:

```
<ограничение выборки>::=
FETCH FIRST <объем выборки> [PERCENT] [WITH TIES]
| LIMIT  [ <начало выборки>,] <количество строк>
```

Описание работы метода см. в пункте Fill(DataTable).

Синтаксис

```
  public int Fill(
    DataSet dataSet,
    int startRecord,
    int maxRecords,
    string srcTable
);
```

`dataSet` – объект `DataSet` для заполнения записями и, если необходимо, схемой данных.

`startRecord` – начальный номер строки в источнике данных, с которой необходимо начать добавление (обновление) строк в `DataSet`.

Отсчет начинается с `0`.

`maxRecords` – максимальное число извлекаемых из источника данных строк.

Если значение maxRecords равно нулю, то извлекаются все строки, найденные после начальной строки. Если значение maxRecords больше, чем число оставшихся строк, возвращаются только оставшиеся строки и ошибка не выдается.

`srcTable` – имя таблицы в источнике данных, используемой для загрузки записей.

Возвращаемое значение

Количество строк, реально добавленных или обновленных в объекте `DataSet`.

Примечание
Хотя вся выборка данных, создаваемая SQL-запросом, передается с ЛИНТЕР-сервера на клиентский компьютер, тем не менее объект `DataSet` не будет содержать больше записей, чем указано в maxRecords.

Исключения

`ArgumentNullException`	Параметр dataSet содержит null-значение.
`ArgumentException`	Значение параметра startRecord или параметра maxRecords меньше 0.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

InvalidOperationException	Свойство SelectCommand не инициализировано.
LinterSqlException	Код завершения СУБД ЛИНТЕР не равен 0.

Пример

Загрузить в DataSet семь записей выборки из двух таблиц, начиная с 25.

1) Стандартный метод.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class FillSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = factory.CreateCommand();
    adapter.SelectCommand.Connection = con;
    adapter.SelectCommand.CommandText =
      "select model, make " +
      "from auto, person " +
      "where auto.personid=person.personid and name like 'A%'";
    // Создание объекта DataSet
    DataSet dataset = new DataSet();
    // Заполнение объекта DataSet данными из таблицы БД
    adapter.Fill(dataset, 25, 7, "Автомобили");
    // Отображение полученных данных
    foreach (DataTable table in dataset.Tables)
    {
      Console.WriteLine("Имя таблицы: " + table.TableName);
      Console.WriteLine("Строки таблицы:");
      foreach (DataRow row in table.Rows)
      {
        foreach (DataColumn column in table.Columns)
        {
          Console.Write("{0} | ", row[column.ColumnName]);
        }
```

```
        Console.WriteLine();
      }
    }
  }
}
```

2) Только при работе с ЛИНТЕР-сервером.

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class FillSample
{
  static void Main()
  {
    // Создание объекта LinterDbDataAdapter
    LinterDbDataAdapter adapter = new LinterDbDataAdapter(
      "select model, make from auto, person " +
      "where auto.personid=person.personid and name like 'A%'
 limit 25, 7",
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8");
    // Создание объекта DataSet
    DataSet dataset = new DataSet();
    // Заполнение объекта DataSet данными из таблицы БД
    adapter.Fill(dataset, "Автомобили");
    // Отображение полученных данных
    foreach (DataTable table in dataset.Tables)
    {
      Console.WriteLine("Имя таблицы: " + table.TableName);
      Console.WriteLine("Строки таблицы:");
      foreach (DataRow row in table.Rows)
      {
        foreach (DataColumn column in table.Columns)
        {
          Console.Write("{0} | ", row[column.ColumnName]);
        }
        Console.WriteLine();
      }
    }
  }
}
```

FillSchema(DataSet, SchemaType, String)

Метод реализует добавление или обновление схемы DataSet в соответствие со схемой в источнике данных.

Этот метод получает сведения схемы из источника данных с использованием свойства SelectCommand.

Операция `FillSchema` добавляет объект `DataTable` к указанному `DataSet`. Затем столбцы добавляются в объект `DataColumnCollection` объекта `DataTable` и настраиваются следующие свойства `DataColumn`, если они существуют в источнике данных:

- AllowDBNull;
- AutoIncrement. Свойства AutoIncrementStep и AutoIncrementSeed нужно задать отдельно;
- MaxLength;
- ReadOnly;
- Unique.

Метод `FillSchema` также настраивает свойства PrimaryKey и Constraints в соответствии со следующими правилами:

1) если один или несколько столбцов первичных ключей возвращаются свойством SelectCommand, то они используются в качестве столбцов первичных ключей для объекта `DataTable`;
2) если столбцы первичных ключей не загружаются, а загружаются уникальные столбцы, то эти уникальные столбцы используются как первичный ключ только в том случае, если все они не могут содержать null-значения. Если хотя бы один столбец допускает null-значения, к объекту `ConstraintCollection` добавляется объект `UniqueConstraint`, но свойство PrimaryKey не задается;
3) если возвращаются и столбцы первичных ключей, и уникальные столбцы, то столбцы первичных ключей используются в качестве столбцов первичных ключей для объекта `DataTable`.

Первичные ключи и уникальные ограничения добавляются к объекту `ConstraintCollection` в соответствии с предыдущими правилами, но другие типы ограничений не добавляются.

Сведения о первичном ключе используются во время применения метода `Fill` для поиска и замены строк, у которых столбцы ключей совпадают. Если это нежелательно, рекомендуется использовать метод `Fill` без запроса сведений о схеме.

Синтаксис

```
  public DataTable[] FillSchema(
    DataSet dataSet,
    SchemaType schemaType,
    string srcTable
);
```

`dataSet` – объект `DataSet` для заполнения схемой.

`schemaType` – одно из значений типа SchemaType, указывающее, как применять схему:

- `Source` – информация о схеме должна браться из источника данных;
- `Mapped` – к полям, возвращенным запросом, должна применяться схема из своей коллекции TableMappings.

srcTable – имя таблицы в источнике данных, используемой для загрузки записей.

Возвращаемое значение

Ссылка на коллекцию объектов DataTable, которые были добавлены в объект DataSet.

Исключения

ArgumentNullException	Параметр dataSet содержит null-значение.
ArgumentException	Параметр srcTable содержит null-значение или пустую строку.
InvalidOperationException	Свойство SelectCommand не инициализировано.
ArgumentOutOfRangeException	Значение параметра schemaType не является одним из значений SchemaType.
LinterSqlException	Код завершения СУБД ЛИНТЕР не равен 0.

Примеры

1) Пример загрузки схемы для SchemaType.Mapped.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class FillSchemaSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание объекта DataSet
    DataSet dataset = new DataSet();
    // Создание объекта DataTable
    DataTable table = dataset.Tables.Add("Автомобили");
    table.Columns.Add("Номер", typeof(int));
    table.Columns.Add("Производитель", typeof(string));
    table.Columns.Add("Модель", typeof(string));
    // Отображение таблицы БД на таблицу DataTable
    DataTableMapping mapping = new DataTableMapping("Table",
 "Автомобили");
    mapping.ColumnMappings.Add("PERSONID", "Номер");
    mapping.ColumnMappings.Add("MAKE", "Производитель");
```

```
    mapping.ColumnMappings.Add("MODEL", "Модель");
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = factory.CreateCommand();
    adapter.SelectCommand.Connection = con;
    adapter.SelectCommand.CommandText =
      "select personid, make, model from auto";
    adapter.TableMappings.Add(mapping);
    // Вывод на экран исходной схемы
    Console.WriteLine("Исходная схема DataSet:");
    OutputSchema(dataset);
    // Заполнение объекта DataSet схемой из БД
    adapter.FillSchema(dataset, SchemaType.Mapped, "Table");
    // Вывод на экран полученной схемы
    Console.WriteLine("Схема DataSet после загрузки из БД:");
    OutputSchema(dataset);
  }
  private static void OutputSchema(DataSet dataset)
  {
    foreach (DataTable table in dataset.Tables)
    {
      Console.WriteLine("Имя таблицы: " + table.TableName);
      Console.WriteLine("Столбцы таблицы:");
      Console.WriteLine("ColumnName    | AllowDBNull |
 AutoIncrement | " +
        "MaxLength | ReadOnly | Unique");
      foreach (DataColumn column in table.Columns)
      {
        Console.WriteLine(
          "{0,-13} | {1,-11} | {2,-13} | {3,-9} | {4,-8} |
 {5,-6}",
          column.ColumnName, column.AllowDBNull,
 column.AutoIncrement,
          column.MaxLength, column.ReadOnly, column.Unique);
      }
    }
  }
}
```

Результат выполнения примера:

```
Исходная схема DataSet:
Имя таблицы: Автомобили
Столбцы таблицы:
ColumnName    | AllowDBNull | AutoIncrement | MaxLength | ReadOnly
 | Unique
```

```
Номер          | True        | False          | -1         | False
 | False
Производитель | True        | False          | -1         | False
 | False
Модель         | True        | False          | -1         | False
 | False
Схема DataSet после загрузки из БД:
Имя таблицы: Автомобили
Столбцы таблицы:
ColumnName     | AllowDBNull | AutoIncrement | MaxLength | ReadOnly
 | Unique
Номер          | False       | False          | -1         | False
 | True
Производитель | True        | False          | 20         | False
 | False
Модель         | True        | False          | 20         | False
 | False
```

2) Пример загрузки схемы для SchemaType.Source.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class FillSchemaSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = factory.CreateCommand();
    adapter.SelectCommand.Connection = con;
    adapter.SelectCommand.CommandText =
      "select personid, make, model from auto";
    // Создание объекта DataSet
    DataSet dataset = new DataSet();
    // Заполнение объекта DataSet схемой из БД
    adapter.FillSchema(dataset, SchemaType.Source, "Table");
    // Вывод на экран полученной схемы
```

```
    Console.WriteLine("Схема DataSet после загрузки из БД:");
    OutputSchema(dataset);
  }
  private static void OutputSchema(DataSet dataset)
  {
    foreach (DataTable table in dataset.Tables)
    {
      Console.WriteLine("Имя таблицы: " + table.TableName);
      Console.WriteLine("Столбцы таблицы:");
      Console.WriteLine("ColumnName    | AllowDBNull |
 AutoIncrement | " +
        "MaxLength | ReadOnly | Unique");
      foreach (DataColumn column in table.Columns)
      {
        Console.WriteLine(
          "{0,-13} | {1,-11} | {2,-13} | {3,-9} | {4,-8} |
 {5,-6}",
          column.ColumnName, column.AllowDBNull,
 column.AutoIncrement,
          column.MaxLength, column.ReadOnly, column.Unique);
      }
    }
  }
}

Результат выполнения примера:

Схема DataSet после загрузки из БД:
Имя таблицы: Table
Столбцы таблицы:
ColumnName    | AllowDBNull | AutoIncrement | MaxLength | ReadOnly
 | Unique
PERSONID      | False       | False         | -1        | False
 | True
MAKE          | True        | False         | 20        | False
 | False
MODEL         | True        | False         | 20        | False
 | False
```

FillSchema(DataSet, SchemaType)

Метод добавляет объект `DataTable` с именем "Table" в объект `DataSet` и настраивает его схему в соответствии с источником данных.

Описание работы метода см. в пункте FillSchema(DataSet, SchemaType, String).

Синтаксис

```
public override DataTable[] FillSchema(
    DataSet dataSet,
    SchemaType schemaType
);
```

`dataSet` – объект `DataSet` для заполнения схемой.

`schemaType` – одно из значений типа SchemaType, указывающее, как применять схему:

- `Source` – информация о схеме должна браться из источника данных;
- `Mapped` – к полям, возвращенным запросом, должна применяться схема из своей коллекции TableMappings.

Возвращаемое значение

Ссылка на коллекцию объектов `DataTable`, которые были добавлены в объект `DataSet`.

Исключения

`ArgumentNullException`	Параметр dataSet содержит null-значение.
`ArgumentException`	Параметр `schemaType` содержит null-значение или пустую строку.
`InvalidOperationException`	Свойство SelectCommand не инициализировано.
`ArgumentOutOfRangeException`	Значение параметра schemaType не является одним из значений SchemaType.
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

См. пример в пункте FillSchema(DataSet, SchemaType, String).

FillSchema(DataTable, SchemaType)

Метод настраивает схему данных для указанного объекта `DataTable`.

Описание работы метода см. в пункте FillSchema(DataSet, SchemaType, String).

Синтаксис

```
public DataTable FillSchema(
    DataTable dataTable,
    SchemaType schemaType
);
```


© Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

`dataTable` – объект `DataTable`, который необходимо заполнить сведениями схемы из источника данных.

`schemaType` – одно из значений типа SchemaType, указывающее, как применять схему:

- `Source` – все сопоставления таблиц в DataAdapter будут игнорироваться. При конфигурировании DataSet будет использоваться не преобразованная входящая схема;
- `Mapped` – все существующие сопоставления таблиц будут применяться к входящей схеме. При конфигурировании DataSet будет использоваться преобразованная схема.

Возвращаемое значение

Объект `DataTable`, который содержит информацию о схеме.

Исключения

`ArgumentNullException`	Параметр dataTable содержит null-значение.
`ArgumentException`	Параметр `schemaType` содержит null-значение или пустую строку.
`InvalidOperationException`	Свойство SelectCommand не инициализировано.
`ArgumentOutOfRangeException`	Значение параметра schemaType не является одним из значений SchemaType.
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

См. пример в пункте FillSchema(DataSet, SchemaType, String).

GetFillParameters

Метод предоставляет информацию о параметрах параметризованного SELECT-оператора в виде массива объектов `IDataParameter` (а не объектов `DbParameter` ADO.NET-провайдера СУБД ЛИНТЕР).

Если не нужно проверять или задавать значения свойств Size, Precision и Scale, то с помощью данного метода можно присваивать значения параметрам.

Синтаксис

```
public override IDataParameter[] GetFillParameters();
```

Возвращаемое значение

Массив объектов `IDataParameter`, соответствующих SQL-оператору, указанному в свойстве SelectCommand.

Если объект `DataAdapter` каким-либо образом не инициализирован (например, свойство SelectCommand равно null), то исключение не генерируется, а возвращается массив объектов `IDataParameter`, не содержащий элементов, т.е. нулевой длины.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetFillParametersSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание параметров для команды
    DbParameter par = factory.CreateParameter();
    par.ParameterName = ":car";
    par.Direction = ParameterDirection.Input;
    par.DbType = DbType.String;
    par.Size = 20;
    par.Value = "CHEVROLET%";
    // Создание команды для выборки записей
    DbCommand selectCommand = factory.CreateCommand();
    selectCommand.Connection = con;
    selectCommand.CommandText =
      "select personid, model from auto where model like :car";
    selectCommand.Parameters.Add(par);
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = selectCommand;
    // Заполнение объекта DataTable данными из таблицы БД
    DataTable table = new DataTable();
    adapter.Fill(table);
    // Отображение полученных данных
    Console.WriteLine("Строки таблицы:");
    foreach (DataRow row in table.Rows)
    {
      foreach (DataColumn column in table.Columns)
      {
        Console.Write("{0} | ", row[column.ColumnName]);
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
      }
      Console.WriteLine();
    }
    // Отображение параметров
    Console.WriteLine("Параметры:");
    foreach (IDataParameter p in adapter.GetFillParameters())
    {
      Console.WriteLine("Имя параметра: " + p.ParameterName);
      Console.WriteLine("Значение параметра: " + p.Value);
    }
  }
}
```

Update(DataRow[])

Метод выполняет необходимые операторы (INSERT, UPDATE или DELETE) для изменения строк в указанном массиве объектов `DataRow`.

Указанные операторы должны быть либо явно прописаны в соответствующих свойствах объекта `DbDataAdapter` (InsertCommand, UpdateCommand, DeleteCommand), либо сконструированы с помощью класса `DbCommandBuilder`.

Когда клиентское приложение вызывает метод `Update`, объект `DbDataAdapter` проверяет свойство RowState и выполняет необходимые операторы INSERT, UPDATE или DELETE для каждой строки объекта `DataSet` на основе заданной очередности выполнения операций. Можно использовать метод `Select` объекта `DataTable` для возврата массива DataRow, который ссылается только на строки с конкретным значением RowState. После этого можно передать возвращенный массив DataRow в метод `Update` объекта `DataAdapter` для обработки измененных строк. Задавая подмножество строк, подлежащих обновлению, можно управлять тем, в какой последовательности обрабатываются вставки, обновления и удаления. Например, метод `Update` может сначала выполнить оператор DELETE, затем оператор INSERT, а затем – еще один оператор DELETE, что определяется порядком строк в объекте `DataTable`.

Операторы манипулирования данными выполняются не как пакетный процесс. Каждая строка обновляется индивидуально. Клиентское приложение может вызвать метод `Select` в случаях, когда необходимо управлять последовательностью типов операторов (например, когда оператор INSERT выполняется до выполнения оператора UPDATE).

Если операторы INSERT, UPDATE или DELETE не были указаны, метод `Update` создает исключение. Однако можно создать объект `DbCommandBuilder` для автоматического создания SQL-операторов для однотабличных обновлений, если задано свойство SelectCommand объекта `DbDataAdapter`. Затем любые дополнительные SQL-операторы, которые не были ранее заданы, создаются объектом `CommandBuilder`. Такая логика создания операторов требует, чтобы сведения о столбце с первичным ключом присутствовали в объекте `DataSet`.

Метод `Update` получает строки из таблицы, указанной в первом сопоставлении перед выполнением обновления. Затем метод `Update` обновляет строку, используя значение свойства UpdatedRowSource. Все возвращенные дополнительные строки игнорируются.

После того, как данные загружены снова в объект `DataSet`, генерируется событие OnRowUpdated, предоставляя пользователю возможность проверить согласованную

строку DataSet и любые выходные параметры, возвращенные командой. После успешного обновления строки принимаются изменения этой строки.

При использовании метода Update порядок выполнения является следующим:

- значения в объекте DataRow переносятся в значения параметров;
- генерируется событие OnRowUpdating;
- выполняется команда;
- если для команды задано значение FirstReturnedRecord, первый возвращенный результат помещается в объект DataRow;
- при наличии выходных параметров они помещаются в объект DataRow;
- генерируется событие OnRowUpdated;
- вызывается метод AcceptChanges.

С каждой командой, связанной с классом DbDataAdapter, обычно связана коллекция параметров. Параметры сопоставляются с текущей строкой с помощью свойств SourceColumn и SourceVersion класса Parameter ADO.NET-провайдера СУБД ЛИНТЕР. Свойство SourceColumn ссылается на столбец объекта DataTable, на который есть ссылка в объекте DbDataAdapter, для получения значений параметров для текущей строки.

Свойство SourceColumn ссылается на несопоставленный столбец перед применением сопоставления таблиц. Если свойство SourceColumn ссылается на несуществующий столбец, выполняемые действия зависят от одного из следующих значений MissingMappingAction:

1) MissingMappingAction.Passthrough – использовать имена исходных столбцов и таблиц в объекте DataSet, если сопоставление отсутствует;
2) MissingMappingAction.Ignore – генерируется исключение SystemException. Если сопоставления заданы явным образом, отсутствие сопоставления для входного параметра обычно является следствием ошибки;
3) MissingMappingAction.Error – генерируется исключение SystemException.

Свойство SourceColumn также используется для сопоставления выходных значений или входных и выходных параметров обратно в объект DataSet. При ссылке на несуществующий столбец генерируется исключение.

Свойство SourceVersion класса Parameter ADO.NET-провайдера СУБД ЛИНТЕР определяет, следует ли использовать версию Original, Current или Proposed значения столбца. Эта возможность обычно используется для включения исходных значений в предложении WHERE оператора UPDATE для проверки нарушений параллелизма типа OPTIMISTIC.



Примечание

Если при обновлении строки возникает ошибка, выдается исключение, и обновление прерывается. Чтобы продолжить операцию обновления без создания исключений при обнаружении ошибки, необходимо установить для свойства ContinueUpdateOnError значение true перед вызовом метода Update. Также можно реагировать на ошибки на построчной основе в событии RowUpdated объекта DbDataAdapter. Чтобы продолжить операцию обновления без создания исключения в событии RowUpdated, необходимо установить для свойства Status объекта RowUpdatedEventArgs значение Continue.

Синтаксис

```
public int Update(DataRow[] dataRows);
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

dataRows – массив объектов DataRow, используемый для обновления источника данных.

Возвращаемое значение

Количество успешно обновленных строк из объекта DataSet.

Исключения

ArgumentNullException	Параметр dataSet содержит null-значение.
InvalidOperationException	Исходная таблица является недопустимой.
SystemException	Возможные причины: • не существует объект DataRow или DataTable для обновления; • не существует объект DataSet для использования в качестве источника.
DBConcurrencyException	Попытка выполнить оператор INSERT, UPDATE или DELETE привела к нулевому количеству обработанных записей.
LinterSqlException	Код завершения СУБД ЛИНТЕР не равен 0.

Пример

В примере демонстрируется первоочередная обработка удаленных строк таблицы, затем происходит обработка обновленных строк, после чего обрабатываются вставленные строки.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class UpdateSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText =
      "create or replace table users ( " +
      "id integer primary key, name varchar(70));" +
```

```
      "insert into users (id, name) values (0, 'Пользователь A');"
 +
      "insert into users (id, name) values (1, 'Пользователь
 B');";
    con.Open();
    cmd.ExecuteNonQuery();
    con.Close();
    // Создание команды для выборки записей
    DbCommand selectCommand = factory.CreateCommand();
    selectCommand.Connection = con;
    selectCommand.CommandText =
      "select id, name from users";
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = selectCommand;
    // Создание объекта DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Заполнение объекта DataTable данными из таблицы БД
    DataTable users = new DataTable();
    adapter.Fill(users);
    // Изменение записей в таблице DataTable
    users.Rows[0]["name"] = "Новый Пользователь A";
    users.Rows[1].Delete();
    users.Rows.Add(2, "Пользователь X");
    // Вначале выполним удаление
    adapter.Update(users.Select(null, null,
 DataViewRowState.Deleted));
    // Далее выполним обновление
    adapter.Update(users.Select(null, null,
      DataViewRowState.ModifiedCurrent));
    // В конце выполним вставку
    adapter.Update(users.Select(null, null,
 DataViewRowState.Added));
    // Отображение таблицы после обновления
    Console.WriteLine("Строки таблицы после обновления:");
    foreach (DataRow row in users.Rows)
    {
      Console.WriteLine("{0}: '{1}' ", row[0], row[1]);
    }
  }
}
```

Update(DataSet)

Метод выполняет необходимые операторы (INSERT, UPDATE или DELETE) для изменения строк в первой таблице объекта `DataSet`.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Указанные операторы должны быть явно прописаны в соответствующих свойствах объекта `DbDataAdapter` (InsertCommand, UpdateCommand, DeleteCommand), либо сконструированы с помощью класса `DbCommandBuilder`.

Подробное описание см. в пункте «Update(DataRow[])».

Синтаксис

```
public override int Update(DataSet dataSet);
```

`dataSet` – объект `DataSet`, используемый для обновления источника данных.

Возвращаемое значение

Количество строк, успешно обновленных из объекта `DataSet` в источнике данных.

Исключения

`InvalidOperationException`	Исходная таблица является недопустимой.
`DBConcurrencyException`	Попытка выполнить оператор INSERT, UPDATE или DELETE привела к нулевому количеству обработанных записей.
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class UpdateSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText =
      "create or replace table users ( " +
      "id integer primary key, name varchar(70));" +
      "insert into users (id, name) values (0, 'Пользователь A');"
 +
```

```
      "insert into users (id, name) values (1, 'Пользователь
 B');";
    con.Open();
    cmd.ExecuteNonQuery();
    con.Close();
    // Создание команды для выборки записей
    DbCommand selectCommand = factory.CreateCommand();
    selectCommand.Connection = con;
    selectCommand.CommandText =
      "select id, name from users";
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = selectCommand;
    // Создание объекта DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Заполнение объекта DataSet данными из БД
    DataSet dataset = new DataSet();
    adapter.Fill(dataset);
    // Изменение записей в таблице DataTable
    dataset.Tables[0].Rows[0]["name"] = "Новый Пользователь A";
    dataset.Tables[0].Rows[1].Delete();
    dataset.Tables[0].Rows.Add(2, "Пользователь X");
    // Обновление БД
    adapter.Update(dataset);
    // Отображение таблицы после обновления
    Console.WriteLine("Строки таблицы после обновления:");
    foreach (DataRow row in dataset.Tables[0].Rows)
    {
      Console.WriteLine("{0}: '{1}' ", row[0], row[1]);
    }
  }
}
```

Update(DataTable)

Метод выполняет необходимые операторы (INSERT, UPDATE или DELETE) для изменения строк в указанном объекте `DataTable`.

Указанные операторы должны быть явно прописаны в соответствующих свойствах объекта `DbDataAdapter` (InsertCommand, UpdateCommand, DeleteCommand), либо сконструированы с помощью класса `DbCommandBuilder`.

Подробное описание см. в пункте «Update(DataRow[])».

Синтаксис

```
public int Update(DataTable dataTable);
```

`dataTable` – объект `DataTable`, используемый для обновления источника данных.

Возвращаемое значение

Количество строк, успешно обновленных из объекта `DataTable` в источнике данных.

Исключения

`ArgumentNullException`	Объект `DataSet` является недопустимым.
`SystemException`	Возможные причины: • не существует объект `DataRow` или `DataTable` для обновления; • не существует объект `DataSet` для использования в качестве источника.
`InvalidOperationException`	Исходная таблица является недопустимой.
`DBConcurrencyException`	Попытка выполнить оператор INSERT, UPDATE или DELETE привела к нулевому количеству обработанных записей.
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class UpdateSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText =
      "create or replace table users ( " +
      "id integer primary key, name varchar(70));" +
      "insert into users (id, name) values (0, 'Пользователь A');"
 +
      "insert into users (id, name) values (1, 'Пользователь
 B');";
    con.Open();
```

```
    cmd.ExecuteNonQuery();
    con.Close();
    // Создание команды для выборки записей
    DbCommand selectCommand = factory.CreateCommand();
    selectCommand.Connection = con;
    selectCommand.CommandText =
      "select id, name from users";
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = selectCommand;
    // Создание объекта DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Заполнение объекта DataTable данными из таблицы БД
    DataTable users = new DataTable();
    adapter.Fill(users);
    // Изменение записей в таблице DataTable
    users.Rows[0]["name"] = "Новый Пользователь A";
    users.Rows[1].Delete();
    users.Rows.Add(2, "Пользователь X");
    // Обновление БД
    adapter.Update(users);
    // Отображение таблицы после обновления
    Console.WriteLine("Строки таблицы после обновления:");
    foreach (DataRow row in users.Rows)
    {
      Console.WriteLine("{0}: '{1}' ", row[0], row[1]);
    }
  }
}
```

Update(DataSet, String)

Метод выполняет необходимые операторы (INSERT, UPDATE или DELETE) для изменения строк в указанной таблице объекта `DataSet`.

Указанные операторы должны быть явно прописаны в соответствующих свойствах объекта `DbDataAdapter` (InsertCommand, UpdateCommand, DeleteCommand), либо сконструированы с помощью класса `DbCommandBuilder`.

Подробное описание см. в пункте «Update(DataRow[])».

Синтаксис

```
public int Update(DataSet dataSet,  string srcTable);
```

`dataSet` – объект `DataSet`, используемый для обновления источника данных.

`srcTable` – имя таблицы в источнике данных, используемой для обновления данных.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Возвращаемое значение

Количество строк, успешно обновленных в источнике данных из объекта `DataSet`.

Исключения

`ArgumentNullException`	Объект `DataSet` является недопустимым.
`InvalidOperationException`	Исходная таблица является недопустимой.
`DBConcurrencyException`	Попытка выполнить оператор INSERT, UPDATE или DELETE привела к нулевому количеству обработанных записей.
`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class UpdateSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText =
      "create or replace table users ( " +
      "id integer primary key, name varchar(70));" +
      "insert into users (id, name) values (0, 'Пользователь A');"
 +
      "insert into users (id, name) values (1, 'Пользователь
 B');";
    con.Open();
    cmd.ExecuteNonQuery();
    con.Close();
    // Создание команды для выборки записей
    DbCommand selectCommand = factory.CreateCommand();
    selectCommand.Connection = con;
    selectCommand.CommandText =
```

```
      "select id, name from users";
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = selectCommand;
    // Создание объекта DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Заполнение объекта DataSet данными из БД
    DataSet dataset = new DataSet();
    adapter.Fill(dataset);
    // Изменение записей в таблице DataTable
    dataset.Tables[0].Rows[0]["name"] = "Новый Пользователь A";
    dataset.Tables[0].Rows[1].Delete();
    dataset.Tables[0].Rows.Add(2, "Пользователь X");
    // Обновление БД
    adapter.Update(dataset, "Table");
    // Отображение таблицы после обновления
    Console.WriteLine("Строки таблицы после обновления:");
    foreach (DataRow row in dataset.Tables[0].Rows)
    {
      Console.WriteLine("{0}: '{1}' ", row[0], row[1]);
    }
  }
}
```

События

FillError

Событие генерируется при возникновении ошибки в методе `Fill`.

Событие FillError позволяет пользователю указать клиентскому приложению, должна ли операция заполнения объекта данными продолжаться после возникновения ошибки. Событие FillError может произойти, например, в следующих случаях:

- данные, добавляемые в DataSet, не могут быть преобразованы в тип данных среды CLR без потери точности;
- добавляемая строка содержит данные, нарушающие ограничение целостности, которое должно быть применено к DataColumn в DataSet.

Наиболее часто событие FillError генерируется в процессе добавления данных в объект `DataSet` с нарушением целостности, например, когда добавляемые данные не могут быть приведены к типу данных ADO.NET-провайдера без потери точности.

При возникновении FillError-события текущая строка не добавляется в объект `DataTable`. Обработчик события FillError позволяет обработать эту ситуацию, и строка будет либо добавлена в `DataTable`, либо проигнорирована методом `Fill`() перед выполнением операции со следующей строкой.

Обработчик FillError-события получает аргумент FillErrorEventArgs, который содержит специальные данные о событии, и которые позволяют эффективно обработать это

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

событие. Свойство Continue аргумента FillErrorEventArgs определяет, должно ли исключение игнорироваться или должно быть обработано как ошибка.

Синтаксис

```
public class FillErrorEventArgs : EventArgs;
```

Свойства аргумента FillErrorEventArgs (значения EventArgs):

- Errors – тип события (исключение Exception);
- DataTable – имя объекта DataTable, в котором возникла ошибка;
- Values – массив объектов со значениями строки, при добавлении которой возникла ошибка. Нумерация элементов массива Values соответствует нумерации столбцов добавляемой строки. Например, Values[0] – значение, добавляемое как первый столбец строки;
- Continue – реакция на событие. Если свойство Continue имеет значение false, текущая операция `Fill` будет остановлена с вызовом исключения. Если свойство Continue имеет значение true, то исключение (ошибка) будет проигнорировано и операция `Fill` будет продолжена.

Примеры

1) Пример обработчика событий для загружаемой таблицы из трех столбцов.

```
SqlDataAdapter da;
// ... code to set up the data adapter da.FillError += new
 FillErrorEventHandler(da_FillError);
DataSet ds = new DataSet();
da.Fill(ds, "MyTable");
private void da_FillError(object sender, FillErrorEventArgs e)
{
// ... code to identify and correct the error
// add the fixed row to the table DataRow
dr = e.DataTable.Rows.Add(new object[] {e.Values[0], e.Values[1],
 e.Values[2]});
// continue the Fill with the rows remaining in the data source
e.Continue = true; }
```

2) В примере добавляется обработчик события для события FillError класса `DataAdapter`, выявляется потенциальная потеря точности и предоставляется возможность выполнить определенные действия в ответ на исключение.

```
adapter.FillError += new FillErrorEventHandler(FillError);
DataSet dataSet = new DataSet();
adapter.Fill(dataSet, "ThisTable");
protected static void FillError(object sender, FillErrorEventArgs
 args)
{
  if (args.Errors.GetType() == typeof(System.OverflowException))
  {
    // Code to handle precision loss.
```

```
    // Add a row to table using the values from the first two
 columns.
    DataRow myRow = args.DataTable.Rows.Add(new object[]
       {args.Values[0], args.Values[1], DBNull.Value});
    // Set the RowError containing the value for the third column.
    args.RowError =
       "OverflowException Encountered. Value from data source: " +
       args.Values[2];
    args.Continue = true;
  }
}
```

3)

…

```
Friend Function Load(ByVal strSQL As String, ByRef DataSet As
System.Data.DataSet, Optional ByVal strTableName As String = "")
 As
Boolean Implements IBackendDataProvider.Load

DbCommand =System.Data.SqlClient.DbCommand
Adapter = New System.Data.SqlClient.DbDataAdapter
DoWeClose  Boolean
 If Parent.ConnectionString.Length <> 0 Then
 DoWeClose = Parent.OpenConditional()

If (DataSet Is Nothing) Then
 DataSet = New System.Data.DataSet
End If

DbCommand = GetSQLCommandObject(strSQL)
DbCommand.CommandTimeout = Parent.CommandTimeout
Adapter = New System.Data.SqlClient.DbDaAdapter(DbCommand)
AddHandler Adapter.FillError, AddressOf FillError
Try
If strTableName.Length <> 0 Then
Adapter.Fill(DataSet, strTableName)
Else
Adapter.Fill(DataSet)
// FAILS HERE - HIGHLIGHTED GREEN WITH ERROR ABOVE
 End If
 Catch
Throw
End Try
RemoveHandler Adapter.FillError, AddressOf FillError
Adapter = Nothing
DbCommand.Dispose()
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
DbCommand = Nothing
Parent.CloseConditional(DoWeClose)
 End If
 End Function
```

RowUpdating

Событие RowUpdating генерируется перед обновлением любой строки из DataSet в источнике данных, поэтому с его помощью можно изменить поведение обновления до того, как оно начнется, чтобы обеспечить, например:

- дополнительную обработку при обновлении;
- сохранить ссылку на обновленную строку;
- отменить текущее обновление;
- запланировать текущее обновление для пакетной обработки впоследствии и т. д.

Синтаксис

```
public class RowUpdatingEventArgs:EventArgs;
```

Свойства аргумента RowUpdatingEventArgs (значения EventArgs):

1) `Command` – ссылка на объект `Command`, применяемый для выполнения обновлений;
2) `Errors` – получать ошибки, выявляемые ADO.NET-провайдером при выполнении свойства Command;
3) `Row` – ссылка на объект `DataRow`, содержащий обновленные сведения;
4) `StatementType` – тип выполняемого обновления (SELECT, INSERT, UPDATE, DELETE);
5) `TableMappings` – описание сопоставленных отношений между исходной таблицей и объектом `DataTable` (элемент коллекции DbDataAdapter.TableMappings);
6) `Status` – реакция на ошибку:

- `Continue` – продолжить операцию обновления;
- `ErrorsOccurred` – прервать операцию обновления и сгенерировать исключение;
- `SkipCurrentRow` – пропустить текущую строку и продолжить операцию обновления;
- `SkipAllRemainingRows` – прервать операцию обновления без генерации исключения.

При возникновении события свойство Status совпадает с Continue или ErrorsOccurred.

Назначение свойству Status значения ErrorsOccurred приведет к генерации исключения.

При использовании других значений Status исключение не вызывается.

Генерацию нужного исключения можно задать с помощью свойства Errors.

Пример

См. пример в пункте «RowUpdated».

RowUpdated

Событие RowUpdated генерируется после обновления в источнике данных строки из DataSet.

Событие RowUpdated полезно для реагирования на ошибки и исключения, возникающие в ходе обновления данных. Сведения об ошибке можно добавить в объект DataSet, а также в логику повторов и т. д.

Синтаксис

```
public class RowUpdatedEventArgs : EventArgs;
```

Свойства аргумента RowUpdatedEventArgs (значения EventArgs) аналогичны свойствам аргумента RowUpdatingEventArgs (см. пункт «RowUpdating»).

Дополнительно определяются следующие свойства:

1) RecordsAffected – содержит количество строк, которые были изменены, вставлены или удалены при выполнении SQL-запроса;
2) RowCount – содержит количество строк, обработанных в пакете обновленных записей.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Data.LinterClient;

class RowUpdatedSample
{
  static void Main()
  {
    // Соединение с БД
    LinterDbConnection con = new LinterDbConnection(
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8");
    // Создание таблицы БД
    LinterDbCommand cmd = new LinterDbCommand(
      "create or replace table users ( " +
      "id integer primary key, name varchar(70));" +
      "insert into users (id, name) values (0, 'Пользователь A');"
 +
      "insert into users (id, name) values (1, 'Пользователь B');"
 +
      "insert into users (id, name) values (2, 'Пользователь
 C');", con);
    con.Open();
    cmd.ExecuteNonQuery();
    con.Close();
    // Создание объекта LinterDbDataAdapter
    LinterDbDataAdapter adapter = new LinterDbDataAdapter(
      "select id, name from users", con);
    adapter.RowUpdating += new
 EventHandler<RowUpdatingEventArgs>(OnRowUpdating);
```

```
   adapter.RowUpdated += new
EventHandler<RowUpdatedEventArgs>(OnRowUpdated);
   // Создание объекта LinterDbCommandBuilder
   LinterDbCommandBuilder builder = new
LinterDbCommandBuilder(adapter);
   // Заполнение объекта DataTable данными из таблицы БД
   DataTable users = new DataTable();
   adapter.Fill(users);
   // Изменяем записи в таблице DataTable
   users.Rows[0]["name"] = "Новый пользователь A";
   users.Rows[1].Delete();
   users.Rows[2]["name"] = "Новый пользователь C";
   // Имитируем несогласованное изменение первой записи в БД
   cmd.CommandText =
     "update users set name = 'Пользователь X' where id = 0";
   con.Open();
   cmd.ExecuteNonQuery();
   con.Close();
   // Теперь обновление БД должно завершиться ошибкой
   adapter.Update(users);
   // Отображение таблицы после обновления
   Console.WriteLine("Строки таблицы после обновления:");
   foreach (DataRow row in users.Rows)
   {
     Console.Write("{0}: '{1}' ", row[0], row[1]);
     if (row.HasErrors)
     {
       Console.WriteLine("(при обновлении строки произошла
ошибка)");
     }
     else
     {
       Console.WriteLine("(обновление строки выполнено
успешно)");
     }
   }
 }
 static void OnRowUpdating(object sender, RowUpdatingEventArgs
args)
 {
   if (args.StatementType == StatementType.Delete)
   {
     Console.WriteLine("{0} Удаление записи '{1}'", DateTime.Now,
       args.Row["name", DataRowVersion.Original]);
   }
 }
```

```
  static void OnRowUpdated(object sender, RowUpdatedEventArgs
 args)
  {
    if (args.Status == UpdateStatus.ErrorsOccurred)
    {
      args.Row.RowError = args.Errors.Message;
      args.Status = UpdateStatus.SkipCurrentRow;
    }
  }
}
```

```
Результат выполнения примера:

07.12.2012 20:13:47 Удаление записи 'Пользователь B'
Строки таблицы после обновления:
0: 'Новый пользователь A' (при обновлении строки произошла ошибка)
2: 'Новый пользователь C' (обновление строки выполнено успешно)
```

Класс DbCommandBuilder

Класс `DbCommandBuilder` предназначен для автоматического генерирования однотабличных SQL-команд, которые позволяют согласовывать изменения, вносимые в объект `DataSet`, со связанной с ним БД. Например, если создать экземпляр `DbCommandBuilder` и связать его с объектом `DbDataAdapter`, то `DbCommandBuilder` сгенерирует SQL-запросы на обновление таблицы на основе запроса, указанного в свойстве SelectCommand объекта `DbDataAdapter`.

Для генерации запросов UPDATE, INSERT и DELETE методы класса `DbCommandBuilder` обращаются к БД за именами базовой таблицы и столбцов, а также о первичных ключах в таблице и в выборке данных из этой таблицы.

SQL-запросы на обновление данных генерируются при выполнении следующих условий:

- запрос возвращает данные только из одной таблицы (обновляемого представления);
- в таблице определен первичный ключ;
- столбец с первичным ключом таблицы входит в выборку данных из этой таблицы.

Первичный ключ гарантирует, что `DbCommandBuilder` обновит не более одной записи.

Если свойство SelectCommand задается динамически во время выполнения клиентского приложения, например, при помощи интерактивного запроса, принимающего от пользователя клиентского приложения текстовые команды, то во время работы приложения нельзя задавать соответствующие свойства InsertCommand, UpdateCommand или DeleteCommand. Если объект `DataTable` сопоставляется с одной таблицей БД или создается из нее, то можно воспользоваться преимуществом объекта `DbCommandBuilder` для автоматического создания запросов DeleteCommand, InsertCommand и UpdateCommand объекта `DbDataAdapter`.

Минимальным требованием для работы автоматического создания команд является задание свойства SelectCommand. Схема таблицы, получаемая свойством

SelectCommand, определяет синтаксис автоматически созданных инструкций INSERT, UPDATE и DELETE.

Для получения метаданных, необходимых для создания команд INSERT, UPDATE и DELETE, объект `DbCommandBuilder` должен выполнить команду `SelectCommand`, что приводит к дополнительному взаимодействию с источником данных, а это может снизить производительность клиентского приложения. Если такое положение дел нежелательно, то вместо использования `DbCommandBuilder` следует задавать команды манипулирования данными явным образом.

Свойство SelectCommand должно также возвратить, по крайней мере, один столбец первичного ключа или столбец с атрибутом UNIQUE. Если отсутствует и то и другое, то возникнет исключение InvalidOperation и автоматическая генерация команд выполняться не будет.

При наличии связи с `DataAdapter` объект `DbCommandBuilder` автоматически создает свойства InsertCommand, UpdateCommand и DeleteCommand объекта `DataAdapter`, если они являются пустыми ссылками. Если для свойства уже существует значение Command, то оно и используется.

Представления БД, созданные соединением двух или более таблиц, не считаются одной таблицей БД. В данном случае нельзя использовать класс `DbCommandBuilder` для автоматического создания команд. Необходимо указывать команды явным образом.

Действия, выполняемые автоматически сконструированными командами, приведены в таблице 29.

Таблица 29. Правила конструирования команд

Команда	Действия
`InsertCommand`	1) Выполняет операцию вставки строки в источнике данных для всех строк таблицы со свойством RowState, равным Added 2) Вставляет значения для всех обновляемых столбцов (за исключением идентификаторов, выражений или временных меток)
`UpdateCommand`	1) Выполняет обновление строк в источнике данных для всех строк таблицы со свойством RowState, равным Modified 2) Обновляет значения всех столбцов, за исключением столбцов, которые не являются обновляемыми, например, идентификаторов или выражений 3) Обновляет все строки, в которых значения столбцов в источнике данных совпадают со значениями столбцов первичных ключей строки и в которых оставшиеся в источнике данных столбцы совпадают с исходными значениями строки **Примечание** Дополнительные сведения см. ниже в описании модели оптимистичного параллелизма для обновлений и удалений
`DeleteCommand`	1) Выполняет операцию удаления строки в источнике данных для всех строк таблицы, у которых свойство RowState равно Deleted 2) Удаляет все строки, в которых значения столбцов совпадают со значениями столбцов первичных ключей строки и в которых

Команда	Действия
	оставшиеся в источнике данных столбцы совпадают с исходными значениями строки **Примечание** Дополнительные сведения см. ниже в описании модели оптимистичного параллелизма для обновлений и удалений

Логика автоматического создания команд для операций UPDATE и DELETE базируется на модели *оптимистичного параллелизма*, то есть записи не блокируются для редактирования и могут быть в любое время изменены другими пользователями или процессами. Вследствие того, что запись можно изменить после ее возврата из инструкции SELECT, но до выполнения операции UPDATE или DELETE, то автоматически созданная инструкция UPDATE или DELETE содержит предложение WHERE, указывающее на то, что строка обновляется только в случае наличия в ней всех исходных значений и она не была удалена из источника данных. Это делается во избежание перезаписи новых данных. Когда автоматически созданное обновление пытается обновить строку, которая была удалена или не содержит исходные значения, найденные в `DataSet`, команда не изменяет записи и вызывается исключение DBConcurrencyException.

Если требуется выполнить операцию UPDATE или DELETE независимо от исходных значений, необходимо явно задать свойство UpdateCommand для `DataAdapter` и не полагаться на автоматическое создание команд.

На автоматическое создание команд накладываются следующие ограничения:

- в конструируемой команде могут использоваться только несвязанные таблицы. Логика автоматического создания команд для операций INSERT, UPDATE или DELETE предполагает использование изолированных таблиц, не принимая во внимание связи с другими таблицами в источнике данных. В результате при вызове UPDATE для выполнения изменений столбца, участвующего в ограничении внешнего ключа БД, может произойти ошибка. Чтобы избежать её, не следует использовать `DbCommandBuilder` для обновления столбцов, вовлеченных в ограничение внешнего ключа (или в другие ограничения целостности). Вместо этого нужно явно задавать команды, используемые для выполнения операции.
- имена таблиц и столбцов не должны содержать спецсимволов. Логика автоматического создания команд не допускает в именах столбцов или таблиц любых специальных символов, например, пробелов, точек, двойных кавычек или других символов, отличных от алфавитно-цифровых, однако СУБД ЛИНТЕР разрешает спецсимволы внутри двойных кавычек. Поддерживаются указанные полностью имена таблиц в виде catalog.schema.table.

Чтобы автоматически создать команды объекта `DataAdapter`, необходимо:

1) установить его свойство SelectCommand;
2) создать объект `CommandBuilder`, указав в аргументе тот объект `DataAdapter`, для которого должна автоматически создаваться команда.

При изменении свойства CommandText, принадлежащего свойству SelectCommand, после автоматического создания команд INSERT, UPDATE или DELETE может возникнуть исключение. Например, если измененное свойство SelectCommand.CommandText содержит сведения о схеме, которые не согласованы

с используемыми SelectCommand.CommandText сведениями при автоматическом создании команд INSERT, UPDATE или DELETE, то последующие вызовы к методу DataAdapter.Update могут содержать попытки обращения к столбцам, которых больше нет в текущей таблице, на которую ссылается SelectCommand, и в этом случае возникнет исключение.

Для обновления сведений о схеме, используемой `CommandBuilder` для автоматического создания команд, следует вызвать метод `RefreshSchema` объекта `CommandBuilder`.

Чтобы узнать, какая команда была автоматически создана, необходимо с помощью методов `GetInsertCommand`, `GetUpdateCommand` и `GetDeleteCommand` объекта `CommandBuilder` получить ссылку на автоматически созданные команды и проверить свойство CommandText, соответствующее команде.

Следующий пример выводит на консоль автоматически созданную команду UPDATE:

```
Console.WriteLine(builder.GetUpdateCommand().CommandText)
```

В следующем примере повторно создается таблица AUTO из набора данных autoDS. Метод `RefreshSchema` вызывается для внесения новых сведений о столбцах в автоматически созданные команды.

```
// C#
    …
    adapter.SelectCommand.CommandText =
      "select MAKE, MODEL from SYSTEM.AUTO";
    builder.RefreshSchema();

    autoDS.Tables.Remove(autoDS.Tables["AUTO"]);
    adapter.Fill(autoDS, "AUTO");
```

Конструкторы класса `LinterDbCommandBuilder` приведены в таблице 30.

Таблица 30. Конструкторы класса LinterDbCommandBuilder

Конструктор	Описание
LinterDbCommandBuilder	Создает новый экземпляр класса LinterDbCommandBuilder.
LinterDbCommandBuilder(LinterDbDataAdapter)	Создает новый экземпляр класса LinterDbCommandBuilder со связанным с ним объектом LinterDbDataAdapter.

Свойства класса `LinterDbCommandBuilder` приведены в таблице 31.

Таблица 31. Свойства класса LinterDbCommandBuilder

Свойство	Описание
CatalogLocation	Предоставляет/устанавливает объект CatalogLocation для экземпляра класса DbCommandBuilder.
CatalogSeparator	Предоставляет/устанавливает символ, который используется в качестве разделителя между идентификатором каталога и остальными идентификаторами.

Свойство	Описание
	Примечание В текущей версии ADO.NET-провайдера СУБД ЛИНТЕР данное свойство не поддерживается.
ConflictOption	Предоставляет/устанавливает значение типа ConflictOption, которое используется объектом DbCommandBuilder.
DataAdapter	Предоставляет/устанавливает объект DbDataAdapter, который связан с объектом DbCommandBuilder.
QuotePrefix	Предоставляет/устанавливает начальный символ или символы, используемые для именования объектов БД, имена которых содержат русские буквы или специальные символы.
QuoteSuffix	Предоставляет/устанавливает конечный символ или символы, используемые для именования объектов БД, имена которых содержат русские буквы или специальные символы.
SchemaSeparator	Предоставляет или задает строку, которую можно использовать в качестве разделителя имени владельца таблицы и имени таблицы в полной спецификации таблицы.
SetAllValues	Предоставляет/устанавливает признак, какие значения столбца включены в сгенерированном операторе UPDATE: все или только измененные.

Методы класса `LinterDbCommandBuilder` приведены в таблице 32.

Таблица 32. Методы класса LinterDbCommandBuilder

Метод	Описание
DeriveParameters	Извлекает сведения о параметрах из хранимой процедуры, указанной в объекте LinterDbCommand, и включает их в коллекцию параметров Parameters указанного объекта LinterDbCommand.
GetDeleteCommand	Предоставляет автоматически сгенерированный для выполнения операций удаления в БД объект DbCommand.
GetDeleteCommand(Boolean)	Предоставляет автоматически сгенерированный для выполнения операций удаления в БД объект DbCommand с заданным правилом именования параметров.
GetInsertCommand	Предоставляет автоматически сгенерированный для выполнения операций добавления записей в БД объект DbCommand.
GetInsertCommand(Boolean)	Предоставляет автоматически сгенерированный для выполнения операций добавления записей в таблицу БД объект DbCommand с заданным правилом именования параметров.
GetUpdateCommand	Предоставляет автоматически сгенерированный для выполнения операции обновления записей в БД объект DbCommand.
GetUpdateCommand(Boolean)	Предоставляет автоматически сгенерированный для выполнения операции обновления записи в таблице

Метод	Описание
	БД объект DbCommand с заданным правилом именования параметров.
QuoteIdentifier	Предоставляет обрамленный прямыми двойными кавычками указанный идентификатор в спецификации объекта БД.
RefreshSchema	Обновляет автоматически сгенерированные SQL-операторы, связанные с указанным объектом DbCommandBuilder.
UnquoteIdentifier	Предоставляет указанный идентификатор в спецификации объектов БД без обрамляющих его кавычек.

Конструкторы

ADO.NET-провайдер СУБД ЛИНТЕР обеспечивает поддержку двух конструкторов класса `LinterDbCommandBuilder`.

LinterDbCommandBuilder

Синтаксис

```
public LinterDbCommandBuilder();
```

Возвращаемое значение

Новый экземпляр класса `LinterDbCommandBuilder`.

LinterDbCommandBuilder(LinterDbDataAdapter)

Синтаксис

```
public LinterDbCommandBuilder(LinterDbDataAdapter adapter);
```

`adapter` – объект класса `LinterDbDataAdapter`.

Возвращаемое значение

Новый экземпляр класса `LinterDbCommandBuilder` со связанным с ним объектом `LinterDbDataAdapter`.

Свойства

CatalogLocation

Предоставляет или устанавливает объект CatalogLocation для экземпляра класса `DbCommandBuilder`.

Примечание
В текущей версии ADO.NET-провайдера СУБД ЛИНТЕР данное свойство не поддерживается.

Декларация

```
public virtual CatalogLocation CatalogLocation {get; set;};
```

Допустимые значения свойства CatalogLocation приведены в таблице 33.

Таблица 33. Допустимые значения свойства CatalogLocation

Значение	Описание
`Start`	Указывает, что в тексте команды в полном имени таблицы имя каталога находится перед именем схемы
`End`	Указывает, что в тексте команды в полном имени таблицы имя каталога находится после имени схемы

Примечание
Значения Start и End являются взаимоисключающими.

Значение свойства

Объект CatalogLocation.

Исключения

Отсутствуют.

CatalogSeparator

Предоставляет или устанавливает символ, который используется в качестве разделителя между идентификатором каталога и остальными идентификаторами.

Примечание
В текущей версии ADO.NET-провайдера СУБД ЛИНТЕР данное свойство не поддерживается.

Декларация

```
public override string CatalogSeparator {get; set;};
```

Значение свойства

Символьная строка (значение типа string), содержащая установленный разделитель.

Исключения

Отсутствуют.

ConflictOption

Предоставляет или устанавливает значение типа ConflictOption, которое используется объектом DbCommandBuilder.

Декларация

```
public virtual ConflictOption ConflictOption {get; set;};
```

Допустимые значения свойства ConflictOption приведены в таблице 34.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Таблица 34. Допустимые значения свойства ConflictOption

Значение	Описание
`CompareAllSearchableValues` (по умолчанию)	Операторы обновления и удаления будут включать все столбцы, по которым может осуществляться поиск (т.е. все столбцы, которые возвращает SelectCommand, будут использованы в опции WHERE запроса UpdateCommand или DeleteCommand) (за исключением BLOB-столбцов). Эквивалентно заданию CompareAllValuesUpdate | CompareAllValuesDelete
`CompareRowVersion`	Если в таблице имеются какие-либо столбцы типа Timestamp, то они используются в предложении WHERE для всех созданных операторов обновления. Эквивалентно заданию CompareRowVersionUpdate | CompareRowVersionDelete **Примечание** В СУБД ЛИНТЕР столбцы Timestamp отсутствуют, поэтому в предложении WHERE будут использованы только столбцы, входящие в первичный ключ
`OverwriteChanges`	Все операторы обновления и удаления включают только столбцы с атрибутом PrimaryKey в конструкции WHERE. Если не определен ни один столбец с атрибутом PrimaryKey, то все столбцы, по которым может осуществляться поиск, будут включены в предложение WHERE. Это эквивалентно OverwriteChangesUpdate | OverwriteChangesDelete

Значение свойства

Значение типа ConflictOption.

Исключения

`ArgumentOutOfRangeException`	Недопустимое значение свойства.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
```

```
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
"DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8";
    con.Open();
    DbCommand cmd = factory.CreateCommand();
    cmd.CommandText = "SELECT MODEL, PERSONID FROM AUTO;";
    cmd.Connection = con;
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = cmd;
    // Связывание объектов DbDataAdapter и DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // ConflictOption.CompareAllSearchableValues
    builder.ConflictOption =
ConflictOption.CompareAllSearchableValues;
    Console.WriteLine(builder.ConflictOption);
    Console.WriteLine(cmd.CommandText);
    Console.WriteLine(builder.GetUpdateCommand().CommandText);
    Console.WriteLine();
    // ConflictOption.CompareRowVersion
    builder.ConflictOption = ConflictOption.CompareRowVersion;
    builder.RefreshSchema();
    Console.WriteLine(builder.ConflictOption);
    Console.WriteLine(cmd.CommandText);
    Console.WriteLine(builder.GetUpdateCommand().CommandText);
    Console.WriteLine();
    // ConflictOption.OverwriteChanges
    builder.ConflictOption = ConflictOption.OverwriteChanges;
    builder.RefreshSchema();
    Console.WriteLine(builder.ConflictOption);
    Console.WriteLine(cmd.CommandText);
    Console.WriteLine(builder.GetUpdateCommand().CommandText);
    Console.WriteLine();
    // Изменение свойства SelectCommand
    cmd.CommandText = "SELECT MODEL, MAKE, PERSONID FROM AUTO;";
    // Обновление схемы
    builder.RefreshSchema();
    Console.WriteLine(builder.ConflictOption);
    Console.WriteLine(cmd.CommandText);
    Console.WriteLine(builder.GetUpdateCommand().CommandText);
    Console.WriteLine();
    // Освобождение ресурсов
    builder.Dispose();
    // Закрытие подключения к БД
    con.Close();
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
  }
}

Результат выполнения примера:

CompareAllSearchableValues
SELECT MODEL, PERSONID FROM AUTO;
UPDATE "SYSTEM"."AUTO" SET "MODEL" = :ret1, "PERSONID" = :ret2
 WHERE (((:ret3 = 1 AND "MODEL"
IS NULL) OR ("MODEL" = :ret4)) AND ("PERSONID" = :ret5))

CompareRowVersion
SELECT MODEL, PERSONID FROM AUTO;
UPDATE "SYSTEM"."AUTO" SET "MODEL" = :ret1, "PERSONID" = :ret2
 WHERE (("PERSONID" = :ret3))

OverwriteChanges
SELECT MODEL, PERSONID FROM AUTO;
UPDATE "SYSTEM"."AUTO" SET "MODEL" = :ret1, "PERSONID" = :ret2
 WHERE (("PERSONID" = :ret3))

OverwriteChanges
SELECT MODEL, MAKE, PERSONID FROM AUTO;
UPDATE "SYSTEM"."AUTO" SET "MODEL" = :ret1, "MAKE" = :ret2,
 "PERSONID" = :ret3
WHERE (("PERSONID" = :ret4))
```

DataAdapter

Предоставляет или устанавливает объект DbDataAdapter, который связан с объектом DbCommandBuilder (т.е. данное свойство определяет объект DbDataAdapter, для которого автоматически генерируются SQL-запросы обновления таблицы).

Объект DbCommandBuilder регистрирует себя в качестве обработчика событий RowUpdating, которые генерируются объектом DbDataAdapter, указанным в этом свойстве.

Когда создается новый экземпляр объекта DbCommandBuilder, то все связанные с ним объекты DbDataAdapter освобождаются.

Декларация

```
public DbDataAdapter DataAdapter {get; set;};
```

Значение свойства

Объект DbDataAdapter.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = "User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    DbCommand cmd = factory.CreateCommand();
    cmd.CommandText = "select MAKE, PERSONID from AUTO";
    cmd.Connection = con;
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = cmd;
    // Получение данных из БД
    DataTable dt = new DataTable();
    adapter.Fill(dt);
    // Изменение данных
    dt.Rows[0][0] = DateTime.Now;
    // Связывание объектов DbDataAdapter и DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Обновление данных
    adapter.Update(dt);
    // Отображение сгенерированного запроса
    Console.WriteLine(builder.GetUpdateCommand().CommandText);
    // Освобождение ресурсов
    builder.Dispose();
    // Закрытие подключения к БД
    con.Close();
  }
}

Результат выполнения примера:

UPDATE "SYSTEM"."AUTO" SET "MAKE" = :param1, "PERSONID" = :param2
 WHERE (((:param3 = 1 AND "MAKE" IS NULL) OR ("MAKE" = :param4))
   AND ("PERSONID" = :param5))
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

QuotePrefix

Предоставляет или устанавливает начальный символ или символы, используемые для именования объектов БД (например, таблиц или столбцов), имена которых содержат русские буквы или специальные символы.

В СУБД ЛИНТЕР можно задавать только двойные прямые кавычки, например, таблица "Гл. бухгалтер"."Материальные ценности".

Значение по умолчанию – двойные прямые кавычки(").

Декларация

```
public string QuotePrefix {get; set;};
```

Значение свойства

Используемый начальный символ или символы (значение типа String).

Исключения

`ArgumentException`	Попытка установить значение, отличное от символа двойных кавычек.
`InvalidOperationException`	Попытка изменить свойство после создания команды INSERT, UPDATE или DELETE.

Примечание
Несмотря на то, что изменение свойства QuotePrefix после создания операции вставки, обновления или удаления невозможно, значения этого свойства можно изменить после вызова метода `Update` DataAdapter.

Пример

См. пример в пункте QuoteSuffix.

QuoteSuffix

Предоставляет или задает конечный символ или символы, используемые для именования объектов БД (например, таблиц или столбцов), имена которых содержат русские буквы или специальные символы.

Декларация

```
public string QuoteSuffix {get; set;};
```

Значение свойства

Используемый конечный символ или символы (значение типа string).

Исключения

`ArgumentException`	Попытка установить значение, отличное от символа двойных кавычек.
`InvalidOperationException`	Попытка изменить свойство после создания команды INSERT, UPDATE или DELETE.

Примечание
Несмотря на то, что изменение свойства QuotePrefix после создания операции вставки, обновления или удаления невозможно, значения этого свойства можно изменить после вызова метода `Update` DataAdapter.

Пример

Сформировать полную спецификации таблицы БД «Гл. бухгалтер. Материальные ценности».

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandBuilderSample
{
  static void Main()
  {
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    string user_name = "Гл. бухгалтер";
    string tab_name = "Материальные ценности";
    DbCommandBuilder cb = factory.CreateCommandBuilder();
    string spec_tab = cb.QuotePrefix + user_name + cb.QuoteSuffix
 + cb.SchemaSeparator
    + cb.QuotePrefix + tab_name + cb.QuoteSuffix;
    Console.WriteLine(spec_tab);
  }
}

Результат выполнения примера:
"Гл. бухгалтер"."Материальные ценности"
```

SchemaSeparator

Предоставляет или задает строку, которую можно использовать в качестве разделителя имени владельца таблицы и имени таблицы в полной спецификации таблицы.

В СУБД ЛИНТЕР разделителем является точка. Попытка установить другой разделитель вызовет исключение ArgumentException.

Значение по умолчанию – точка.

Декларация

```
public string SchemaSeparator {get; set;};
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Значение свойства

Символьная строка, содержащая используемый или устанавливаемый разделитель (значение типа string).

Исключения

`ArgumentException`	Попытка установить значение, отличное от символа точки.
`InvalidOperationException`	Попытка изменить свойство после создания команды INSERT, UPDATE или DELETE.

Примечание
Несмотря на то, что изменение свойства QuotePrefix после создания операции вставки, обновления или удаления невозможно, значения этого свойства можно изменить после вызова метода `Update` DataAdapter.

Пример

Сформировать полную спецификацию таблицы SYSTEM.AUTO. Разделитель взять из свойства SchemaSeparator.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandBuilderSample
{
  static void Main()
  {
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    string user_name = "SYSTEM";
    string tab_name = "AUTO";
    DbCommandBuilder cb = factory.CreateCommandBuilder();
    string spec_tab = user_name + cb.CatalogSeparator + tab_name;
    Console.WriteLine(spec_tab);
  }
}

Результат выполнения примера:

SYSTEM.AUTO
```

SetAllValues

Предоставляет или задает признак, какие значения столбца включены в сгенерированном операторе UPDATE: все или только измененные.

Оператор UPDATE, созданный объектом DbCommandBuilder, может включать значения всех столбцов таблицы (как обновляемых, так и не обновляемых) или же только значения измененных столбцов. Задание для свойства SetAllValues значения true определяет, что создаваемый оператор UPDATE должен включать все столбцы, независимо от того, изменялись их значения или нет.

Декларация

```
public bool SetAllValues {get; set;};
```

Значение свойства

Значение типа boolean:

- true, если оператор UPDATE, созданный объектом DbCommandBuilder, включает все столбцы;
- false, если он включает только измененные столбцы.

Значение по умолчанию false.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Data.LinterClient;

class CommandBuilderSample
{
  static void Main()
  {
    // создать соединение с БД
    LinterDbConnection con = new LinterDbConnection();
    con.ConnectionString =
 "DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8";
    // создать объект LinterDbDataAdapter
    LinterDbDataAdapter adapter = new LinterDbDataAdapter(
      "select * from AUTO", con);
    adapter.RowUpdated += new
 EventHandler<RowUpdatedEventArgs>(OnRowUpdated);
    // создать объект LinterDbCommandBuilder
    LinterDbCommandBuilder builder = new
 LinterDbCommandBuilder(adapter);
    // загрузить данные в DataTable
    DataTable dt = new DataTable();
    adapter.Fill(dt);
    // изменить в строке с id=250 поле COLOR и MAKE
```

```
    dt.Rows[250]["COLOR"] = "BLACK";
    dt.Rows[250]["MAKE"] = "FORD";
    // установить SetAllValues false  и посмотреть какой Update -
 оператор
    // сгенерировал LinterDbCommandBuilder
    builder.SetAllValues = false;
    Console.WriteLine("SetAllValues = false");
    adapter.Update(dt);
    Console.WriteLine();
    // изменить в строке с id=250 поле COLOR и MAKE
    dt.Rows[250]["COLOR"] = "WHITE";
    dt.Rows[250]["MAKE"] = "OPEL";
    // установить SetAllValues true  и посмотреть какой Update -
 оператор
    //сгенерировал LinterDbCommandBuilder
    builder.SetAllValues = true;
    Console.WriteLine("SetAllValues = true");
    adapter.Update(dt);
    Console.WriteLine();
    // освободить ресурсы
    builder.Dispose();
  }
  static void OnRowUpdated(object sender, RowUpdatedEventArgs
 args)
  {
    Console.WriteLine(args.Command.CommandText);
  }
}
```

Результат выполнения примера:

```
SetAllValues = false
UPDATE "SYSTEM"."AUTO" SET "MAKE" = :param1, "COLOR" = :param2
WHERE (((:param3 = 1 AND "MAKE" IS NULL) OR ("MAKE" = :param4))
 AND ((:param5 = 1 AND "MODEL" IS NULL)
OR ("MODEL" = :param6)) AND ((:param7 = 1 AND "BODYTYPE" IS NULL)
 OR ("BODYTYPE" = :param8))
AND ((:param9 = 1 AND "CYLNDERS" IS NULL) OR ("CYLNDERS"
 = :param10))
AND ((:param11 = 1 AND "HORSEPWR" IS NULL) OR ("HORSEPWR"
 = :param12))
AND ((:param13 = 1 AND "DSPLCMNT" IS NULL) OR ("DSPLCMNT"
 = :param14))
AND ((:param15 = 1 AND "WEIGHT" IS NULL) OR ("WEIGHT" = :param16))
 AND ((:param17 = 1
```

```
AND "COLOR" IS NULL) OR ("COLOR" = :param18)) AND ((:param19 = 1
 AND "YEAR" IS NULL)
OR ("YEAR" = :param20)) AND ((:param21 = 1 AND "SERIALNO" IS NULL)
OR ("SERIALNO" = :param22)) AND ((:param23 = 1 AND "CHKDATE" IS
 NULL)
OR ("CHKDATE" = :param24)) AND ((:param25 = 1 AND "CHKMILE" IS
 NULL)
OR ("CHKMILE" = :param26)) AND ("PERSONID" = :param27))

SetAllValues = true
UPDATE "SYSTEM"."AUTO" SET "MAKE" = :param1, "MODEL" = :param2,
 "BODYTYPE" = :param3,
"CYLNDERS" = :param4, "HORSEPWR" = :param5, "DSPLCMNT" = :param6,
 "WEIGHT" = :param7,
"COLOR" = :param8, "YEAR" = :param9, "SERIALNO" = :param10,
 "CHKDATE" = :param11,
"CHKMILE" = :param12, "PERSONID" = :param13 WHERE (((:param14 = 1
 AND "MAKE" IS NULL)
OR ("MAKE" = :param15)) AND ((:param16 = 1 AND "MODEL" IS NULL) OR
 ("MODEL" = :param17))
AND ((:param18 = 1 AND "BODYTYPE" IS NULL) OR ("BODYTYPE"
 = :param19)) AND ((:param20 = 1
AND "CYLNDERS" IS NULL) OR ("CYLNDERS" = :param21)) AND ((:param22
 = 1 AND "HORSEPWR" IS NULL)
OR ("HORSEPWR" = :param23)) AND ((:param24 = 1 AND "DSPLCMNT" IS
 NULL) OR ("DSPLCMNT" = :param25))
AND ((:param26 = 1 AND "WEIGHT" IS NULL) OR ("WEIGHT" = :param27))
 AND ((:param28 = 1
AND "COLOR" IS NULL) OR ("COLOR" = :param29)) AND ((:param30 = 1
 AND "YEAR" IS NULL)
OR ("YEAR" = :param31)) AND ((:param32 = 1 AND "SERIALNO" IS NULL)
 OR ("SERIALNO" = :param33))
AND ((:param34 = 1 AND "CHKDATE" IS NULL) OR ("CHKDATE"
 = :param35)) AND ((:param36 = 1
AND "CHKMILE" IS NULL) OR ("CHKMILE" = :param37)) AND ("PERSONID"
 = :param38))
```

Методы

См. также раздел Общие свойства и методы классов ADO.NET-провайдера.

DeriveParameters

Метод извлекает сведения о параметрах из хранимой процедуры, указанной в объекте LinterDbCommand, и включает их в коллекцию параметров Parameters указанного объекта LinterDbCommand.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

DeriveParameters перезаписывает любые имеющиеся сведения о параметрах для объекта LinterDbCommand.

Для получения этих сведений DeriveParameters требуется дополнительное обращение к СУБД. Если сведения о параметрах известны заранее, рекомендуется явно заполнить ими коллекцию параметров.

DeriveParameters можно использовать только с хранимыми процедурами.

Синтаксис

```
public static void DeriveParameters(LinterDbCommand cmd);
```

cmd – объект LinterDbCommand, определяющий хранимую процедуру, из которой необходимо извлечь сведения о параметрах. Извлеченные параметры добавляются в коллекцию параметров Parameters данного объекта LinterDbCommand.

Возвращаемое значение

Значение типа void.

Исключения

ArgumentNullException	Параметр cmd равен null.
InvalidOperationException	Возможные причины: • тип команды не CommandType.StoredProcedure; • свойство Connection не инициализировано; • свойство CommandText не инициализировано; • хранимая процедура не найдена.
LinterSqlException	Код завершения СУБД ЛИНТЕР не равен 0.

Пример

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class DeriveParametersSample
{
  static void Main()
  {
    LinterDbConnection conn = null;
    try
    {
      // Соединение с БД
      conn = new LinterDbConnection(
        "DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8");
      conn.Open();

      // Создание процедуры
```

```
      LinterDbCommand cmd = conn.CreateCommand();
      cmd.CommandText = @"
        create or replace procedure SUMMA (in A int; in B int)
          result int for debug
        code
          return A + B;
        end;";
      cmd.ExecuteNonQuery();

      // Выполнение процедуры
      cmd.CommandText = "SUMMA";
      cmd.CommandType = CommandType.StoredProcedure;
      LinterDbCommandBuilder.DeriveParameters(cmd);
      cmd.Parameters["A"].Value = 2;
      cmd.Parameters["B"].Value = 3;
      int result = (int)cmd.ExecuteScalar();
      Console.WriteLine("Результат: " + result);
    }
    catch (LinterSqlException ex)
    {
      Console.WriteLine(
        "Исключение ядра СУБД ЛИНТЕР \n" +
        "Текст сообщения: " + ex.Message + "\n" +
        "Код СУБД ЛИНТЕР: " + ex.Number + "\n" +
        "Код операционной системы: " + ex.LinterSysErrorCode +
 "\n");
    }
    catch (Exception ex)
    {
      Console.WriteLine(
        "Исключение ADO.NET провайдера \n" +
        "Тип ошибки: " + ex.GetType() + "\n" +
        "Сообщение: " + ex.Message + "\n");
    }
    finally
    {
      Console.WriteLine("Освобождение ресурсов.");
      if (conn != null)
      {
        conn.Close();
      }
      Console.WriteLine("Выполнение команды завершено.");
    }
  }
}
```

GetDeleteCommand

Метод предоставляет автоматически сгенерированный для выполнения операций удаления в БД объект DbCommand.

Клиентское приложение может с помощью метода `GetDeleteCommand` получить сгенерированный SQL-запрос DELETE и, при необходимости, модифицировать его (например, изменить значение CommandTimeout, а затем явно задать это значение для DbDataAdapter). В этом случае приложение должно явным образом вызвать метод `RefreshSchema`, если каким-либо образом изменится SELECT-запрос в свойстве DbDataAdapter.SelectCommand. В противном случае метод `GetDeleteCommand` будет продолжать использовать сведения от предыдущих команд, которые могут оказаться неверными.

Первоначально SQL-команды создаются, когда приложение вызывает метод `Update` либо метод `GetDeleteCommand`.

Синтаксис

```
public DbCommand GetDeleteCommand();
```

Возвращаемое значение

Автоматически созданный объект DbCommand, содержащий текст SQL-запроса для удаления строк из таблицы БД.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
 "DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8";
    con.Open();
    DbCommand cmd = factory.CreateCommand();
    cmd.CommandText = "select * from AUTO";
    cmd.Connection = con;
    DbDataAdapter adapter = factory.CreateDataAdapter();
```

```
    adapter.SelectCommand = cmd;
    // Связывание объектов DbDataAdapter и DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Отображение автоматически сгенерированной команды DELETE
    Console.WriteLine(builder.GetDeleteCommand().CommandText);
    // Освобождение ресурсов
    builder.Dispose();
    // Закрытие подключения к БД
    con.Close();
  }
}
```

GetDeleteCommand(Boolean)

Метод предоставляет автоматически сгенерированный для выполнения операций удаления в БД параметризованный объект DbCommand.

Синтаксис

```
public DbCommand
GetDeleteCommand(bool useColumnsForParameterNames);
```

`useColumnsForParameterNames` – задает правило именования параметров:

- true – имена параметров должны совпадать с именами столбцов (а если столбец неименованный, типа select to_char(id) …), то параметр для такого столбца не генерируется;
- false – имена параметров должны совпадать с их порядковыми номерами в запросе (:p1, :p2 и т. д.).

Значение по умолчанию false.

Для создания параметров с именами столбцов (useColumnsForParameterNames=true) необходимы дополнительные условия:

- должно быть задано значение свойства ParameterNameMaxLength, возвращаемое методом `GetSchema`, и присутствующее в коллекции DataSourceInformation; указанная в этом свойстве длина должна быть не меньшее длины сгенерированного имени параметра;
- сгенерированное имя параметра соответствует критериям, заданным в свойстве ParameterNamePattern, возвращаемом методом `GetSchema`, и присутствующее в коллекции DataSourceInformation;
- должно быть задано значение свойства ParameterMarkerFormat, возвращаемое методом `GetSchema`, и присутствующее в коллекции DataSourceInformation.

Возвращаемое значение

Автоматически созданный объект DbCommand, содержащий параметризованный текст SQL-запроса для удаления строк из таблицы БД.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
 "DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8";
    con.Open();
    DbCommand cmd = factory.CreateCommand();
    cmd.CommandText = "select MAKE, PERSONID from AUTO";
    cmd.Connection = con;
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = cmd;
    // Связывание объектов DbDataAdapter и DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Отображение автоматически сгенерированных команд DELETE
    Console.WriteLine("useColumnsForParameterNames = false");

 Console.WriteLine(builder.GetDeleteCommand(false).CommandText);
    Console.WriteLine();
    builder.RefreshSchema();
    Console.WriteLine("useColumnsForParameterNames = true");
    Console.WriteLine(builder.GetDeleteCommand(true).CommandText);
    Console.WriteLine();
    // Освобождение ресурсов
    builder.Dispose();
    // Закрытие подключения к БД
    con.Close();
  }
}

Результат выполнения примера:
```

```
useColumnsForParameterNames = false
DELETE FROM "SYSTEM"."AUTO" WHERE (((:param1 = 1 AND "MAKE" IS
 NULL)
OR ("MAKE" = :param2)) AND ("PERSONID" = :param3))

useColumnsForParameterNames = true
DELETE FROM "SYSTEM"."AUTO" WHERE (((:IsNull_MAKE = 1 AND "MAKE"
 IS NULL)
OR ("MAKE" = :Original_MAKE)) AND ("PERSONID"
 = :Original_PERSONID))
```

GetInsertCommand

Метод предоставляет автоматически сгенерированный для выполнения операций добавления записей в БД объект DbCommand.

Клиентское приложение может с помощью метода `GetInsertCommand` получить сгенерированный SQL-запрос INSERT и, при необходимости, модифицировать его (например, изменить значение CommandTimeout, а затем явно задать это значение для DbDataAdapter). В этом случае приложение должно явным образом вызвать метод `RefreshSchema`, если каким-либо образом изменится SELECT-запрос в свойстве DbDataAdapter.SelectCommand. В противном случае метод `GetInsertCommand` будет продолжать использовать сведения от предыдущих команд, которые могут оказаться неверными.

Первоначально SQL-команды создаются, когда приложение вызывает метод `Update`, либо метод `GetInsertCommand`.

Синтаксис

```
public DbCommand GetInsertCommand();
```

Возвращаемое значение

Автоматически созданный объект DbCommand, содержащий текст SQL-запроса для добавления записей в таблицу БД.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
 "DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8";
    con.Open();
    DbCommand cmd = factory.CreateCommand();
    cmd.CommandText = "select MAKE, PERSONID from AUTO";
    cmd.Connection = con;
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = cmd;
    // Связывание объектов DbDataAdapter и DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Отображение автоматически сгенерированной команды INSERT
    Console.WriteLine(builder.GetInsertCommand().CommandText);
    // Освобождение ресурсов
    builder.Dispose();
    // Закрытие подключения к БД
    con.Close();
  }
}

Результат выполнения примера:

INSERT INTO "SYSTEM"."AUTO" ("MAKE", "PERSONID") VALUES
 (:param1, :param2)
```

GetInsertCommand(Boolean)

Метод предоставляет автоматически сгенерированный для выполнения операций добавления записей в таблицу БД параметризованный объект DbCommand.

Синтаксис

```
public DbCommand
GetInsertCommand(bool useColumnsForParameterNames);
```

`useColumnsForParameterNames` – задает правило именования параметров:

- true – имена параметров должны совпадать с именами столбцов (а если столбец неименованный, типа select to_char(id) …), то параметр для такого столбца не генерируется;
- false – имена параметров должны совпадать с их порядковыми номерами в запросе (:p1, :p2 и т. д.).

Значение по умолчанию false.

Для получения параметров с именами столбцов (useColumnsForParameterNames=true) необходимы дополнительные условия:

- должно быть задано значение свойства ParameterNameMaxLength, возвращаемое методом `GetSchema`, и присутствующее в коллекции DataSourceInformation; указанная в этом свойстве длина должна быть не меньше длины сгенерированного имени параметра;
- сгенерированное имя параметра соответствует критериям, заданным в свойстве ParameterNamePattern, возвращаемом методом `GetSchema` и присутствующее в коллекции DataSourceInformation;
- должно быть задано значение свойства ParameterMarkerFormat, возвращаемое методом `GetSchema`, и присутствующее в коллекции DataSourceInformation.

Возвращаемое значение

Автоматически созданный объект DbCommand, содержащий параметризованный текст SQL-запроса для добавления строк в таблицу БД.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
 "DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8";
    con.Open();
    DbCommand cmd = factory.CreateCommand();
    cmd.CommandText = "select MAKE, PERSONID from AUTO";
    cmd.Connection = con;
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = cmd;
    // Связывание объектов DbDataAdapter и DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Отображение автоматически сгенерированных команд INSERT
    Console.WriteLine("useColumnsForParameterNames = false");
```

```
 Console.WriteLine(builder.GetInsertCommand(false).CommandText);
    Console.WriteLine();
    Console.WriteLine("useColumnsForParameterNames = true");
    Console.WriteLine(builder.GetInsertCommand(true).CommandText);
    Console.WriteLine();
    // Освобождение ресурсов
    builder.Dispose();
    // Закрытие подключения к БД
    con.Close();
  }
}

Результат выполнения примера:

useColumnsForParameterNames = false
INSERT INTO "SYSTEM"."AUTO" ("MAKE", "PERSONID") VALUES
 (:param1, :param2)

useColumnsForParameterNames = true
INSERT INTO "SYSTEM"."AUTO" ("MAKE", "PERSONID") VALUES
 (:MAKE, :PERSONID)
```

GetUpdateCommand

Метод предоставляет автоматически сгенерированный для выполнения операции обновления записей в БД объект DbCommand.

Клиентское приложение может с помощью метода `GetUpdateCommand` получить сгенерированный SQL-запрос UPDATE и, при необходимости, модифицировать его (например, изменить значение CommandTimeout, а затем явно задать это значение для DbDataAdapter). В этом случае приложение должно явным образом вызвать метод `RefreshSchema`, если каким-либо образом изменится SELECT-запрос в свойстве DbDataAdapter.SelectCommand. В противном случае метод `GetUpdateCommand` будет продолжать использовать сведения от предыдущих команд, которые могут оказаться неверными.

Первоначально SQL-команды создаются, когда приложение вызывает метод `Update`, либо метод `GetUpdateCommand`.

Синтаксис

```
public DbCommand GetUpdateCommand();
```

Возвращаемое значение

Автоматически созданный объект DbCommand, содержащий текст SQL-запроса для обновления записей в таблице БД.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
 "DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8";
    con.Open();
    DbCommand cmd = factory.CreateCommand();
    cmd.CommandText = "select MAKE, PERSONID from AUTO";
    cmd.Connection = con;
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = cmd;
    // Связывание объектов DbDataAdapter и DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Отображение автоматически сгенерированной команды UPDATE
    Console.WriteLine(builder.GetUpdateCommand().CommandText);
    // Освобождение ресурсов
    builder.Dispose();
    // Закрытие подключения к БД
    con.Close();
  }
}

Результат выполнения примера:

UPDATE "SYSTEM"."AUTO" SET "MAKE" = :param1, "PERSONID" = :param2
WHERE (((:param3 = 1 AND "MAKE" IS NULL) OR ("MAKE" = :param4))
 AND ("PERSONID" = :param5))
```

GetUpdateCommand(Boolean)

Предоставляет автоматически сгенерированный для выполнения операции обновления записи в таблице БД параметризованный объект DbCommand.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Синтаксис

```
public DbCommand
GetUpdateCommand(bool useColumnsForParameterNames);
```

`useColumnsForParameterNames` – задает правило именования параметров:

- true – имена параметров должны совпадать с именами столбцов (а если столбец неименованный, типа select to_char(id) …), то параметр для такого столбца не генерируется;
- false – имена параметров должны совпадать с их порядковыми номерами в запросе (:p1, :p2 и т. д.).

Значение по умолчанию false.

Для получения параметров с именами столбцов (useColumnsForParameterNames=true) необходимы дополнительные условия:

- должно быть задано значение свойства ParameterNameMaxLength, возвращаемое методом `GetSchema`, и присутствующее в коллекции DataSourceInformation; указанная в этом свойстве длина должна быть не меньшее длины сгенерированного имени параметра;
- сгенерированное имя параметра соответствует критериям, заданным в свойстве ParameterNamePattern, возвращаемом методом `GetSchema`, и присутствующее в коллекции DataSourceInformation;
- должно быть задано значение свойства ParameterMarkerFormat, возвращаемое методом `GetSchema`, и присутствующее в коллекции DataSourceInformation.

Возвращаемое значение

Автоматически созданный объект DbCommand, содержащий параметризованный текст SQL-запроса для обновления записей в таблице БД.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
```

```
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
 "DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8";
    con.Open();
    DbCommand cmd = factory.CreateCommand();
    cmd.CommandText = "select MAKE, PERSONID from AUTO";
    cmd.Connection = con;
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = cmd;
    // Связывание объектов DbDataAdapter и DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Отображение автоматически сгенерированных команд UPDATE
    Console.WriteLine("useColumnsForParameterNames = false");

 Console.WriteLine(builder.GetUpdateCommand(false).CommandText);
    Console.WriteLine();
    Console.WriteLine("useColumnsForParameterNames = true");
    Console.WriteLine(builder.GetUpdateCommand(true).CommandText);
    Console.WriteLine();
    // Освобождение ресурсов
    builder.Dispose();
    // Закрытие подключения к БД
    con.Close();
  }
}
```

Результат выполнения примера:

```
useColumnsForParameterNames = false
UPDATE "SYSTEM"."AUTO" SET "MAKE" = :param1, "PERSONID" = :param2
WHERE (((:param3 = 1 AND "MAKE" IS NULL) OR ("MAKE" = :param4))
 AND ("PERSONID" = :param5))

useColumnsForParameterNames = true
UPDATE "SYSTEM"."AUTO" SET "MAKE" = :MAKE, "PERSONID" = :PERSONID
WHERE (((:IsNull_MAKE = 1 AND "MAKE" IS NULL) OR ("MAKE"
 = :Original_MAKE))
AND ("PERSONID" = :Original_PERSONID))
```

QuoteIdentifier

Метод предоставляет обрамленный прямыми двойными кавычками указанный идентификатор в спецификации объекта БД. Если идентификатор уже обрамлен кавычками, ещё одни кавычки не добавляются.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Обрамление делается теми символами, которые установлены в свойствах QuotePrefix и QuoteSuffix.

Синтаксис

```
public override
string QuoteIdentifier(string unquotedIdentifier);
```

`unquotedIdentifier` – идентификатор объекта БД без обрамления.

Возвращаемое значение

Заданный идентификатор, обрамленный прямыми двойными кавычками.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandBuilderSample
{
  static void Main()
  {
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    string column_name1 = "id";
    string column_name2 = "name";
    string schema_name = "Гл. Бухгалтер";
    string table_name = "Материальные ценности";
    string commandText = "Create view XXX as select " +
      builder.QuoteIdentifier(column_name1) + ", " +
      builder.QuoteIdentifier(column_name2) + " FROM " +
      builder.QuoteIdentifier(schema_name) + "." +
      builder.QuoteIdentifier(table_name);
    Console.WriteLine(commandText);
  }
}
```

```
Результат выполнения примера:

Create view XXX as select "id", "name" FROM "Гл.
 Бухгалтер"."Материальные ценности"
```

RefreshSchema

Метод обновляет автоматически сгенерированные SQL-операторы, связанные с указанным объектом DbCommandBuilder.

Если текст SQL-запроса в DbDataAdapter был изменен, то для приведения в соответствие с ним автоматически сгенерированных объектов GetDeleteCommand, GetInsertCommand, GetUpdateCommand надо выполнить метод `RefreshSchema`, потому что DbDataAdapter не получает сигнал об изменении связанного с ним свойства SelectCommand.CommandText.

Вызов метода `RefreshSchema` не приводит к немедленному обновлению сгенерированных команд – просто устанавливается флаг в DbCommandBuilder, что логика генерации команд изменилась. DbCommandBuilder будет реально обновлять сгенерированные команды в момент вызова метода `Update` объекта DbDataAdapter или при вызове одного из методов `Get/Delete/Insert/UpdateCommand` объекта DbCommandBuilder.

Синтаксис

```
public virtual void RefreshSchema();
```

Возвращаемое значение

Значение типа void.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
 "DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8";
    con.Open();
    DbCommand cmd = factory.CreateCommand();
    cmd.CommandText = "select MAKE, PERSONID from AUTO";
    cmd.Connection = con;
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = cmd;
    // Связывание объектов DbDataAdapter и DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    builder.DataAdapter = adapter;
    // Отображение автоматически сгенерированных команд
    Console.WriteLine(builder.GetInsertCommand().CommandText);
    Console.WriteLine(builder.GetUpdateCommand().CommandText);
    Console.WriteLine(builder.GetDeleteCommand().CommandText);
    Console.WriteLine();
    // Изменение SELECT-запроса
    cmd.CommandText = "select MODEL, PERSONID from AUTO";
    builder.RefreshSchema();
    // Отображение новых команд
    Console.WriteLine(builder.GetInsertCommand().CommandText);
    Console.WriteLine(builder.GetUpdateCommand().CommandText);
    Console.WriteLine(builder.GetDeleteCommand().CommandText);
    Console.WriteLine();
    // Освобождение ресурсов
    builder.Dispose();
    // Закрытие подключения к БД
    con.Close();
  }
}
```

UnquoteIdentifier

Метод предоставляет указанный идентификатор в спецификации объектов БД без обрамляющих его кавычек.

Удаляются обрамления, которые установлены в свойствах QuotePrefix и QuoteSuffix.

Если задан идентификатор без обрамляющих кавычек, он возвращается в первоначальном виде.

Синтаксис

```
public override
string UnquoteIdentifier(string quotedIdentifier);
```

`quotedIdentifier` – идентификатор объекта БД с обрамляющими кавычками.

Возвращаемое значение

Идентификатор объекта БД, из которого удалены обрамляющие кавычки.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class CommandBuilderSample
{
  static void Main()
  {
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    string quotedIdentifier = "\"AUTO\"";
    string unquotedIdentifier =
 builder.UnquoteIdentifier(quotedIdentifier);
    Console.WriteLine(unquotedIdentifier);
  }
}
```

Класс DbConnectionStringBuilder

Класс `DbConnectionStringBuilder` предназначен для создания и управления содержимым строк подключения, которые используются классом `DbConnection`.

Класс `DbConnectionStringBuilder` предоставляет ограниченный внутренний набор пар «ключ/значение». Если в строке подключения отсутствует значение какого-либо параметра, то в процессе соединения с ЛИНТЕР-сервером используется значение по умолчанию. При извлечении свойства ConnectionString того или иного объекта строка содержит только такие пары «ключ/значение», в которых «значение» отличается от стандартного значения.

Ключи строки подключения:

- **DataSource**=<имя сервера>;
- **User ID**=<имя пользователя>;
- **Password**=<пароль>;
- **Persist Security Info**=<аутентификации>;
- **IsolationLevel**=<уровень изоляции>;
- **Autocommit**=<режим канала>;
- **Minimum Pool Size**=<число>;
- **Maximum Pool Size**=<число>;
- **ConnectionTimeout**=<число>;
- **Channel Priority**=<число>;
- **Messages Language**=<язык сообщений>;
- **Charset**=<кодировка>.

Формат строки подключения приведён в пункте «ConnectionString».

Конструкторы класса `DbConnectionStringBuilder` приведены в таблице 35.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Таблица 35. Конструкторы класса DbConnectionStringBuilder

Конструктор	Описание
LinterDbConnectionStringBuilder	Создает новый экземпляр класса LinterDbConnectionStringBuilder с параметрами соединения по умолчанию.
LinterDbConnectionStringBuilder(String)	Создает новый экземпляр класса LinterDbConnectionStringBuilder на основе заданной строки подключения.

Свойства класса `DbConnectionStringBuilder` приведены в таблице 36.

Таблица 36. Свойства класса DbConnectionStringBuilder

Свойство	Описание
BrowsableConnectionString	Предоставляет/устанавливает видимость свойства ConnectionString в конструкторах.
ConnectionString	Предоставляет/устанавливает строку подключения, связанную с DbConnectionStringBuilder.
Count	Предоставляет текущее количество ключей, содержащихся в строке подключения.
DataSource	Предоставляет/устанавливает имя сервера источника данных, к которому осуществляется подключение.
IsFixedSize	Предоставляет информацию о возможности добавления новых ключей в строку подключения.
IsReadOnly	Предоставляет информацию о возможности изменения объекта LinterDbConnectionStringBuilder (т.е. является коллекция «только для чтения» или нет).
Item	Предоставляет/устанавливает значение заданного ключа строки подключения.
Keys	Предоставляет список ключей, задействованных в строке подключения.
Password	Предоставляет/устанавливает пароль текущего пользователя БД источника данных.
PersistSecurityInfo	Предоставляет или устанавливает режим отображения пароля в строке подключения.
UserID	Предоставляет/устанавливает имя пользователя, которое должно использоваться при подключении к серверу источника данных.
Values	Предоставляет массив значений всех ключей текущей строки подключения.

Методы класса `DbConnectionStringBuilder` приведены в таблице 37.

Таблица 37. Методы класса DbConnectionStringBuilder

Метод	Описание
Add	Добавляет параметр подключения (пару «ключ/значение») в текущую строку подключения.

Метод	Описание
AppendKeyValuePair(StringBuilder, String, String)	Добавляет пару «ключ/значение» в массив элементов «ключ/значение» (в объект StringBuilder)
AppendKeyValuePair(StringBuilder, String, String, Boolean)	Добавляет элемент «ключ/значение» в указанную строку подключения с заданным разделителем элементов «ключ/значение».
Clear	Удаляет содержимое текущей строки подключения.
ContainsKey	Проверяет поддержку ADO.NET-провайдером в строке подключения указанного ключа.
EquivalentTo	Сравнивает две строки подключения.
Remove	Удаляет элемент «ключ/значение» из строки подключения.
ShouldSerialize	Проверяет наличие указанного ключа в строке подключения.
TryGetValue	Предоставляет значение указанного ключа строки подключения.

Конструкторы

ADO.NET-провайдер СУБД ЛИНТЕР обеспечивает поддержку двух конструкторов класса `LinterDbConnectionStringdBuilder`.

LinterDbConnectionStringBuilder

Синтаксис

```
public LinterDbConnectionStringBuilder();
```

Возвращаемое значение

Новый экземпляр класса `LinterDbConnectionStringBuilder` с параметрами соединения по умолчанию.

LinterDbConnectionStringBuilder(String)

Синтаксис

```
public LinterDbConnectionStringBuilder(string connectionString);
```

`connectionString` – строка соединения с БД в формате ADO.NET-провайдера СУБД ЛИНТЕР.

Возвращаемое значение

Новый экземпляр класса `LinterDbConnectionStringBuilder` с заданными параметрами соединения.

Если некоторые элементы строки соединения известны заранее, их можно сохранить в файле конфигурации и во время выполнения получить для построения полной строки соединения. Например, в отличие от имени сервера, имя БД может быть известно заранее.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Данный конструктор принимает в качестве аргумента значение типа String, что позволяет использовать частичную строку соединения, которую впоследствии пользователь может дополнить. Частичную строку соединения можно сохранить в файле конфигурации и получить во время выполнения клиентского приложения.

Пример

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    LinterDbConnectionStringBuilder builder = new
 LinterDbConnectionStringBuilder(
      "DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8");
    Console.WriteLine(builder.DataSource);
    Console.WriteLine(builder.UserID);
    Console.WriteLine(builder.Password);
  }
}
```

Свойства

BrowsableConnectionString

Устанавливает видимость или предоставляет информацию о видимости свойства ConnectionString в конструкторах.

Декларация

```
[BrowsableAttribute(false)] public bool BrowsableConnectionString
 {get; set;};
```

Значение свойства

Значение true, если строка подключения является видимой в конструкторах, в противном случае – значение false.

Значение по умолчанию true.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
```

```
using System.Data.Common;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    DbConnectionStringBuilder builder =
      factory.CreateConnectionStringBuilder();
    Console.WriteLine(builder.BrowsableConnectionString);
  }
}
```

ConnectionString

Предоставляет или задает строку подключения, связанную с DbConnectionStringBuilder.

Формат строки:

<параметр>;<параметр>; …

<параметр>::=<ключ>=<значение>

Например:

```
"Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8"
```

Класс DbConnectionStringBuilder не проверяет пары «ключ/значение», связанные со строкой подключения, хотя классы, которые наследуются из него, это могут делать.

Свойство ConnectionString класса DbConnectionStringBuilder действует обычно как механизм для создания и анализа списков пар «ключ/значение», в которых в качестве разделителя используется точка с запятой и которые отделяются друг от друга знаками равенства. Свойство не выполняет семантическую проверку устанавливаемой строки подключения.

После добавления (изменения) параметра строки подключения свойство ConnectionString будут отображать данные изменения.

Декларация

```
public string ConnectionString {get; set;};
```

Значение свойства

Текущая строка подключения, созданная по парам «ключ/значение», содержащимся в DbConnectionStringBuilder.

Значение по умолчанию – пустая строка.

Исключения

ArgumentException	Неизвестное имя ключа в строке подключения.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Пример

В примере демонстрируется возможное поведение свойства ConnectionString:

- создается строка подключения посредством добавления по одной паре «ключ/значение» в пустой объект DbConnectionStringBuilder;
- назначается полная строка подключения свойству ConnectionString экземпляра DbConnectionStringBuilder и изменяется единственная пара «ключ/значение» в данной строке;
- назначается недопустимая строка подключения свойству ConnectionString и показывается, что в этом случае выдается исключение.

Примечание
В приведенном пример используется пароль, чтобы продемонстрировать, как DbConnectionStringBuilder работает со строками подключения. В клиентских приложениях рекомендуется использовать проверку подлинности средствами ОС Windows. Если требуется использовать пароль, не следует добавлять жестко запрограммированный пароль в исходный текст клиентского приложения.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание объекта DbConnectionStringBuilder, и добавление
    // элементов в коллекцию ключ/значение
    DbConnectionStringBuilder builder =
      factory.CreateConnectionStringBuilder();
    builder.Add("Data Source", "LOCAL");
    builder.Add("User ID", "SYSTEM");
    builder.Add("Password", "MANAGER8");
    // Отображение строки подключения, которая сейчас будет
    // содержать все пары ключ/значение, разделенные точкой с
 запятой
    Console.WriteLine(builder.ConnectionString);
    Console.WriteLine();
    // Очистка объекта DbConnectionStringBuilder и установка
 полной
    // строки подключения для демонстрации того, как данный класс
    // выполняет синтаксический разбор строк подключения
    builder.Clear();
```

```
    builder.ConnectionString =
      "Data Source=TEST;User ID=admin;Password=passwd";
    // Класс DbConnectionStringBuilder выполнил разбор строки,
 поэтому
    // теперь можно работать с индивидуальными парами ключ/
значение
    builder["Data Source"] = "NEW";
    Console.WriteLine(builder.ConnectionString);
    Console.WriteLine();
    builder.Clear();
    try
    {
      // Присвоение некорректной строки подключения
      // генерирует исключение ArgumentException
      builder.ConnectionString = "xxx";
    }
    catch (ArgumentException)
    {
      Console.WriteLine("Некорректная строка подключения");
    }
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}
```

Count

Предоставляет текущее количество ключей, содержащихся в свойстве ConnectionString (в строке подключения).

Декларация

```
public virtual int Count {get; set;};
```

Значение свойства

Количество ключей, содержащихся в строке подключения экземпляра `DbConnectionStringBuilder` (значение типа System.Int32).

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание объекта DbConnctionStringBuilder
    DbConnectionStringBuilder builder =
      factory.CreateConnectionStringBuilder();
    builder.ConnectionString =
      "DataSource=LOCAL;UserID=SYSTEM;Password=MANGER";
    // Следующая команда должна отобразить "3" в окне консоли
    Console.WriteLine("Начальное количество ключей: " +
 builder.Count);
    builder.Add("Connection Timeout", 25);
    // Следующая команда должна отобразить новую строку
 подключения и
    // количество ключей (4) в окне консоли
    Console.WriteLine(builder.ConnectionString);
    Console.WriteLine("Новое количество ключей: " +
 builder.Count);
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}
```

DataSource

Предоставляет или устанавливает имя ЛИНТЕР-сервера, к которому осуществляется подключение.

Это свойство соответствует ключу «Data Source».

Если установлено пустое значение свойства DataSource, то при его изменении оно будет переопределено.

Декларация

```
public string DataSource {get; set;};
```

Значение свойства

Значение ключа «DataSource» в строке подключения или String.Empty, если ключ не задан.

Исключения

Отсутствуют.

Пример

В примере класс DbConnectionStringBuilder преобразует синонимы ключа «Data Source» в строке подключения в известный ключ.

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание объекта LinterDbConnectionStringBuilder
    LinterDbConnectionStringBuilder builder =
      new LinterDbConnectionStringBuilder();
    builder.ConnectionString =
      "Server=LOCAL;UserID=SYSTEM;Password=MANGER";
    // Отображение строки подключения, которая сейчас должна
    // содержать ключ "DataSource", соответствующий параметру
 "Server"
    Console.WriteLine(builder.ConnectionString);
    // Получение свойства DataSource
    Console.WriteLine("DataSource = " + builder.DataSource);
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}
```

IsFixedSize

Предоставляет информацию о возможности добавления новых ключей в строку подключения.

Декларация

```
public override bool IsFixedSize {get;};
```

Значение свойства

Значение true, если поддерживается строго ограниченный набор ключей, false – в противном случае.

Примечание
В текущей версии ADO.NET-провайдера СУБД ЛИНТЕР всегда возвращается значение true.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Исключения

Отсутствуют.

IsReadOnly

Предоставляет информацию о возможности изменения объекта LinterDbConnectionStringBuilder (т.е. является коллекция «только для чтения» или нет).

В коллекции, доступной только для чтения, не разрешается добавлять, удалять или изменять элементы после её создания.

Декларация

```
[BrowsableAttribute(false)]
public bool IsReadOnly {get;};
```

Значение свойства

Значение true, если объект LinterDbConnectionStringBuilder доступен только для чтения, в противном случае – значение false.

Значение по умолчанию false.

Примечание
В текущей версии ADO.NET-провайдера СУБД ЛИНТЕР всегда возвращается значение false.

Исключения

Отсутствуют.

Item

Предоставляет или устанавливает значение заданного ключа строки подключения.

Декларация

```
public override Object Item[string keyword] {get; set;};
```

`keyword` – имя ключа, значение которого требуется получить или установить.

Значение свойства

Текущее или установленное значение запрошенного ключа.

Исключения

`ArgumentException`	Попытка добавить несуществующий ключ.
`FormatException`	Недопустимое значение в строке подключения.
`ArgumentNullException`	Null-значение ключа.

Примеры

1)

```
// C#
using System;
```

```
using System.Data;
using System.Data.Common;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание объекта DbConnectionStringBuilder
    DbConnectionStringBuilder builder =
      factory.CreateConnectionStringBuilder();
    builder["Data Source"] = "LOCAL";
    builder["User ID"] = "SYSTEM";
    builder["Password"] = "MANAGER8";
    // Переопределение существующего значения Data Source
    builder["Data Source"] = "Test";
    Console.WriteLine(builder.ConnectionString);
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}
```

2) Создать строку подключения к ЛИНТЕР-серверу и выполнить подключение.

Если ЛИНТЕР-сервер недоступен, выбрать другой ЛИНТЕР-сервер (например, из конфигурационного файла nodetab), изменить строку подключения и подсоединиться к новому ЛИНТЕР-серверу

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание объекта LinterDbConnectionStringBuilder
    LinterDbConnectionStringBuilder builder =
      new LinterDbConnectionStringBuilder();
    builder["Data Source"] = "Test";
    builder["User ID"] = "SYSTEM";
    builder["Password"] = "MANAGER8";
    // Подключение к БД
    LinterDbConnection con = new LinterDbConnection();
```

```
    con.ConnectionString = builder.ConnectionString;
    try
    {
      con.Open();
      Console.WriteLine("Установлено соединение с сервером " +
        builder["Data Source"]);
    }
    catch (LinterSqlException ex)
    {
      Console.WriteLine("Ошибка при подключении к серверу " +
        builder["Data Source"]);
      Console.WriteLine(ex.Message);
      if ((ex.Number >= 1001 && ex.Number <= 1004) ||
          (ex.Number == 1069) ||
          (ex.Number >= 4000 && ex.Number <= 4999))
      {
        builder["Data Source"] = "LOCAL";
        con.ConnectionString = builder.ConnectionString;
        try
        {
          Console.WriteLine("Подключение к серверу " +
 builder["Data Source"]);
          con.Open();
          Console.WriteLine("Установлено соединение с сервером " +
            builder["Data Source"]);
        }
        catch (LinterSqlException ex2)
        {
          Console.WriteLine("Ошибка при подключении к серверу " +
            builder["Data Source"]);
          Console.WriteLine(ex2.Message);
        }
      }
    }
    // Освобождение ресурсов
    con.Close();
  }
}
```

Keys

Предоставляет список ключей, задействованных в объекте LinterDbConnectionStringBuilder.

Декларация

```
public override ICollection Keys {get;};
```

Значение свойства

Значение типа ICollection, содержащее список ключей объекта LinterDbConnectionStringBuilder.

Порядок значений в ICollection не определен, но аналогичен порядку связанных значений в ICollection, возвращенном свойством Values.

Исключения

Отсутствуют.

Пример

В примере создается новый объект DbConnectionStringBuilder.

Код реализует цикл по ICollection, возвращенному свойством Keys, и отображает пары «ключ/значение».

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание объекта DbConnectionStringBuilder
    DbConnectionStringBuilder builder =
      factory.CreateConnectionStringBuilder();
    builder["Data Source"] = "LOCAL";
    builder["User ID"] = "SYSTEM";
    builder["Password"] = "MANAGER8";
    // Просмотр коллекции ключей и отображение каждого ключа и
 значения
    foreach (string key in builder.Keys)
    {
      Console.WriteLine("{0}={1}", key, builder[key]);
    }
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Password

Предоставляет или устанавливает пароль текущего пользователя БД ЛИНТЕР-сервера.

Это свойство соответствует ключу «Password» в строке подключения.

Если установлено пустое значение, то при его изменении свойство Password будет переопределено.

Если значение не задано, то возвращается значение String.Empty.

Примечание
Для поддержания наивысшего уровня безопасности вместо указания пароля настоятельно рекомендуется при создании пользователя БД использовать опцию `IDENTIFIED BY SYSTEM`, которая устанавливает режим встроенной аутентификации средствами операционной системы.

Декларация

```
public string Password {get; set;};
```

Значение свойства

Значение свойства Password или String.Empty, если значение не предоставлено.

Исключения

Отсутствуют.

Пример

В примере демонстрируется:

- подсоединение к БД ЛИНТЕР-сервера с неправильным паролем;
- получение от СУБД соответствующего кода завершения;
- интерактивный ввод правильного пароля;
- замена его в строке подключения;
- повторное соединение с БД.

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание объекта LinterDbConnectionStringBuilder
    LinterDbConnectionStringBuilder builder =
      new LinterDbConnectionStringBuilder();
    builder.DataSource = "LOCAL";
```

```
    builder.UserID = "SYSTEM";
    // Подключение к БД
    LinterDbConnection con = new LinterDbConnection();
    con.ConnectionString = builder.ConnectionString;
    try
    {
      con.Open();
      Console.WriteLine("Установлено соединение с сервером");
    }
    catch (LinterSqlException ex)
    {
      Console.WriteLine("Ошибка при подключении к серверу:");
      Console.WriteLine(ex.Message);
      if (ex.Number == 1026)
      {
        Console.Write("Введите пароль: ");
        builder.Password = Console.ReadLine();
        con.ConnectionString = builder.ConnectionString;
        try
        {
          con.Open();
          Console.WriteLine("Установлено соединение с сервером");
        }
        catch (LinterSqlException ex2)
        {
          Console.WriteLine("Ошибка при подключении к серверу:");
          Console.WriteLine(ex2.Message);
        }
      }
    }
    // Освобождение ресурсов
    con.Close();
  }
}
```

PersistSecurityInfo

Предоставляет или устанавливает режим отображения пароля в строке подключения. Это свойство соответствует ключу «Persist Security Info» в строке подключения. Если установлено значение false (настоятельно рекомендуется), то пароль не отображается в строке подключения.

Если установлено значение true, то пароль отображается в строке подключения.

Значение по умолчанию false.

Декларация

```
public bool PersistSecurityInfo { get; set; }
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Значение свойства

Режим отображения пароля в строке подключения.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    LinterDbConnectionStringBuilder builder =
      new LinterDbConnectionStringBuilder();
    builder.DataSource = "LOCAL";
    builder.UserID = "SYSTEM";
    builder.Password = "MANAGER8";

    builder.PersistSecurityInfo = false;
    LinterDbConnection conn1 = new LinterDbConnection();
    conn1.ConnectionString = builder.ConnectionString;
    Console.WriteLine(conn1.ConnectionString);

    builder.PersistSecurityInfo = true;
    LinterDbConnection conn2 = new LinterDbConnection();
    conn2.ConnectionString = builder.ConnectionString;
    Console.WriteLine(conn2.ConnectionString);
  }
}
```

Результат выполнения примера

```
DataSource=LOCAL;UserID=SYSTEM;PersistSecurityInfo=False
DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8;PersistSecurityInfo=True
```

UserID

Предоставляет или устанавливает имя пользователя, которое должно использоваться при подключении к ЛИНТЕР-серверу.

Это свойство соответствует ключу «UserId» в строке подключения.

Если установлено пустое значение, то при его изменении свойство UserId будет переопределено.

Примечание
В СУБД ЛИНТЕР длина пароля не может быть больше 18 символов. Если задан более длинный пароль, то ADO.NET-провайдер СУБД ЛИНТЕР исключение не генерирует, а использует первые 18 символов.

Декларация

```
public string UserID {get; set};
```

Значение свойства

Значение свойства UserID или String.Empty, если значение не установлено.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание объекта LinterDbConnectionStringBuilder
    LinterDbConnectionStringBuilder builder =
      new LinterDbConnectionStringBuilder();
    builder.DataSource = "LOCAL";
    builder.UserID = "SYSTEM";
    // Подключение к БД
    LinterDbConnection con = new LinterDbConnection();
    con.ConnectionString = builder.ConnectionString;
    try
    {
      con.Open();
      Console.WriteLine("Установлено соединение с сервером");
    }
    catch (LinterSqlException ex)
    {
      Console.WriteLine("Ошибка при подключении к серверу:");
      Console.WriteLine(ex.Message);
```

```
      if (ex.Number == 1026)
      {
        Console.Write("Введите пароль: ");
        builder.Password = Console.ReadLine();
        con.ConnectionString = builder.ConnectionString;
        try
        {
          con.Open();
          Console.WriteLine("Установлено соединение с сервером");
        }
        catch (LinterSqlException ex2)
        {
          Console.WriteLine("Ошибка при подключении к серверу:");
          Console.WriteLine(ex2.Message);
        }
      }
    }
    // Освобождение ресурсов
    con.Close();
  }
}
```

Values

Предоставляет массив значений всех ключей текущего объекта DbConnectionStringBuilder.

Порядок значений в массиве не определен, но аналогичен порядку связанных ключей в ICollection, возвращенном свойством Keys. Поскольку в каждом экземпляре DbConnectionStringBuilder может содержаться один и тот же ограниченный набор ключей, то свойство Values всегда возвращает значения, соответствующие этому ограниченному набору ключей.

Декларация

```
public override ICollection Values {get;};
```

Значение свойства

Массив значений ключей ICollection, которые установлены в объекте DbConnectionStringBuilder.

Исключения

Отсутствуют.

Примеры

1) В примере сначала создается новый объект DbConnectionStringBuilder, а затем выполняется перебор всех пар «ключ/значение» в этом объекте (т.е. в строке подключения).

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание объекта DbConnectionStringBuilder
    DbConnectionStringBuilder builder =
      factory.CreateConnectionStringBuilder();
    builder.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Перебор всех значений и отображение каждого значения
    foreach (object value in builder.Values)
    {
      Console.WriteLine(value);
    }
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}
```

2) Просмотр и отображение ключей строки подключения и их значений.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание объекта DbConnectionStringBuilder
    DbConnectionStringBuilder builder =
      factory.CreateConnectionStringBuilder();
    builder["Data Source"] = "LOCAL";
    builder["User ID"] = "SYSTEM";
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    builder["Password"] = "MANAGER8";
    // Просмотр коллекции ключей и отображение каждого ключа и
 значения
    foreach (string key in builder.Keys)
    {
      Console.WriteLine("{0}={1}", key, builder[key]);
    }
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}
```

Методы

См. также раздел Общие свойства и методы классов ADO.NET-провайдера.

Add

Добавляет параметр подключения (пару «ключ/значение») в текущий объект DbConnectionStringBuilder.

Вызов метода `Add` с именем ключа, равным NULL, приводит к выдаче исключения ArgumentNullException.

Вызов метода `Add` со значением ключа, равным NULL, приводит к удалению из строки подключения пары «ключ/значение».

Строка подключения анализируется с помощью алгоритма «по последнему значению», то есть если в строке пара «ключ/значение» встречается несколько раз, то используется самое последнее значение.

Выполняются проверки на допустимые пары «ключ-значение», и недопустимая пара вызывает исключение.

При добавлении дубликата ключа выполняется изменение значения ключа.

Примечание
Свойство Item можно также использовать для установки значения ключа, например, `myCollection["myKey"] = myValue`.

Синтаксис

```
public void Add(string keyword, Object value);
```

`keyword` – имя добавляемого ключа.

`value` – значение добавляемого ключа.

Возвращаемое значение

Значение типа void.

Исключения

ArgumentNullException	Null-значение ключа.
FormatException	Попытка добавить неподдерживаемый ключ.
NotSupportedException	Возможные причины: • объект DbConnectionStringBuilder доступен только для чтения; • объект DbConnectionStringBuilder имеет фиксированный размер.

Примеры

1) Переопределение существующего ключа.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    try
    {
      // Создание фабрики классов провайдера
      DbProviderFactory factory =

 DbProviderFactories.GetFactory("System.Data.LinterClient");
      // Создание объекта DbConnectionStringBuilder
      DbConnectionStringBuilder builder =
        factory.CreateConnectionStringBuilder();
      builder.Add("Data Source", "LOCAL");
      builder.Add("User ID", "SYSTEM");
      builder.Add("Password", "MANAGER8");
      // Переопределение существующего значения "User ID"
      builder.Add("User ID", "SYS");
      // Следующая команда генерирует исключение
 ArgumentNullException
      // builder.Add(null, "некоторое значение");
      Console.WriteLine(builder.ConnectionString);
    }
    catch (ArgumentNullException)
    {
      Console.WriteLine("Не допускается ключ с именем null");
    }
    Console.WriteLine();
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}
```

2) Добавление нового ключа.

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    try
    {
      // Создание фабрики классов провайдера
      DbProviderFactory factory =

 DbProviderFactories.GetFactory("System.Data.LinterClient");
      // Создание объекта DbConnectionStringBuilder
      DbConnectionStringBuilder builder =
        factory.CreateConnectionStringBuilder();
      // Следующая команда генерирует исключение ArgumentException
      builder.Add("неизвестный ключ", "некоторое значение");
    }
    catch (ArgumentException)
    {
      Console.WriteLine("Неизвестный ключ");
    }
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}
```

3) Обработка дополнительного значения ключа Data Source.

Результат выполнения примера показывает, что объект DbConnectionStringBuilder правильно выполняет обработку ключа путем экранирования дополнительного значения, заключенного в двойные кавычки, вместо того чтобы добавить его в строку подключения в качестве новой пары «ключ-значение».

```
// C#
using System;
using System.Data;
using System.Data.Common;
```

```
class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание объекта DbConnectionStringBuilder
    DbConnectionStringBuilder builder =
      factory.CreateConnectionStringBuilder();
    builder.Add("Data Source", "LOCAL;NewValue=Bad");
    builder.Add("User ID", "SYSTEM");
    builder.Add("Password", "MANAGER8");
    Console.WriteLine(builder.ConnectionString);
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}
```

```
Результат выполнения примера:

DataSource="LOCAL;NewValue=Bad";UserID=SYSTEM;Password=MANAGER8
```

AppendKeyValuePair(StringBuilder, String, String)

Метод добавляет пару «ключ/значение» в указанный объект StringBuilder.

Этот метод позволяет с помощью объекта StringBuilder создать массив пар «ключ/значение» без дополнительных издержек, связанных с созданием и поддержкой экземпляра класса DbConnectionStringBuilder.

Метод `AppendKeyValuePair` правильно форматирует ключ и его значение и добавляет новую строку к заданному объекту StringBuilder.

Синтаксис

```
public static void AppendKeyValuePair(
  StringBuilder builder,
  string keyword,
  string value
);
```

`builder` – объект StringBuilder, в который надо добавить пару «ключ/значение».

`keyword` – имя добавляемого ключа.

`value` – значение добавляемого ключа.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Возвращаемое значение

Значение типа void.

Исключения

`ArgumentNullException`	Null-значение параметра builder или keyword.
`ArgumentException`	Недопустимое значение параметра builder или keyword.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Text;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание объекта StringBuilder
    StringBuilder builder = new StringBuilder();
    DbConnectionStringBuilder.AppendKeyValuePair(builder, "Data
 Source", "LOCAL");
    DbConnectionStringBuilder.AppendKeyValuePair(builder, "User
 ID",
      "ado.net user;");
    DbConnectionStringBuilder.AppendKeyValuePair(builder,
 "Password",
      "ado.net password;");
    // Отображение полученной строки подключения
    Console.WriteLine(builder.ToString());
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}

Результат выполнения примера:

Data Source=LOCAL;User ID="ado.net user;";Password="ado.net
 password;"
```

AppendKeyValuePair(StringBuilder, String, String, Boolean)

Метод добавляет пару «ключ/значение» в указанную строку подключения с заданным разделителем пар «ключ/значение».

Метод позволяет с помощью объекта StringBuilder создать коллекцию пар «ключ/значение» без дополнительных издержек, связанных с созданием и поддержкой экземпляра `DbConnectionStringBuilder`. Метод `AppendKeyValuePair` правильно форматирует символьную строку «ключ/значение» и добавляет её к указанному объекту StringBuilder.

Синтаксис

```
public static void AppendKeyValuePair(
  StringBuilder builder,
  string keyword,
  string value,
  bool useOdbcRules
);
```

`builder` – объект StringBuilder, в который надо добавить пару «ключ/значение».

`keyword` – имя добавляемого ключа.

`value` – значение добавляемого ключа.

`UseOdbcRules` – тип разделителя:

- true – использовать фигурные скобки {} для разделения полей;
- false – использовать двойные кавычки.

Возвращаемое значение

Значение типа void.

Исключения

`ArgumentNullException`	Null-значение параметра builder или keyword.
`ArgumentException`	Недопустимое значение параметра builder или keyword.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Text;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание объекта StringBuilder
```

```
    StringBuilder builder = new StringBuilder();
    DbConnectionStringBuilder.AppendKeyValuePair(builder, "Dsn",
      "Linter 6.0 Unicode", true);
    DbConnectionStringBuilder.AppendKeyValuePair(builder, "Uid",
      "odbc user;", true);
    DbConnectionStringBuilder.AppendKeyValuePair(builder, "Pwd",
      "odbc password;", true);
    // Отображение полученной строки подключения
    Console.WriteLine(builder.ToString());
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}

Результат выполнения примера:

Dsn=Linter 6.0 Unicode;Uid={odbc user;};Pwd={odbc password;}
```

Clear

Метод удаляет содержимое текущей строки подключения экземпляра объекта DbConnectionStringBuilder, т.е. удаляет все пары «ключ/значение» и сбрасывает все соответствующие свойства. При этом задается нулевое значение для свойства Count и пустая строка для свойства ConnectionString.

Синтаксис

```
public override void Clear();
```

Возвращаемое значение

Значение типа void.

Исключения

Отсутствуют.

Примеры

1)

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ConnectionStringBuilderSample
{
  static void Main()
  {
```

```
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание объекта DbConnectionStringBuilder
    DbConnectionStringBuilder builder =
      factory.CreateConnectionStringBuilder();
    builder["Data Source"] = "LOCAL";
    builder["User ID"] = "SYSTEM";
    builder["Password"] = "MANAGER8";
    Console.WriteLine("Исходная строка подключения: " +
 builder.ConnectionString);
    builder.Clear();
    Console.WriteLine("Количество элементов после вызова Clear = "
 +
      builder.Count);
    Console.WriteLine("Очищенная строка подключения: " +
      builder.ConnectionString);
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}
```

2)Сформировать строку подключения к основному ЛИНТЕР-серверу. Получить код завершения «Сервер недоступен», очистить строку подключения, сформировать её заново для подключения к резервному ЛИНТЕР-серверу.

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание объекта LinterDbConnectionStringBuilder
    LinterDbConnectionStringBuilder builder =
      new LinterDbConnectionStringBuilder();
    builder.DataSource = "MAIN";
    builder.UserID = "MAIN_UID";
    builder.Password = "MAIN_PWD";
    // Подключение к БД
    LinterDbConnection con = new LinterDbConnection();
    con.ConnectionString = builder.ConnectionString;
    try
    {
```

```
      con.Open();
      Console.WriteLine("Установлено соединение с главным
 сервером");
    }
    catch (LinterSqlException ex)
    {
      Console.WriteLine("Ошибка при подключении к главному
 серверу");
      Console.WriteLine(ex.Message);
      if ((ex.Number >= 1001 && ex.Number <= 1004) ||
          (ex.Number == 1069) ||
          (ex.Number >= 4000 && ex.Number <= 4999))
      {
        builder.Clear();
        builder.DataSource = "RESERV";
        builder.UserID = "RESERV_UID";
        builder.Password = "RESERV_PWD";
        con.ConnectionString = builder.ConnectionString;
        try
        {
          Console.WriteLine("Подключение к резервному
 серверу...");
          con.Open();
          Console.WriteLine("Установлено соединение с резервным
 сервером");
        }
        catch (LinterSqlException ex2)
        {
          Console.WriteLine("Ошибка при подключении к резервному
 серверу");
          Console.WriteLine(ex2.Message);
        }
      }
    }
    // Освобождение ресурсов
    con.Close();
  }
}
```

ContainsKey

Метод проверяет, поддерживает ли ADO.NET-провайдер СУБД ЛИНТЕР в строке подключения указанный ключ.

Синтаксис

```
public override bool ContainsKey(string keyword);
```

`keyword` – имя проверяемого ключа.

Возвращаемое значение

Значение true, если ключ поддерживается ADO.NET-провайдером, false – в противном случае.

Исключения

`ArgumentNullException`	Null-значение параметра keyword.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание объекта DbConnectionStringBuilder
    DbConnectionStringBuilder builder =
      factory.CreateConnectionStringBuilder();
    builder["Data Source"] = "LOCAL";
    builder["User ID"] = "SYSTEM";
    builder["Password"] = "MANAGER8";
    Console.WriteLine("Строка подключения = " +
 builder.ConnectionString);
    // Для ключей, которые поддерживаются, возвращается true
    Console.WriteLine(builder.ContainsKey("Server"));
    // Сравнение регистронезависимое, синонимы автоматически
 конвертируются в
    // известные обозначения
    Console.WriteLine(builder.ContainsKey("Database"));
    // Для поддерживаемых ADO.NET-провайдером, но не установленных
 в строке
    // подключения ключей возвращается true
    Console.WriteLine(builder.ContainsKey("Max Pool Size"));
    // Для неподдерживаемых ADO.NET-провайдером ключей
 возвращается false
    Console.WriteLine(builder.ContainsKey("MyKey"));
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
  }
}
```

В примере отображается следующий текст в окне консоли:

Строка подключения = DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8
True True True False

EquivalentTo

Метод сравнивает две строки подключения.

При сравнении имен ключей регистр не учитывается; сравнение значений осуществляется с учетом регистра.

При сравнении строк подключения порядок пар «ключ/значение» не учитывается, однако различный порядок может оказывать влияние на пул подключений, основанный на этих строках подключения.



Примечание
В текущей версии ADO.NET-провайдера СУБД ЛИНТЕР пул подключений не поддерживается.

Синтаксис

```
public virtual bool EquivalentTo(DbConnectionStringBuilder
 connectionStringBuilder);
```

`connectionStringBuilder` – строка подключения, которая сравнивается с текущей строкой подключения объекта DbConnectionStringBuilder.

Возвращаемое значение

Значение true, если пары «ключ/значение» являются одинаковыми, независимо от их порядка, false – в противном случае.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
```

```
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    DbConnectionStringBuilder builder1 =
      factory.CreateConnectionStringBuilder();
    builder1.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    Console.WriteLine("builder1 = " + builder1.ConnectionString);
    DbConnectionStringBuilder builder2 =
      factory.CreateConnectionStringBuilder();
    builder2.ConnectionString =
      "User ID=SYSTEM;PASSWORD=MANAGER8;Data Source=LOCAL";
    Console.WriteLine("builder2 = " + builder2.ConnectionString);
    DbConnectionStringBuilder builder3 =
      factory.CreateConnectionStringBuilder();
    builder3.ConnectionString =
      "User ID=SYSTEM;PASSWORD=MANAGER8;Data Source=SERV1";
    Console.WriteLine("builder3 = " + builder3.ConnectionString);
    // Объекты builder1 и builder2 содержат одинаковые пары ключ/
значение,
    // но установлены в разном порядке и в разном регистре ключей.
    // Строки подключения равнозначны
    Console.WriteLine("builder1.EquivalentTo(builder2) = " +
      builder1.EquivalentTo(builder2).ToString());
    // Объекты builder2 и builder3 содержат пары ключ/значение,
    // установленные в совпадающем порядке, но значения ключей
 отличаются,
    // поэтому строки подключения не равнозначны
    Console.WriteLine("builder2.EquivalentTo(builder3) = " +
      builder2.EquivalentTo(builder3).ToString());
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
    Console.ReadLine();
  }
}
```

```
Результат выполнения примера:

builder1 = DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8
builder2 = DataSource=LOCAL;UserID=SYSTEM;Password=MANAGER8
builder3 = DataSource=SERV1;UserID=SYSTEM;Password=MANAGER8
builder1.EquivalentTo(builder2) = True
builder2.EquivalentTo(builder3) = False
```

Remove

Метод удаляет пару «ключ/значение» из строки подключения экземпляра класса `DbConnectionStringBuilder`.

Поскольку метод Remove возвращает значение, которое свидетельствует об успешном его выполнении, не требуется искать ключ перед попыткой удалить пару «ключ/значение» из экземпляра DbConnectionStringBuilder.

Метод Remove не удаляет физически из строки подключения пару «ключ/значение», а присваивает ключу значение по умолчанию.

Присваиваемые значения по умолчанию приведены в таблице 38.

Таблица 38. Значения по умолчанию ключей строки подключения

Ключ	Значение по умолчанию
DataSource	Пустая строка
User ID	Пустая строка
Password	Пустая строка
Persist Security Info	False
IsolationLevel	IsolationLevel.Unspecified
Autocommit	True
Minimum Pool Size	0
Maximum Pool Size	100
ConnectionTimeout	15
Messages Language	en-US
Charset	Encoding.Default.WebName
Integrated Security	False

Синтаксис

```
public override bool Remove(string keyword);
```

keyword – имя ключа удаляемой пары «ключ/значение».

Возвращаемое значение

Значение true, если ключ существовал в строке подключения и был удален, false – в противном случае.

Исключения

ArgumentNullException	Null-значение параметра keyword.

ShouldSerialize

Метод проверяет наличие указанного ключа в строке подключения.

ShouldSerialize и Reset – необязательные методы, которые могут использоваться свойством, если оно не имеет простого значения по умолчанию. Если свойство имеет простое значение по умолчанию, следует воспользоваться атрибутом DefaultValueAttribute и применить значение по умолчанию для конструктора класса атрибутов.

Примечание
Поведение этого метода идентично поведению метода ContainsKey.

Синтаксис

```
public override bool ShouldSerialize(string keyword);
```

`keyword` – имя проверяемого ключа.

Возвращаемое значение

Значение true, если объект DbConnectionStringBuilder содержит пару «ключ/значение» с указанным ключом, false – в противном случае.

Исключения

`ArgumentNullException`	Null-значение параметра keyword.

TryGetValue

Метод предоставляет значение указанного ключа строки подключения.

Метод `TryGetValue` позволяет безопасно извлекать значение ключа из строки подключения, не требуя предварительной проверки его существования в строке.

При запросе значения несуществующего ключа исключение не генерируется, для него возвращается null-значение.

Синтаксис

```
public override bool TryGetValue(
  string keyword,
  out Object value
);
```

`keyword` – имя ключа.

`value` – возвращаемое текущее значение ключа в строке подключения.

Возвращаемое значение

Значение true, если ключевое слово keyword найдено в строке подключения, false – в противном случае.

Исключения

`ArgumentNullException`	Null-значение параметра keyword.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class ConnectionStringBuilderSample
{
  static void Main()
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
 {
   // Создание фабрики классов провайдера
   DbProviderFactory factory =
     DbProviderFactories.GetFactory("System.Data.LinterClient");
   // Создание объекта DbConnectionStringBuilder
   DbConnectionStringBuilder builder =
     factory.CreateConnectionStringBuilder();
   builder.ConnectionString =
     "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
   // Вызов метода TryGetValue для нескольких имен ключей.
   // Имена ключей конвертируются в синонимы.
   DisplayValue(builder, "Data Source");
   DisplayValue(builder, "UID");
   DisplayValue(builder, "InvalidKey");
   DisplayValue(builder, null);
   Console.WriteLine();
   Console.WriteLine("Нажмите клавишу Ввод для завершения");
   Console.ReadLine();
 }
 private static void DisplayValue(DbConnectionStringBuilder
builder, string key)
 {
   object value = null;
   // Хотя метод TryGetValue нормально обрабатывает отсутствующие
ключи,
   // он генерирует ошибку, если имя ключа равно null. Данный
пример
   // перехватывает эту ошибку, но передает наверх любое другое
исключение.
   try
   {
     if (builder.TryGetValue(key, out value))
     {
       Console.WriteLine("{0}='{1}'", key, value);
     }
     else
     {
       Console.WriteLine("Нельзя получить значение для ключа
'{0}'", key);
     }
   }
   catch (ArgumentNullException)
   {
     Console.WriteLine("Нельзя получить значение для ключа
null");
   }
```

```
   }
}

Результат выполнения примера:

Data Source='LOCAL'
UID='SYSTEM'
Нельзя получить значение для ключа 'InvalidKey'
Нельзя получить значение для ключа null
```

Класс LinterBlob

Класс `LinterBlob` позволяет добавлять порции данных к BLOB-значению.

Методы класса `LinterBlob` приведены в таблице 39.

Таблица 39. Методы класса LinterBlob

Метод	Описание
Append	Добавляет последовательность байт к BLOB-значению.
Clear	Очищает BLOB-значение.

Конструкторы

Отсутствуют. Экземпляр данного класса может быть создан только методом `GetLinterBlobForUpdate`.

Свойства

Отсутствуют.

Методы

Append

Добавляет последовательность байт к BLOB-значению. Данные будут добавляться в BLOB-поле в текущей записи выборки данных. Номер BLOB-поля (столбца) должен быть предварительно установлен с помощью метода `GetLinterBlobForUpdate`.

Синтаксис

```
public void Append(byte[] buffer, int offset, int count);
```

`buffer` – массив байт, которые нужно добавить к BLOB-значению.

`offset` – смещение в буфере массива байт, начиная с которого следует добавлять байты к BLOB-значению (отсчет начинается с `0`).

`count` – количество добавляемых байт.

Возвращаемое значение

Значение типа void.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Исключения

`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

См. пример для метода GetLinterBlobForUpdate.

Clear

Очищает BLOB-значение. Данные будут очищены в BLOB-поле в текущей записи выборки данных. Номер BLOB-поля (столбца) должен быть предварительно установлен с помощью метода `GetLinterBlobForUpdate`.

Синтаксис

```
public void Clear();
```

Возвращаемое значение

Значение типа void.

Исключения

`LinterSqlException`	Код завершения СУБД ЛИНТЕР не равен `0`.

Пример

См. пример для метода GetLinterBlobForUpdate.

Класс LinterSqlException

Класс предназначен для обработки кодов завершения, возвращаемых ADO.NET-провайдеру СУБД ЛИНТЕР.

Этот класс создается всякий раз при обнаружении ADO.NET-провайдером СУБД ЛИНТЕР кода завершения, полученного от ЛИНТЕР-сервера. (Ошибки на стороне клиентского приложения возникают как стандартные исключения среды CLR).

Полный список сообщений LinterSqlException и рекомендации по устранению ошибок см. в документе «Справочник кодов завершения».

Пример

```
// C#
using System;
using System.Data.LinterClient;

class Program
{
  public static void ShowLinterException(string connectionString)
  {
    string queryString = "EXECUTE NonExistantStoredProcedure";

    using (LinterDbConnection connection = new
 LinterDbConnection(connectionString))
```

```
    {
      LinterDbCommand command = new LinterDbCommand(queryString,
 connection);
      try
      {
        command.Connection.Open();
        command.ExecuteNonQuery();
      }
      catch (LinterSqlException ex)
      {
        Console.WriteLine("Код СУБД ЛИНТЕР: " + ex.Number);
        Console.WriteLine("Код операционной системы: " +
 ex.LinterSysErrorCode);
     Console.WriteLine("Номер строки SQL: " + ex.SqlLineNumber);
        Console.WriteLine("Позиция в строке SQL: " +
 ex.SqlPositionInLine);
     Console.WriteLine("Текст сообщения: " + ex.Message);
        Console.WriteLine("Имя поставщика: " + ex.Source);
        Console.WriteLine("Метод: " + ex.TargetSite);
        Console.WriteLine("Стек вызовов: ");
        Console.WriteLine(ex.StackTrace);
      }
    }
  }
  static void Main(string[] args)
  {
    ShowLinterException("Data Source=LOCAL;User
 ID=SYSTEM;Password=MANAGER8");
  }
}
```

Результат выполнения примера:

```
Код СУБД ЛИНТЕР: 2229
Код операционной системы: 2228225
Номер строки SQL: 1
Позиция в строке SQL: 34
Текст сообщения: [Linter error] unknown procedure
Имя поставщика: .Net LinterClient Data Provider
Метод: IntPtr prepare(System.Data.LinterClient.TCBL ByRef,
 System.String)
Стек вызовов:
   at System.Data.LinterClient.LinterBaseFunctions.prepare(TCBL&
 cbl, String strQuery)
   at System.Data.LinterClient.CommandInfo.prepare(TCBL cbl,
 LinterDbParameter[]& lpars)
```

```
at System.Data.LinterClient.MultipleCommands.prepare()
at System.Data.LinterClient.LinterDbCommand.Prepare()
at System.Data.LinterClient.LinterDbCommand.ExecuteNonQuery()
at Program.ShowLinterException(String connectionString)
```

Свойства класса `LinterSqlException` приведены в таблице 40.

Таблица 40. Свойства класса LinterSqlException

Свойство	Описание
Data	Предоставляет массив пар «ключ/значение» с дополнительной информацией об исключении.
ErrorCode	Предоставляет HRESULT кода завершения (32-битное значение (ULONG) используемое для описания кодов завершения).
Errors	Предоставляет один или несколько объектов класса `LinterSqlError`, которые содержат детальные сведения об исключениях, создаваемых ADO.NET-провайдером. **Примечание** В текущей версии ADO.NET провайдера экземпляры класса `LinterSqlError` не создаются.
HelpLink	Предоставляет/устанавливает ссылку на файл справки, связанный с заданным исключением.
InnerException	Предоставляет экземпляр класса `Exception`, вызвавший текущее исключение.
Message	Предоставляет текст сообщения, которое описывает текущее исключение.
Number	Предоставляет числовое значение кода завершения СУБД ЛИНТЕР.
Source	Предоставляет имя поставщика данных, сгенерировавшего код завершения.
SqlLineNumber	Предоставляет номер строки в SQL запросе для ошибок транслятора SQL.
SqlPositionInLine	Предоставляет позицию в строке SQL запроса для ошибок транслятора SQL
StackTrace	Предоставляет строковое представление фрагмента стека вызова в момент возникновения текущего исключения.
TargetSite	Предоставляет имя метода, в котором возникло текущее исключение.

Методы класса `LinterSqlException` приведены в таблице 41.

Таблица 41. Методы класса LinterSqlException

Метод	Описание
GetBaseException	Предоставляет корневое исключение.
GetObjectData	Управляет сериализацией исключения.

Синтаксис

```
[Serializable]
```

```
public class LinterSqlException : DbException
```

Конструкторы

Отсутствуют.

Экземпляр данного класса может быть создан только внутри ADO.NET-провайдера. Клиентское приложение может перехватить и обработать исключение в блоке catch.

Свойства

Data

Предоставляет массив пар «ключ/значение» с дополнительной информацией об исключении.

Объект System.Collections.IDictionary, возвращаемый свойством Data, используется для хранения и извлечения вспомогательных сведений, касающихся исключения. Эти сведения предоставляются клиентскому приложению при выполнении метода `Exception.Data.Add`(<имя ключа>, <значение ключа>), т.е. в форме произвольного количества заданных пользователем пар «ключ/значение». В каждой паре «ключ/значение» ключом обычно является идентифицирующая строка, а значением – объект любого типа.

Характеристики свойства:

- безопасность пар «ключ/значение».

  Пары «ключ/значение», хранящиеся в коллекции, возвращаемой свойством Data, не защищены. Если приложение вызывает вложенную последовательность подпрограмм и в каждой подпрограмме содержится обработчик исключений, то в результирующем стеке вызова содержится иерархия таких обработчиков исключений. Если подпрограмма нижнего уровня порождает (генерирует) исключение, то все обработчики исключений верхнего уровня в иерархии стека вызова могут считывать и/или изменять пары «ключ/значение», сохраненные в коллекции любым другим обработчиком исключений. Поэтому необходимо обеспечить отсутствие в парах «ключ/значение» конфиденциальной информации, а также надлежащую работу приложения в случае, если данные в этих парах будут повреждены;
- конфликты ключей.

  Конфликт ключей происходит, если в разных обработчиках исключений задан один и тот же ключ для доступа к паре «ключ-значение». При разработке приложения следует соблюдать осторожность, поскольку вследствие конфликта ключей обработчики исключений нижнего уровня могут по ошибке обмениваться данными с обработчиками исключений верхнего уровня, а такой обмен может послужить причиной неочевидных ошибок программы. Однако при осторожном подходе конфликты ключей можно использовать для расширения возможностей приложения;
- предупреждение конфликтов ключей.

  Для предупреждения конфликтов ключей можно принять контекст именования, чтобы создавать уникальные ключи для пар «ключ/значение». Например, контекст именования может обеспечить создание ключа, состоящего из имени приложения, метода, предоставляющего вспомогательные сведения для пары «ключ/значение», и уникального идентификатора, разделенных между собой точкой.

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Предположим, есть два приложения с именами Products (Продукты) и Suppliers (Поставщики), у каждого из которых есть метод `Sales` (Продажи). Метод `Sales` в приложении Products предоставляет идентификационный номер продукта (SKU). Метод `Sales` в приложении Suppliers предоставляет идентификационный номер поставщика (SID). Следовательно, контекст именования в этих случаях будет выглядеть как Products.Sales.SKU и Suppliers.Sales.SID;

- использования конфликтов ключей.

  Конфликтом ключей можно воспользоваться, применив один или несколько заданных ключей для управления обработкой исключений. Предположим, что в одном сценарии обработчик исключений самого высокого уровня в иерархии стека вызова перехватывает все исключения, генерируемые обработчиками исключений нижнего уровня. Если пара «ключ/значение» с особым ключом существует, то обработчик исключений высшего уровня форматирует остальные пары «ключ/значение» в объекте IDictionary каким-либо нестандартным образом; в противном случае оставшиеся пары «ключ/значение» форматируются неким обычным способом.

Теперь предположим, что в другом сценарии обработчики исключений на каждом уровне в иерархии стека вызова перехватывают исключения, генерируемые обработчиками исключений предыдущего, более низкого уровня. Кроме того, каждому обработчику исключений известно, что в коллекции, возвращенной свойством Data, содержится набор пар «ключ/значение», доступ к которым можно получить с помощью заданного набора ключей.

Каждый обработчик исключений использует заданный набор ключей для обновления значений в соответствующей паре «ключ/значение» уникальными для этого обработчика исключений сведениями. По завершении процесса обновления обработчик событий генерирует исключение следующему обработчику исключений верхнего уровня. Наконец обработчик исключений наивысшего уровня получает доступ к паре «ключ/значение» и отображает сводные сведения для обновления, полученные от всех обработчиков исключений более низкого уровня.

Декларация

```
public virtual IDictionary Data {get;};
```

Значение свойства

Объект, который реализует интерфейс System.Collections.IDictionary и содержит в себе коллекцию пар «ключ/значение». По умолчанию коллекция является пустой.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Collections;

class Sample
{
  public static void Main()
  {
```

```
    Console.WriteLine();
    Console.WriteLine("Исключение с частичной дополнительной
информацией");
    RunTest(false);

    Console.WriteLine();
    Console.WriteLine("Исключение с полной дополнительной
информацией");
    RunTest(true);
  }
  public static void RunTest(bool displayDetails)
  {
    try
    {
      NestedRoutine1(displayDetails);
    }
    catch (Exception e)
    {
      Console.WriteLine("Исключение было сгенерировано.");
      Console.WriteLine(e.Message);
      if (e.Data != null)
      {
        Console.WriteLine("  Дополнительная информация об
исключении:");
        foreach (DictionaryEntry de in e.Data)
        {
          Console.WriteLine("    Ключ '{0}' и его значение : {1}",
            de.Key, de.Value);
        }
      }
    }
  }
  public static void NestedRoutine1(bool displayDetails)
  {
    try
    {
      NestedRoutine2(displayDetails);
    }
    catch (Exception e)
    {
      e.Data["ExtraInfo"] =
        "Частичная информация от вспомогательной подпрограммы1.";
      e.Data.Add("MoreExtraInfo",
        "Полная информация от вспомогательной подпрограммы1.");
      throw e;
    }
```

```
  }
  public static void NestedRoutine2(bool displayDetails)
  {
    Exception e = new Exception("Оригинальное сообщение об
 исключении.");
    if (displayDetails)
    {
      string s = "Информация от вспомогательной подпрограммы2.";
      int i = -903;
      DateTime dt = DateTime.Now;
      e.Data.Add("stringInfo", s);
      e.Data["IntInfo"] = i;
      e.Data["DateTimeInfo"] = dt;
    }
    throw e;
  }
}

Результат выполнения примера:

Исключение с частичной дополнительной информацией
Исключение было сгенерировано.
Оригинальное сообщение об исключении.
Дополнительная информация об исключении:
    Ключ 'ExtraInfo' и его значение : Частичная информация от
 вспомогательной подпрограммы1.
    Ключ 'MoreExtraInfo' и его значение : Полная информация от
 вспомогательной подпрограммы1.

Исключение с полной дополнительной информацией
Исключение было сгенерировано.
Оригинальное сообщение об исключении.
  Дополнительная информация об исключении:
    Ключ 'stringInfo' и его значение : Информация от
 вспомогательной подпрограммы2.
    Ключ 'IntInfo' и его значение : -903
    Ключ 'DateTimeInfo' и его значение : 24.04.2013 14:44:35
    Ключ 'ExtraInfo' и его значение : Частичная информация от
 вспомогательной подпрограммы1.
    Ключ 'MoreExtraInfo' и его значение : Полная информация от
 вспомогательной подпрограммы1.
```

ErrorCode

Предоставляет HRESULT кода завершения (32-битное значение (ULONG) используемое для описания кодов завершения).

Структура значения HRESULT приведена на рисунке 5, описание полей в таблице 42.

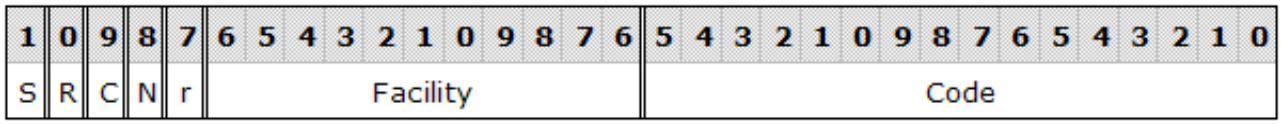


Рисунок 5. Структура значения HRESULT

Таблица 42. Описание полей структуры HRESULT

Группа битов	Назначение	Количество битов	Описание
S	Признак серьезности ошибки	1	0 – нормальное завершение 1 – ошибка (`SEVERITY_ERROR`)
R	Зарезервирован	1	Игнорируется
C	Зарезервирован	1	Игнорируется
N	Зарезервирован	1	Игнорируется
R	Зарезервирован	1	Игнорируется
Facility (источник)	Источник определения кода завершения	11	Содержит идентификатор программного компонента, в котором определен данный код завершения, например, `FACILITY_RPC`
Code (код)	Значение кода завершения	16	Числовое значение кода завершения, например, `E_UNEXPECTED`

Декларация

```
public virtual int ErrorCode {get;};
```

Значение свойства

Значение HRESULT кода завершения.

В текущей версии ADO.NET-провайдер СУБД ЛИНТЕР не устанавливает значение данного свойства, поэтому оно всегда имеет значение E_FAIL (0x80004005).

Исключения

Отсутствуют.

Errors

Предоставляет один или несколько объектов класса `LinterSqlError`, которые содержат детальные сведения об исключениях, создаваемых ADO.NET-провайдером СУБД ЛИНТЕР.

Примечание
В текущей версии ADO.NET провайдера экземпляры класса LinterSqlError не создаются.

Декларация

```
public LinterSqlErrorCollection Errors {get;};
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Значение свойства

Экземпляры объектов класса `LinterSqlError`.

> **Примечание**
> В текущей версии ADO.NET-провайдер СУБД ЛИНТЕР не заполняет коллекцию Errors, поэтому данное свойство всегда равно пустой коллекции LinterSqlErrorCollection.

Исключения

Отсутствуют.

HelpLink

Предоставляет или устанавливает ссылку на файл справки, связанный с заданным исключением.

Декларация

```
public virtual string HelpLink {get; set;};
```

Значение свойства

URN или URL-адрес файла справки.

Например, `file:///C:/Applications/Sale/help.html#ErrorNum42`.

> **Примечание**
> В текущей версии ADO.NET-провайдер СУБД ЛИНТЕР не устанавливает значение данного свойства, поэтому оно всегда имеет null-значение.

Исключения

Отсутствуют.

InnerException

Предоставляет экземпляр класса `Exception`, вызвавший текущее исключение.

Свойство InnerException возвращает то же значение, что было передано в конструктор, или null-указатель, если конструктору не было передано значение внутреннего исключения.

При создании исключения `X`, являющегося непосредственным результатом предыдущего исключения `Y`, свойство InnerException параметра `X` должно содержать ссылку на параметр `Y`.

Свойство InnerException используется, чтобы получить список предшествующих исключений, ставших причиной текущего исключения.

Можно создать новое исключение, которое перехватывает более раннее исключение. Код, обрабатывающий второе исключение, для более адекватной обработки ошибки может использовать дополнительные сведения из более раннего исключения.

Допустим, что существует функция `ReadFile`, которая считывает файл и форматирует данные на основе этого файла. Если при считывании файла функцией `ReadFile`

возникает исключение IOException, то функция `ReadFile` перехватывает это исключение IOException и генерирует новое исключение FileNotFoundException. Исключение IOException можно сохранить в свойстве InnerException исключения FileNotFoundException, разрешив выполнение кода, перехватывающего исключение FileNotFoundException, для выяснения причины первоначальной ошибки.

Свойство InnerException, содержащее ссылку на внутреннее исключение, устанавливается при инициализации объекта исключения.

Декларация

```
public Exception InnerException {get;};
```

Значение свойства

Экземпляр Exception, описывающий ошибку, вызвавшую текущее исключение.

Примечание
В текущей версии ADO.NET-провайдер СУБД ЛИНТЕР не устанавливает значение данного свойства, поэтому оно всегда имеет null-значение.

Исключения

Отсутствуют.

Message

Предоставляет текст сообщения, которое описывает текущее исключение.

Текст сообщения Message должен содержать полное описание кода завершения (ошибки) и, по возможности, объяснение способа устранения ошибки. Значение свойства Message включается в сведения, возвращаемые методом `ToString`.

Свойство Message устанавливается только при создании исключения Exception, значение по умолчанию отсутствует, потому что в ADO.NET-провайдере СУБД ЛИНТЕР нет конструктора без параметров для класса `LinterSqlException`.

Примечание
Полный список сообщений LinterSqlException и рекомендации по устранению ошибок см. в документе «Справочник кодов завершения».

Язык сообщения Message может быть английский или русский, в зависимости от языка текущего потока:

```
System.Threading.Thread.CurrentThread.CurrentUICulture.
```

Английские текстовые сообщения находятся в сборке `System.Data.LinterClient.dll`.

Русские текстовые сообщения находятся в сборке `System.Data.LinterClient.resources.dll`, которая расположена в подкаталоге ru-RU пакета провайдера СУБД ЛИНТЕР.

Декларация

```
public virtual string Message {get;};
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Значение свойства

Текстовая расшифровка кода завершения с объяснением причин исключения или пустая строка (""), если в ADO.NET провайдере отсутствует расшифровка кода завершения.

Исключения

Отсутствуют.

Пример

```
// В данном примере создается метод Execute, в котором
 генерируется исключение
// LinterSqlException при попытке выполнить неизвестную хранимую
 процедуру. Данный
// метод вызывается два раза из основного метода Main. Перед
 каждым вызовом метода
// Execute в методе Main устанавливается язык текущего потока:
 первый раз
// устанавливается язык "en-US", второй раз – "ru-RU".
// C#
using System;
using System.Data.LinterClient;
using System.Threading;
using System.Globalization;

class Program
{
  static void Main()
  {
    Console.WriteLine("Язык en-US");
    // Установка языка текущего потока "en-US"
    Thread.CurrentThread.CurrentUICulture = new CultureInfo("en-
US");
    // Выполнение метода, в котором генерируется исключение
    Execute();

    Console.WriteLine("Язык ru-RU");
    // Установка языка текущего потока "ru-RU"
    Thread.CurrentThread.CurrentUICulture = new CultureInfo("ru-
RU");
    // Выполнение метода, в котором генерируется исключение
    Execute();
  }

  private static void Execute()
  {
    LinterDbConnection con = null;
```

```
    try
    {
      // Создание объекта соединение
      con = new
 LinterDbConnection("UserID=SYSTEM;Password=MANAGER8");

      // Создание объекта команда
      LinterDbCommand cmd = new LinterDbCommand("execute
 NOT_EXIST", con);

      // Открытие соединения
      con.Open();

      // Выполнение команды, которая генерирует исключение
 LinterSqlException
      cmd.ExecuteNonQuery();
    }
    catch (LinterSqlException ex)
    {
      // Обработка исключений СУБД ЛИНТЕР
      Console.WriteLine(
        "Текст сообщения: " + ex.Message + "\n" +
        "Код СУБД ЛИНТЕР: " + ex.Number + "\n");
    }
    catch (Exception ex)
    {
      // Обработка исключений других типов
      Console.WriteLine(
        "Текст сообщения: " + ex.Message + "\n" +
        "Тип исключения: " + ex.GetType() + "\n");
    }
    finally
    {
      // Освобождение ресурсов
      if (con != null)
      {
        con.Close();
      }
    }
  }
}
Результат выполнения примера
Язык en-US
Текст сообщения: [Linter Code] unknown procedure
Код СУБД ЛИНТЕР: 2229
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
Язык ru-RU
Текст сообщения: [Код СУБД ЛИНТЕР] неизвестная процедура
Код СУБД ЛИНТЕР: 2229
```

Number

Предоставляет числовое значение кода завершения СУБД ЛИНТЕР (значение поля NMRERR в записи системной таблицы ERRORS в БД ЛИНТЕР).

Коды завершения СУБД ЛИНТЕР см. в документе «Справочник кодов завершения».

Декларация

```
public int Number {get;};
```

Значение свойства

Числовое значение кода завершения СУБД ЛИНТЕР.

Исключения

Отсутствуют.

Пример

См. пример в разделе Класс LinterSqlException.

Source

Предоставляет имя поставщика данных, сгенерировавшего код завершения.

Декларация

```
public string Source {get;};
```

Значение свойства

Имя поставщика, сгенерировавшего код завершения.

ADO.NET-провайдер СУБД ЛИНТЕР возвращает ".Net LinterClient Data Provider".

Исключения

Отсутствуют.

Пример

См. пример в разделе Класс LinterSqlException.

SqlLineNumber

Предоставляет номер строки в SQL запросе для ошибок транслятора SQL (диапазон кодов завершения от 2000 до 2999).

Декларация

```
public ushort SqlLineNumber {get;}
```

Значение свойства

Номер строки, содержащей ошибку (отсчет начинается с `1`).

Исключения

Отсутствуют.

Пример

См. пример в разделе Класс LinterSqlException.

SqlPositionInLine

Предоставляет позицию в строке SQL запроса для ошибок транслятора SQL (диапазон кодов завершения от 2000 до 2999).

Декларация

```
public ushort SqlPositionInLine {get;}
```

Значение свойства

Позиция в строке, где обнаружена ошибка (отсчет начинается с `1`).

Исключения

Отсутствуют.

Пример

См. пример в разделе Класс LinterSqlException.

StackTrace

Предоставляет строковое представление фрагмента стека вызова в момент возникновения текущего исключения.

В стеке выполнения отслеживаются все методы, выполняемые в данном экземпляре клиентского приложения. Трассировка вызовов метода называется трассировкой стека. Список трассировок стека позволяет проследить последовательность вызовов до номера строки метода, в котором происходит исключение.

Из-за таких преобразований, как «встраивание», происходящих с кодом во время оптимизации, свойство StackTrace может сообщить о меньшем количестве вызовов, чем ожидается.

Декларация

```
public virtual string StackTrace {get;};
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Значение свойства

Строка, описывающая содержимое стека вызова, в которой первым отображается самый последний вызванный метод (т.е. тот, который сгенерировал исключение при помощи оператора throw).

Исключения

Отсутствуют.

Пример

См. пример в разделе Класс LinterSqlException.

TargetSite

Предоставляет имя метода, в котором возникло текущее исключение.

Если метод, генерирующий исключение, недоступен и трассировка стека не является null-указателем, то свойство TargetSite извлекает метод из трассировки стека. Если трассировка стека является null-указателем, то свойство TargetSite также возвращает null-указатель.

Декларация

```
public MethodBase TargetSite {get;};
```

Значение свойства

Метод `MethodBase`, в котором возникло текущее исключение.

Исключения

Отсутствуют.

Пример

См. пример в разделе Класс LinterSqlException.

Методы

GetBaseException

Метод предоставляет исключение Exception, которое является корневой причиной одного или нескольких исключений.

Цепочка исключений состоит из набора исключений, поэтому каждое исключение в цепочке генерируется как непосредственный результат исключения, на которое ссылается свойство InnerException. Для данной цепочки может существовать только одно исключение, являющееся корневой причиной всех других исключений в этой цепочке. Это исключение называется базовым и его InnerException свойство всегда содержит null-значение.

Для всех исключений в цепочке исключений метод `GetBaseException` возвращает один и тот же объект – базовое исключение.

Метод GetBaseException используется при необходимости найти корневую причину исключения.

Синтаксис

```
public virtual Exception GetBaseException();
```

Возвращаемое значение

Корневое исключение из цепочки исключений.

Если свойство InnerException текущего исключения возвращает null-значение, то метод предоставляет текущее исключение.

Примечание
Так как в текущей версии ADO.NET-провайдера СУБД ЛИНТЕР свойство InnerException текущего исключения всегда возвращает null-значение, то данный метод предоставляет текущее исключение.

Исключения

Отсутствуют.

GetObjectData

Метод управляет сериализацией исключения.

Сериализация: процесс перевода объекта в поток битов с целью сохранения его в памяти (или передаче по каналу связи) с возможностью его воссоздания при необходимости.

Десериализация: операция восстановления состояния структур данных из битовой последовательности.

Синтаксис

```
[SecurityPermissionAttribute(SecurityAction.Demand,
 SerializationFormatter = true)]
public override void GetObjectData(
  SerializationInfo info,
  StreamingContext context
);
```

info – используемый объект SerializationInfo, содержащий имя, тип и значение каждой составной части сериализуемого объекта. Во время десериализации эти сведения извлекаются с помощью соответствующей функции.

Например, если сериализуется структура данных из 5 полей, то объект SerializationInfo описывает характеристики каждого поля (имя, тип данных и значение поля).

context – используемый объект StreamingContext. Описывает источник и назначение данного сериализованного потока и предоставляет дополнительный, определяемый

вызывающим, контекст, т.е. описывает функцию (метод), которая должна вызываться для сериализации каждой составной части сериализуемого объекта.

Например, для сериализации значения типа integer – одна функция, для double – другая и т.п.



Примечание
Исключение LinterSqlException нельзя сериализовать.

Возвращаемое значение

Значение типа void.

Исключения

`SerializationException`	Попытка сериализовать исключение LinterSqlException (это исключение сериализовать нельзя).

Общие свойства и методы классов ADO.NET-провайдера

Общие свойства и методы классов ADO.NET-провайдера СУБД ЛИНТЕР представлены в таблице 43.

Таблица 43. Общие методы и свойства классов ADO.NET-провайдера

Класс	Свойство		Метод						
	Container	Site	Dispose()	Equals	GetHashCode	GetLifetimeService	GetType	InitializeLifetimeService	ToString
DbCommand	+	+	+	+	+	+	+	+	+
DbCommandBuilder	+	+	+	+	+	+	+	+	+
DbConnection	+	+	+	+	+	+	+	+	+
DbConnectionStringBuilder				+	+		+		+
DbDataAdapter	+	+	+	+	+	+	+	+	+
DbDataReader			+	+	+	+	+	+	+
DbParameter				+	+	+	+	+	+
DbParameterCollection				+	+	+	+	+	+
DbTransaction			+	+	+	+	+	+	+
LinterClientFactory				+	+		+		+

Общие свойства

Общие свойства классов ADO.NET-провайдера приведены в таблице 44.

Таблица 44. Общие свойства классов ADO.NET-провайдера

Свойство	Описание
Container	Предоставляет контейнер IContainer, содержащий компонент Component.
Site	Предоставляет/устанавливает экземпляр ISite для компонента Component.

Container

Предоставляет контейнер IContainer, содержащий компонент Component.

Компоненты в контейнере организованы в форме списка FIFO («первым пришел, первым ушел»), который также определяет порядок компонентов в контейнере. Последний добавленный компонент является последним в списке.

© Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Декларация

```
[BrowsableAttribute(false)]
public IContainer Container {get;};
```

Значение свойства

Контейнер IContainer, содержащий компонент Component, если таковой присутствует, или null-значение, если компонент Component не инкапсулирован в контейнер IContainer.

Исключения

Отсутствуют.

Site

Предоставляет или задает экземпляр ISite для компонента Component.

Компоненту Component будет назначен узел ISite, если этот компонент был добавлен в контейнер IContainer. Контейнер IContainer отвечает за назначение узла ISite компоненту Component. Изменение значения узла ISite, связанного с компонентом, не обязательно приводит к изменению имени узла, которому назначен компонент Component. Свойство Site должно устанавливаться только контейнером IContainer.

Установка для свойства null-значения необязательно удаляет компонент Component из контейнера IContainer.

У компонента Component имя может отсутствовать. Если у компонента Component имя есть, оно не должно совпадать с именем другого объекта Component, содержащегося в том же контейнере IContainer. Имя объекта Component хранится в узле ISite, поэтому компонент Component может получить имя, только если с ним связан узел ISite.

Декларация

```
[BrowsableAttribute(false)]
public virtual ISite Site {get; set;};
```

Значение свойства

Узел ISite, связанный с компонентом Component, или null-значение, если:

- компонент Component не инкапсулирован в контейнер IContainer;
- с компонентом Component не связан узел ISite;
- компонент Component удален из своего контейнера IContainer.

Исключения

Отсутствуют.

Общие методы

Общие методы классов ADO.NET-провайдера приведены в таблице 45.

Таблица 45. Общие методы классов ADO.NET-провайдера

Метод	Описание
Dispose	Освобождает все ресурсы, используемые объектом Component.
Equals	Сравнивает текущий и заданный объекты.
GetHashCode	Предоставляет хеш-функцию для определенного типа данных.
GetLifetimeService	Предоставляет объект, который управляет временем существования указанного экземпляра класса.
GetType	Предоставляет тип объекта для текущего экземпляра класса.
InitializeLifetimeService	Предоставляет объект обслуживания аренды для управления политикой времени существования указанного экземпляра класса.
ToString	Предоставляет строковое значение имени указанного компонента.

Dispose

Освобождает все ресурсы, используемые объектом.

Метод `Dispose` должен вызываться по окончании использования объекта. После вызова метода `Dispose` необходимо удалить все ссылки на объект, чтобы сборщик мусора мог освободить память, занимаемую объектом.

Чтобы правильно вызвать метод `Dispose`, следует объявить и создать объект в операторе using. Оператор using соответствующим образом вызывает метод `Dispose` в объекте и приводит к выводу объекта из области действия сразу после вызова `Dispose`. Оператор using гарантирует вызов метода `Dispose`, даже если при вызове методов в объекте происходит исключение. Такого же результата можно достичь при размещении объекта в блоке try и последующем вызове метода `Dispose` в блоке finally (см. приложение 1).

Примечание
Метод необходимо всегда вызывать для освобождения последней ссылки на объект. В противном случае используемые им ресурсы не будут освобождены.

Синтаксис

```
public void Dispose();
```

Возвращаемое значение

Значение типа void.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
using System.Data;
using System.Data.Common;

class DisposeSample
{
  static void Main()
  {
    // Создание объекта DbProviderFactory
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание объекта DbConnection
    using (DbConnection con = factory.CreateConnection())
    {
      con.ConnectionString =
        "User ID=SYSTEM;Password=MANAGER8;Data Source=LOCAL";
      con.Open();
      // Создание объекта DbCommand
      using (DbCommand cmd = factory.CreateCommand())
      {
        cmd.Connection = con;
        cmd.CommandText = "select * from some_table";
        // Создание объекта DbDataReader
        using (DbDataReader reader = cmd.ExecuteReader())
        {
          while (reader.Read())
          {
            for (int i = 0; i < reader.FieldCount; i++)
            {
              Console.Write(reader.GetValue(i) + "\t");
            }
            Console.WriteLine();
          }
        } // Метод DbDataReader.Dispose() был вызван автоматически
      } // Метод DbCommand.Dispose() был вызван автоматически
    } // Метод DbConnection.Dispose() был вызван автоматически
  }
}
```

Equals

Метод сравнивает текущий и заданный объекты. Равенство объектов проверяется путем побитового сравнения типов значений, на которые ссылаются объекты.

Равенство ссылок означает, что сравниваемые ссылки на объект указывают на один и тот же объект.

Побитовое равенство означает, что сравниваемые объекты обладают одинаковым двоичным представлением.

Равенство значений означает, что сравниваемые объекты обладают одинаковыми значениями, несмотря на то, что их двоичные представления отличаются. Например, есть два объекта Decimal, представляющие числа 1.10 и 1.1000. Объекты типа Decimal побитово неравны, поскольку они обладают различными битовыми представлениями, позволяющими учесть различное количество последних нулей в дробной части. Однако значения этих объектов равны, поскольку числа 1.10 и 1.1000 при сравнении считаются равными, поскольку последние нули в дробной части не значимы.

Метод `Equals` предназначен только для сравнения примитивов и объектов (но не для сложных структур, таких, как массивы объектов).

Следующие утверждения справедливы для всех вызовов метода `Equals`. В списке буквы x, y, и z обозначают ссылки на объекты, не равные null-ссылке:

- x.Equals(x) возвращает значение true, кроме случая, когда x относится к типу с плавающей запятой;
- x.Equals(y) возвращает то же значение, что и метод y.Equals(x);
- x.Equals(y) возвращает значение true, если x и y относятся к типу NaN;
- (x.Equals(y) && y.Equals(z)) возвращает значение true, в том и только в том случае, если метод x.Equals(z) возвращает значение true;
- последующие вызовы метода x.Equals(y) возвращают то же значение до тех пор, пока объекты, ссылками на которые являются x и y, остаются неизменными;
- x.Equals(null-ссылка) возвращает значение false.

Синтаксис

```
public virtual bool Equals(Object obj);
```

`obj` – объект который требуется сравнить с текущим объектом.

Возвращаемое значение

Результат сравнения:

- true – сравниваемые объекты равны;
- false – сравниваемые объекты не равны.

Исключения

Отсутствуют.

Пример

```
// C#
using System;

class EqualsSample
{
  static void Main()
  {
    Object obj1 = new Object();
    Object obj2 = new Object();
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    Console.WriteLine(obj1.Equals(obj2)); //===> false
    obj2 = obj1;
    Console.WriteLine(obj1.Equals(obj2)); //===> true
  }
}
```

GetHashCode

Метод предоставляет хеш-функцию для определенного типа данных.

Метод `GetHashCode` можно использовать в алгоритмах хеширования и таких структурах данных, как хеш-таблицы.

Метод `GetHashCode` по умолчанию не гарантирует уникальность возвращаемых для объекта значений. Следовательно, метод по умолчанию не следует использовать для хеширования в качестве уникального идентификатора объекта.

Хеш-функция используется для быстрого создания числа (хеш-кода), соответствующего значению объекта. Обычно каждому объекту Type соответствует своя хеш-функция, у которой в качестве входного аргумента должно использоваться хотя бы одно из полей экземпляра.

Хеш-функция обладает следующими свойствами:

- если два объекта при сравнении оказались равны, методы `GetHashCode` обоих этих объектов возвращают одинаковые значения. Однако если при сравнении оказалось, что эти объекты не равны, методы `GetHashCode` этих объектов не обязательно должны возвращать разные значения;
- метод `GetHashCode` последовательно возвращает для объекта один и тот же хеш-код, пока в состоянии объекта не произойдут изменения, определяющие значения, возвращаемые для объекта методом `Equals`. Это справедливо только для текущего выполнения клиентского приложения, и при повторном запуске приложения может возвращаться другой хеш-код;
- значения хеш-функции подчиняются случайному распределению для всех входных аргументов.

Например, метод `GetHashCode` для класса `String` возвращает уникальные хеш-коды для уникальных строковых значений. Следовательно, два объекта String возвращают тот же хеш-код, если они представляют одно и то же строковое значение. Кроме того, этот метод использует все символы в строке для создания случайно распределенного результата, даже если входной параметр ограничен каким-либо диапазоном (например, многие пользователи применяют строки, содержащие только первые 128 символов набора ASCII, хотя строка может содержать любые из 65535 символов Юникода).

Предоставление хорошей хеш-функции для каждого класса может значительно ускорить добавление соответствующих объектов в хеш-таблицу. Поиск элементов в хеш-таблице при надлежащей реализации хеш-функции занимает постоянное время (например, операция O(1)). Скорость поиска элементов при плохой реализации хеш-функции зависит от числа элементов в хеш-таблице (например, O(n), где n – число элементов в хеш-таблице). Вычисление хеш-функций не отнимает значительного объема ресурсов.

Синтаксис

```
public virtual int GetHashCode();
```

Возвращаемое значение

Хеш-код для текущего объекта Object.

Исключения

Отсутствуют.

Пример

```
// C#
using System;
using System.Data;
using System.Data.Common;

class GetHashCodeSample
{
  static void Main()
  {
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "User ID=SYSTEM;Password=MANAGER8";
    con.Open();
    // Создание объекта DbCommand
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    // Формирование текста SQL-запроса
    cmd.CommandText =
      "select NAME, FIRSTNAM from PERSON limit 2";
    // Выполнение SQL-запроса
    DbDataReader reader = cmd.ExecuteReader();
    // Обработка результатов запроса
    string value_name;
    string value_firstnam;
    int hash_code;
    while (reader.Read())
    {
      value_name = reader.GetString(0);
      value_firstnam = reader.GetString(1);
      hash_code = value_name.GetHashCode() +
 value_firstnam.GetHashCode();
      Console.WriteLine("Фамилия " + value_name +
        " имя " + value_firstnam + " хеш-код " + hash_code);
    }
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    // Освобождение ресурсов
    reader.Dispose();
    cmd.Dispose();
    con.Dispose();
  }
}
```

```
Результат выполнения примера:

Фамилия QUIHLLAULT  имя CHARLES   хеш-код -558848851
Фамилия KIM         имя EDDIE     хеш-код 1246850474
```

GetLifetimeService

Метод предоставляет объект, который управляет временем существования данного экземпляра класса. Этот объект используется при распределенной сборке мусора.

Клиентские приложения могут создавать на сервере различные объекты. Некоторые объекты (типа `SingleCall`) уничтожаются сразу после вызова метода и освобождают использованные ресурсы.

Другие активизированные клиентским приложением объекты имеют длительное время существования и при сборке мусора необходимо знать, они еще активны или их уже можно уничтожить. Часто о текущем состоянии таких объектов узнать нельзя (либо клиентское приложение завершило свою работу и оставило их за ненадобностью, не уведомив об этом, либо с клиентским приложением разорвана связь и оно больше не может управлять своими объектами).

Для таких активизированных клиентским приложением объектов, на которые ссылаются извне домена приложения, создается так называемая «аренда», т.е. задается максимальное допустимое время существования объекта. Когда время аренды достигает нуля, аренда заканчивается, удаленный объект отсоединяется от клиентского приложения, после чего оставшийся от объекта мусор можно будет уже убирать.

Время аренды по умолчанию для объекта составляет 300 секунд.

Если клиентское приложение вызывает метод на объекте, когда аренда истекла, возникает исключение SecurityException.

Если созданный клиентским приложением объект должен быть активным более 300 с, то существует три способа обновления аренды:

1) неявное обновление делается автоматически, когда клиентское приложение вызывает метод на удаленном объекте. Если текущее время аренды меньше, чем значение RenewOnCallTime, то аренда задается как RenewOnCallTime;
2) при явном обновлении клиентское приложение устанавливает новое время аренды. Это делается с помощью метода `Renew`() из интерфейса ILease. Доступ к интерфейсу ILease можно получить, вызывая метод `GetLifetimeService`() на прозрачном прокси-сервере;
3) третьей возможностью обновления аренды является так называемое «спонсорство». Клиентское приложение может создать спонсора, который размещается там же, где и арендуемый объект.

Спонсор – это объект, которому клиентское приложение передало право управлять активностью созданных им на удаленном компьютере объектов после завершения своей работы.

Спонсор реализует интерфейс ISponsor и регистрирует себя в службах аренды с помощью метода `Register`() из интерфейса ILease. Когда аренда заканчивается, то длительность её продления запрашивается у спонсора, а не у клиентского приложения (которое к этому времени может быть недоступно или неактивно). Механизм спонсорства используется, если на сервере требуются долгоживущие удаленные объекты.

Распределенная сборка мусора управляет временем жизни серверных объектов и отвечает за их удаление по истечении времени жизни. Обычно распределенная сборка мусора использует подсчет ссылок на объект и тестовый опрос для решения вопроса о возможности удаления мусора от объекта. Это хорошо срабатывает, если на один объект приходится несколько клиентских приложений, но с их увеличением до тысяч на каждый объект такая распределенная сборка мусора работает все хуже. В такой ситуации служба времени жизни объекта может принимать функцию традиционного распределенного сборщика мусора и хорошо масштабируется при увеличении количества клиентов.

Синтаксис

```
public Object GetLifetimeService();
```

Возвращаемое значение

Объект типа ILease, используемый для управления политикой времени существования данного экземпляра.

Исключения

`SecurityException`	У непосредственно вызывающего объекта отсутствует разрешение инфраструктуры.

Примеры

1) Получение информации об аренде.

Для интерфейса ILease необходимо открыть пространство имен System.Runtime.Remoting.Lifetime:

```
ILease lease = (ILease)obj.GetLifetimeService();
if (lease != null) {
  Console.WriteLine("Lease Configuration:");
  Console.WriteLine("InitialLeaseTime: " +
 lease.InitialLeaseTime);
  Console.WriteLine("RenewOnCallTime: " + lease.RenewOnCallTime);
  Console.WriteLine("SponsorshipTimeout: " +
 lease.SponsorshipTimeout);
  Console.WriteLine(lease.CurrentLeaseTime);
}
```

```
Результат выполнения примера:
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
Lease Configuration:
InitialLeaseTime: 00:05:00
RenewOnCallTime: 00.02.00
SponsorshipTimeout: 00.02.00
00.04.59.4191648
```

2) Предположим, что клиентское приложение создало пользовательский класс, реализующий ISponsor и вызывающий метод `Renewal`() для возврата конкретной величины времени (через тип TimeSpan). Ассоциировать указанный тип с созданным удаленным объектом можно либо с доменом приложения сервера, либо с доменом приложения клиента.

Для этого заинтересованная сторона должна получить ссылку ILease (с помощью наследуемого метода `GetLifetimeService`() на стороне сервера или статического метода `RemotingServices.GetLifetimeService`() на стороне клиента) и вызвать `Register`().

```
// Регистрация спонсора на стороне сервера
CarSponsor mySponsor = new CarSponsor();
ILease itfLeaseInfo = (ILease)this.GetLifetimeService();
itfLeaseInfo.Register(mySponsor);
// Регистрация спонсора на стороне клиента
CarSponsor mySponsor = new CarSponsor();
CarProvider cp = new CarProvider(cars);
ILease itfLeaseInfo =
 (ILease)Remoting.Services.GetLifetimeService(cp);
itfLeaseInfo.Register.(mySponsor);

В любом случае, если клиент или сервер желают отменить
 спонсорство,
это можно сделать с помощью метода ILease.Unregister(), например:

// Отключение спонсора для данного объекта
itfLeaseInfo.Unregister(mySponsor);
```

GetType

Метод предоставляет тип объекта для текущего экземпляра класса.

Синтаксис

```
public Type GetType();
```

Возвращаемое значение

Экземпляр `Type`, представляющий точный тип среды выполнения для текущего экземпляра класса.

Исключения

Отсутствуют.

Примеры

1)

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Data.LinterClient;

class GetTypeSample
{
  static void Main()
  {
    // Регистрация фабрики классов провайдера

 DbProviderFactories.RegisterFactory("System.Data.LinterClient",
 typeof(LinterClientFactory));
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание объекта DbCommandBuilder
    DbCommandBuilder builder = factory.CreateCommandBuilder();
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    // Создание объекта RemoteServer
    RemoteServer server = new RemoteServer();
    // Создание объекта string
    string str = "СУБД ЛИНТЕР";
    // Создание объекта int
    int i = 123;
    // Создание объекта byte[]
    byte[] array = new byte[] { 1, 2, 3 };
    // Отображение типов данных
    Console.WriteLine("builder: " + builder.GetType());
    Console.WriteLine("adapter: " + adapter.GetType());
    Console.WriteLine("server:  " + server.GetType());
    Console.WriteLine("str:     " + str.GetType());
    Console.WriteLine("i:       " + i.GetType());
    Console.WriteLine("array:   " + array.GetType());
  }
}
struct RemoteServer
{
  int AccessCode;
  string ConnectionString;
}
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Результат выполнения примера:

```
builder: System.Data.LinterClient.LinterDbCommandBuilder
adapter: System.Data.LinterClient.LinterDbDataAdapter
server:  RemoteServer
str:     System.String
i:       System.Int32
array:   System.Byte[]
```

2)

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Data.LinterClient;
using System.Text;

class GetTypeSample
{
  static void Main()
  {
    // Регистрация провайдера кодировок
    Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
    // Регистрация фабрики классов провайдера

 DbProviderFactories.RegisterFactory("System.Data.LinterClient",
 typeof(LinterClientFactory));
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Соединение с БД
    DbConnection con = factory.CreateConnection();
    con.ConnectionString =
      "User ID=SYSTEM;Password=MANAGER8";
    // Создание таблицы БД
    DbCommand cmd = factory.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText =
      "create or replace table USERS ( " +
      "ID integer primary key, NAME varchar(70));" +
      "insert into USERS (ID, NAME) values (0, 'Пользователь A');"
 +
      "insert into USERS (ID, NAME) values (1, 'Пользователь
 B');";
    con.Open();
```

```
    cmd.ExecuteNonQuery();
    con.Close();
    // Создание объекта DataTable
    DataTable users = new DataTable();
    users.Columns.Add("ID", Type.GetType("System.Int32"));
    users.Columns.Add("NAME", Type.GetType("System.String"));
    // Создание объекта DbDataAdapter
    DbDataAdapter adapter = factory.CreateDataAdapter();
    adapter.SelectCommand = factory.CreateCommand();
    adapter.SelectCommand.Connection = con;
    adapter.SelectCommand.CommandText =
      "select ID, NAME from USERS";
    // Заполнение объекта DataTable данными из таблицы БД
    adapter.Fill(users);
    // Отображение данных таблицы
    Console.WriteLine("Данные таблицы:");
    foreach (DataRow row in users.Rows)
    {
      foreach (DataColumn column in users.Columns)
      {
        Console.Write("{0} | ", row[column.ColumnName]);
      }
      Console.WriteLine();
    }
  }
}
```

```
Результат выполнения примера:

Данные таблицы:
0 | Пользователь A |
1 | Пользователь B |
```

InitializeLifetimeService

Метод возвращает объект обслуживания аренды для управления политикой времени существования данного экземпляра класса.

Этот объект является текущим объектом обслуживания аренды во время существования экземпляра класса, если таковой существует; в противном случае, метод создает новый объект обслуживания аренды на время существования экземпляра класса, инициализируя свойство LifetimeServices.LeaseManagerPollTime.

Синтаксис

```
[SecurityPermissionAttribute(SecurityAction.LinkDemand, Flags =
 SecurityPermissionFlag.Infrastructure)]
public virtual Object InitializeLifetimeService();
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Возвращаемое значение

Объект типа ILease (значение System.Object), используемый для управления временем существования (аренды) данного экземпляра класса.

Исключения

SecurityException	У непосредственно вызывающего оператора отсутствует разрешение инфраструктуры.

Пример

Создание аренды:

```
public class MyClass : MarshalByRefObject
{
  [SecurityPermissionAttribute(SecurityAction.Demand,

 Flags=SecurityPermissionFlag.Infrastructure)]
  public override Object InitializeLifetimeService()
  {
    ILease lease = (ILease)base.InitializeLifetimeService();
    if (lease.CurrentState == LeaseState.Initial)
    {
         lease.InitialLeaseTime = TimeSpan.FromMinutes(1);
         lease.SponsorshipTimeout = TimeSpan.FromMinutes(2);
          lease.RenewOnCallTime = TimeSpan.FromSeconds(2);
    }
      return lease;
  }
}
```

ToString

Метод предоставляет строковое значение имени указанного компонента.

Синтаксис

```
public override string ToString();
```

Возвращаемое значение

Строка String, содержащая имя компонента Component, если таковое имеется, или null-значение, если компонент Component является безымянным.

Исключения

Отсутствуют.

Примеры

1) Получение текущего значения строки подключения.

```
// C#
using System;
using System.Data;
using System.Data.Common;
using System.Data.LinterClient;

class ConnectionStringBuilderSample
{
  static void Main()
  {
    // Регистрация фабрики классов провайдера

 DbProviderFactories.RegisterFactory("System.Data.LinterClient",
 typeof(LinterClientFactory));
    // Создание фабрики классов провайдера
    DbProviderFactory factory =
      DbProviderFactories.GetFactory("System.Data.LinterClient");
    // Создание объекта DbConnectionStringBuilder
    DbConnectionStringBuilder builder =
      factory.CreateConnectionStringBuilder();
    builder.ConnectionString =
      "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    // Создание объекта DbConnection
    DbConnection con = factory.CreateConnection();
    con.ConnectionString = builder.ToString();
    try
    {
      // Подключение к БД
      con.Open();
      Console.WriteLine("Установлено подключение. Строка
 подключения:");
      Console.WriteLine(builder.ToString());
    }
    catch (DbException ex)
    {
      // Обработка ошибок
      Console.WriteLine("Ошибка при подключении:");
      Console.WriteLine(ex.Message);
      Console.WriteLine("Строка подключения:");
      Console.WriteLine(builder.ToString());
    }
    Console.WriteLine();
    Console.WriteLine("Нажмите клавишу Ввод для завершения");
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    Console.ReadLine();
  }
}
```

2) Строковое представление текущего исключения.

Информация предоставляется в формате:

Linter Error (code={0}): {1}; System Error (code={2});

где:

{0} – код завершения СУБД ЛИНТЕР;

{1} – текстовая расшифровка кода завершения;

{2} – код операционной системы.

```
using System;
using System.Data.LinterClient;

class Program
{
  public static void ShowLinterException(string connectionString)
  {
    string queryString = "EXECUTE NonExistantStoredProcedure";

    using (LinterDbConnection connection = new
 LinterDbConnection(connectionString))
    {
      LinterDbCommand command = new LinterDbCommand(queryString,
 connection);
      try
      {
        command.Connection.Open();
        command.ExecuteNonQuery();
      }
      catch (LinterSqlException ex)
      {
        Console.WriteLine(ex.ToString());
      }
    }
  }
  static void Main(string[] args)
  {
    ShowLinterException("Data Source=LOCAL;User
 ID=SYSTEM;Password=MANAGER8");
  }
}
```

Результат выполнения примера:

```
Linter Error (code=2229): [Linter error] unknown procedure; System
 Error
(code=2228225);
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

Обработка событий

Одинаковые события могут быть у разных классов (например, событие StateChange есть в классах `SqlConnection`, `OracleConnection`, `LinterDbConnection`). Чтобы определить, какой класс сгенерировал событие, в обработчике события нужно проверить значение параметра sender.

Клиентское приложение может иметь как индивидуальные обработчики событий (для каждого конкретного типа события), так и один универсальный обработчик, например:

```
/* Универсальный обработчик событий для разных провайдеров */
static private void OnStateChange(object sender,
StateChangeEventArgs e)
     {
       if (sender is LinterDbConnection)
      {
         Console.WriteLine("Изменилось состояние подключения
 ЛИНТЕР:");
       }
       else if (sender is OracleConnection)
       {
         Console.WriteLine("Изменилось состояние подключения
 Oracle:");
       }
       else
       {
         Console.WriteLine(
           "Изменилось состояние подключения неизвестного
 провайдера:");
       }
       Console.WriteLine("  Начальное состояние = " +
 e.OriginalState);
       Console.WriteLine("  Текущее состояние = " +
 e.CurrentState);
     }

...
/* Добавляем обработчик событий */

linterDataAdapter.SelectCommand.Connection.StateChange +=
new StateChangeEventHandler(OnStateChange);

oracleDataAdapter.SelectCommand.Connection.StateChange +=
new StateChangeEventHandler(OnStateChange);
...
```

Пример обработки событий.

```
using System;
using System.Data;
using System.Data.LinterClient;

namespace Test
{
  class Program
  {
    /* Обработка событий ADO.NET-провайдера */
    static private void OnStateChange(object sender,
 StateChangeEventArgs e)
    {
      Console.WriteLine("Изменилось состояние подключения:");
      Console.WriteLine("  Начальное состояние = " +
 e.OriginalState);
      Console.WriteLine("  Текущее состояние = " +
 e.CurrentState);
    }

    static private void FillDataSet()
    {
      string connectionString = GetConnectionString();
      string queryString = "SELECT MAKE, MODEL FROM AUTO;";

      /* Создаем набор команд и подключения базы данных */
      using (LinterDbDataAdapter dataAdapter =
        new LinterDbDataAdapter(queryString, connectionString))
      {
        /* Добавляем обработчики */
        dataAdapter.SelectCommand.Connection.StateChange
          += new StateChangeEventHandler(OnStateChange);

        /* Создаем расположенный в памяти кэш данных */
        DataSet dataSet = new DataSet();

        /* Заполняем кэш данных, что приводит к возникновению
         * нескольких событий изменения состояния подключения */
        dataAdapter.Fill(dataSet, 0, 5, "Table");
      }
    }

    static private string GetConnectionString()
    {
      /* Чтобы не хранить строку подключения в коде, вы можете
```

```
        * получить ее из файла конфигурации */
       return "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
     }

     static void Main(string[] args)
     {
       FillDataSet();
     }
   }
}
```

Обработка исключений

Термин «исключение» обозначает ситуацию, которая может возникать во время выполнения клиентского приложения и которую трудно, а порой и вообще невозможно, предусмотреть во время программирования приложения. Например, попытка подключения к ЛИНТЕР-серверу, который не существует, попытка открытия поврежденного файла или попытка установить связь с компьютером, который находится в автономном режиме. В каждом из этих случаев программист (и конечный пользователь) мало что может сделать с подобными исключительными обстоятельствами.

В подобных случаях CLR будет часто автоматически генерировать соответствующее исключение с описанием текущей проблемы. В классах ADO.NET-провайдера определено множество различных исключений, таких, как IndexOutOfRangeException, FileNotFoundException, ArgumentOutOfRangeException и т.д.

Для обработки исключений используется блок операторов try… catch, который позволяет перехватывать предопределенные ошибочные условия и выполнять соответствующие действия.

Исключения реализованы в виде классов, и если блоки перехвата ожидают появления исключений базового класса до возникновения специфического унаследованного исключения, то это специфическое исключение перехватить не получится.

Исключение содержит читабельное описание проблемы, а также детальный снимок стека вызовов на момент, когда изначально возникло исключение. Более того, конечному пользователю можно предоставлять справочную ссылку, которая указывает на определенный URL-адрес с описанием деталей ошибки, а также специальные данные, определенные программистом.

Обработка исключений подразумевает использование следующих связанных между собой сущностей:

1) тип класса, который представляет детали исключения;
2) член класса, способный генерировать (throw) в вызывающем коде экземпляр класса исключения при соответствующих обстоятельствах;
3) блок кода на вызывающей стороне, ответственный за обращение к члену, в котором может произойти исключение;
4) блок кода на вызывающей стороне, который будет обрабатывать (или перехватывать (catch)) исключение в случае его возникновения.

Примечание
Полный список сообщений LinterSqlException и рекомендации по устранению ошибок см. в документе «Справочник кодов завершения».

Пример обработки исключений

```
// C#
using System;
using System.Data.LinterClient;

class ExceptionSample
{
```


© Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
  static void Main()
  {
    string connectionString = "User ID=SYSTEM;Password=MANAGER8";
    string queryString = "execute SAMPLE_PROCEDURE";

    LinterDbConnection connection = null;
    LinterDbCommand command = null;
    try
    {
      connection = new LinterDbConnection(connectionString);
      command = new LinterDbCommand(queryString, connection);
      command.Connection.Open();
      command.ExecuteNonQuery();
    }
    catch (LinterSqlException ex)
    {
      Console.WriteLine(
        "Исключение ядра СУБД ЛИНТЕР \n" +
        "Текст сообщения: " + ex.Message + "\n" +
        "Код СУБД ЛИНТЕР: " + ex.Number + "\n" +
        "Код операционной системы: " + ex.LinterSysErrorCode +
 "\n");
    }
    catch (Exception ex)
    {
      Console.WriteLine(
        "Исключение ADO.NET провайдера \n" +
        "Тип ошибки: " + ex.GetType() + "\n" +
        "Сообщение: " + ex.Message + "\n");
    }
    finally
    {
      if (connection != null)
      {
        connection.Close();
      }
    }
  }
}
```

Результат выполнения примера:

```
 Исключение ядра СУБД ЛИНТЕР
Текст сообщения: [Linter error] unknown procedure
Код СУБД ЛИНТЕР: 2229
Код операционной системы: 1572865
```

Прерывание запроса

Если запрос выполняется слишком долго, то прервать его выполнение можно с помощью метода `DbCommand.Cancel`.



Примечание
В текущей версии ADO.NET-провайдера прерывание запроса с помощью метода `Thread.Abort` не поддерживается.

Провайдер Entity Framework 6

Примечание
Поддержка остановлена, использовать не рекомендуется.

Для использования Linter с Entity Framework 6 необходимо добавить в проект зависимости на пакеты `EntityFramework`, `EntityFramework.Linter` и `System.Data.LinterClient` при помощи менеджера пакетов NuGet, зарегистрировать и сконфигурировать для испоьлзования классы LinterClientFactory и LinterProviderServices. После этого можно использовать стандартные методы Entity Framework для разработки приложения. Например, следующая программа сохраняет в БД время запуска и выводит на экран список всех запусков. Данный пример можно использовать для создания журнала посещений, где каждый запуск приложения соответствует визиту одного человека:

```
// C#
using System;
using System.Data.Entity;

namespace CodeFirstDemo
{
  class Program
  {
    static void Main(string[] args)
    {
      // Регистрация фабрики классов провайдера
      DbConfiguration.SetConfiguration(new
 LinterContextConfiguration());

      DbProviderFactory factory =
 DbProviderFactories.GetFactory("System.Data.LinterClient");
      DbConnectionStringBuilder builder =
 factory.CreateConnectionStringBuilder();
      builder.ConnectionString = "Data Source=LOCAL;User
 ID=SYSTEM;Password=MANAGER8";
      DbConnection con = factory.CreateConnection();
      con.ConnectionString = builder.ToString();
      con.Open();
      using (var context = new VisitContext(con))
      {
        // Подключение к БД и создание таблицы, если она не
 существует
        context.Database.CreateIfNotExists();

        // Добавление новой записи и сохранение изменений
        context.Visits.Add(
          new Visit { Name = "Пользователь A", Date =
 DateTime.Now });
```

```
        context.SaveChanges();

        // Чтение записей из БД
        Console.WriteLine("Список визитов:");
        foreach (var visit in context.Visits)
        {
          Console.WriteLine(visit.Id + " | " + visit.Name + " | "
+ visit.Date);
        }

        Console.WriteLine("Для продолжения нажмите любую
клавишу...");
        Console.ReadKey();
      }
    }
 }

 class LinterContextConfiguration : DbConfiguration
 {
   public LinterContextConfiguration()
   {

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

DbProviderFactories.RegisterFactory("System.Data.LinterClient",
typeof(LinterClientFactory));
     SetProviderServices("System.Data.LinterClient",
LinterProviderServices.Instance);
   }
 };

 public class VisitContext : DbContext
 {
   public VisitContext(DbConnection con) : base(con, true)
   {
   }

   protected override void OnModelCreating(DbModelBuilder
modelBuilder)
   {
       base.OnModelCreating(modelBuilder);
       modelBuilder.HasDefaultSchema("SYSTEM");
   }

   public DbSet<Visit> Visits { get; set; }
 }
```

```
}
```

При первом запуске приложения в БД будет создана схема "SYSTEM" и таблица "SYSTEM"."Visits", если запуск производится на платформе, поддерживаемой миграциями EF6.

Таблицу также можно создать на этапе разработки приложения с помощью автоматических миграций Code First Migrations. Для этого в меню Visual Studio выбрать **Tools(Сервис)=>NuGet Package Manager(Диспетчер пакетов NuGet)=>Package Manager Console(Консоль диспетчера пакетов)** и в появившемся окне после приглашения PM> ввести команды:

PM> Enable-Migrations -EnableAutomaticMigrations
PM> Add-Migration Visit_create
PM> Update-Database

Примечание
Если контекст унаследован от класса ObjectContext, то методы CreateDatabase() и DeleteDatabase() выполняют создание и удаление таблиц БД соответственно.
Если контекст унаследован от класса DbContext, то удаление таблиц методом DeleteDatabase() не поддерживается в текущей версии.

Строки подключения

Строка подключения Code First является обычной строкой ADO.NET (см. пункт «ConnectionString»).

Примечание
Если для аутентификации используется имя пользователя и пароль, то в строке подключения Code First надо указать параметр Persist Security Info=true.

Пример

```
string connectionString =
  "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8;Persist
 Security Info=true";
```

Строка подключения Database First и Model First является специальным типом строки подключения, формат которой приведён в таблице 46.

Пример

```
string connectionString =
  "metadata=entitydemo.edmx.ssdl|entitydemo.edmx.csdl|
entitydemo.edmx.msl;" +
  "provider=System.Data.LinterClient;" +
  "provider connection string=\"Data Source=LOCAL;User
 ID=SYSTEM;Password=MANAGER8\"";
```

Таблица 46. Формат строки подключения Database First и Model First

Ключ	Значение
metadata	Список каталогов, файлов или ресурсов, в которых находится информация объектно-реляционного

Ключ	Значение
	сопоставления и метаданные. Значения в списке должны быть разделены символом «|». Пример: `metadata=entitydemo.edmx.ssdl	entitydemo.edmx.csdl	entitydemo.edmx.msl`
`provider`	Имя провайдера Entity Framework. Пример: `provider=System.Data.LinterClient`
`provider connection string`	Строка подключения ADO.NET провайдера (см. пункт «ConnectionString»). Пример: `provider connection string= \"Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8\"`

Отображение типов данных

В таблице 47 представлено отображение типов данных .NET Framework на типы данных СУБД ЛИНТЕР, которое реализовано по умолчанию в провайдере Entity Framework.

Таблица 47. Отображение типов .NET Framework на типы СУБД ЛИНТЕР

Тип данных .NET Framework	Тип данных СУБД ЛИНТЕР
`System.Int16`	SMALLINT
`System.Int32`	INTEGER
`System.Int64`	BIGINT
`System.String`	NVARCHAR(2000)
`System.Single`	REAL
`System.Double`	DOUBLE
`System.DateTime`	DATE
`System.Boolean`	BOOLEAN
`System.Decimal`	DECIMAL(18, 2)
`System.Guid`	BYTE(16)
`System.Byte`	BYTE(1)
`System.Byte[] длиной до 4000 байт включительно`	VARBYTE(4000)
`System.Byte[] длиной больше 4000 байт`	BLOB

Примечание
Длину можно установить с помощью атрибута `[MaxLength(<длина>)]`.

Пример

```
public class Class1
{
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
  public Int32 Id { get; set; }
  public Int16 Int16Property { get; set; }
  public Int32 Int32Property { get; set; }
  public Int64 Int64Property { get; set; }
  public String StringProperty { get; set; }
  public Single SingleProperty { get; set; }
  public Double DoubleProperty { get; set; }
  public DateTime DateTimeProperty { get; set; }
  public Boolean BooleanProperty { get; set; }
  public Decimal DecimalProperty { get; set; }
  public Guid GuidProperty { get; set; }
  public Byte ByteProperty { get; set; }
  [MaxLength(4000)]
  public Byte[] BytesProperty { get; set; }
  public Byte[] BytesPropertyBlob { get; set; }
}
Для этого класса в БД будет создана следующая таблица:
create if not exists table "dbo"."Class1"
   (  "Id" int not null autoinc,
      "Int16Property" smallint not null,
      "Int32Property" int not null,
      "Int64Property" bigint not null,
      "StringProperty" nvarchar(2000),
      "SingleProperty" real not null,
      "DoubleProperty" double not null,
      "DateTimeProperty" date not null,
      "BooleanProperty" boolean not null,
      "DecimalProperty" decimal(18, 2) not null,
      "GuidProperty" byte(16) not null,
      "ByteProperty" byte(1) not null,
      "BytesProperty" varbyte(4000),
      "BytesPropertyBlob" blob,
      primary key ("Id") );
```

Использование хранимых процедур для выполнения операций INSERT|UPDATE|DELETE

Хранимые процедуры можно использовать, если свойства объектов имеют следующие типы:

- System.Int16;
- System.Int32;
- System.Int64;
- System.Single;
- System.Double;
- System.DateTime;

- System.Boolean;
- System.Decimal;
- System.Guid;
- System.Byte[] длиной до 3919 байт;
- System.String длиной до 1959 символов.

Для использования хранимых процедур надо в классе контекста переопределить метод `OnModelCreating(DbModelBuilder modelBuilder)` и для каждого пользовательского класса добавить следующую строку:

modelBuilder.Entity<[класс]>().MapToStoredProcedures();

Пример

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace Demo
{
  // Класс автомобиль
  public class Auto
  {
    // Уникальный идентификатор объекта
    public int Id { get; set; }

    // Название автомобиля
    [MaxLength(1959)]
    public string Name { get; set; }
  }

  // Класс контекст для доступа к базе данных
  public class AutoContext : DbContext
  {
    public AutoContext()
      : base("User ID=SYSTEM;Password=MANAGER8;Persist Security
 Info=true")
    {
    }

    protected override void OnModelCreating(DbModelBuilder
 modelBuilder)
    {
      // Отображение операций INSERT|UPDATE|DELETE на хранимые
 процедуры
      modelBuilder.Entity<Auto>().MapToStoredProcedures();
    }
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
    public DbSet<Auto> Autos { get; set; }
  }

  class Program
  {
    static void Main(string[] args)
    {
      using (var context = new AutoContext())
      {
        // Подключение к базе данных и создание таблицы, если она
 не существует
        context.Database.CreateIfNotExists();

        // Добавление новой записи и сохранение изменений
        context.Autos.Add(new Auto { Name = "Автомобиль A" });
        context.SaveChanges();

        // Чтение записей из базы данных
        Console.WriteLine("Список авто:");
        foreach (var auto in context.Autos)
        {
          Console.WriteLine(auto.Id + " | " + auto.Name);
        }

        Console.WriteLine("Для продолжения нажмите любую
 клавишу...");
        Console.ReadKey();
      }
    }
  }
}
```

После выполнения примера в БД будут созданы три хранимые процедуры: "dbo"."Auto_Insert", "dbo"."Auto_Update" и "dbo"."Auto_Delete", которые можно изменить. Для этого в заголовке хранимых процедур необходимо заменить выражение `CREATE IF NOT EXISTS` на выражение `CREATE OR REPLACE`, внести необходимые изменения в код и сохранить хранимые процедуры:

```
create or replace procedure "dbo"."Auto_Insert"(in Name
 nvarchar(1959)) result cursor(Id int)
declare
  var c typeof(result);
code
  execute "insert into \"dbo\".\"Autoes\" (\"Name\") values (?)"
 using Name;
  open c for direct "select \"Id\" from \"dbo\".\"Autoes\" where
 \"Id\" = LAST_AUTOINC";
```

```
  return c;
end;

create or replace procedure "dbo"."Auto_Update"(in Id int; in Name
 nvarchar(1959))
code
  execute "update \"dbo\".\"Autoes\" set \"Name\" = ? where \"Id\"
 = ?" using Name,Id;
end;

create or replace procedure procedure "dbo"."Auto_Delete"(in Id
 int)
code
  execute "delete from \"dbo\".\"Autoes\" where \"Id\" = ?" using
 Id;
end;
```

Приложение 1
Освобождение ресурсов

Для освобождения ресурсов надо использовать оператор using или блок **try...finally**. Во многих примерах данного документа они опущены с целью ограничения размера документа, но при разработке реальных приложений они обязательно должны использоваться.

Оператор **using**

```
// C#
using System;
using System.Data;
using System.Data.LinterClient;

class UsingSample
{
  static void Main()
  {
    // Создание соединения
    using (LinterDbConnection con = new LinterDbConnection())
    {
      con.ConnectionString = "Data Source=LOCAL;User
 ID=SYSTEM;Password=MANAGER8";
      con.Open();

      // Создание команды
      using (LinterDbCommand cmd = new LinterDbCommand())
      {
        cmd.Connection = con;
        cmd.CommandText = "select 'СУБД', 'ЛИНТЕР'";

        // Выполнение команды
        using (LinterDbDataReader reader = cmd.ExecuteReader())
        {
          // Получение данных
          while (reader.Read())
          {
            Console.WriteLine(reader[0] + " " + reader[1]);
          }
        }
      }
    }
  }
}
```

Блок **try...finally**

```
// C#
```

```
using System;
using System.Data;
using System.Data.LinterClient;

class TryFinallySample
{
  static void Main()
  {
    LinterDbConnection con = null;
    LinterDbCommand cmd = null;
    LinterDbDataReader reader = null;

    try
    {
      // Создание соединения
      con = new LinterDbConnection();
      con.ConnectionString = "Data Source=LOCAL;User
 ID=SYSTEM;Password=MANAGER8";
      con.Open();

      // Создание команды
      cmd = new LinterDbCommand();
      cmd.Connection = con;
      cmd.CommandText = "select 'СУБД', 'ЛИНТЕР'";

      // Выполнение команды
      reader = cmd.ExecuteReader();

      // Получение данных
      while (reader.Read())
      {
        Console.WriteLine(reader[0] + " " + reader[1]);
      }
    }
    finally
    {
      // Освобождение ресурсов
      if (reader != null)
      {
        reader.Close();
      }
      if (cmd != null)
      {
        cmd.Dispose();
      }
      if (con != null)
```

 © Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
        {
          con.Close();
        }
      }
    }
}
```

Приложение 2
Пример асинхронной обработки данных

```
using System;
using System.Data;
using System.Data.LinterClient;
using System.Threading;

namespace Test
{
  class Program
  {
    /* Асинхронная работа */
    static void Main(string[] args)
    {
      /* Создание тестовой таблицы */
      CreateDataTable();
      /* Создание разных нитей для поиска минимального
       * и максимального значений в таблице */
      Thread findMinThread = new Thread(FindMin);
      Thread findMaxThread = new Thread(FindMax);
      /* Старт нитей */
      findMinThread.Start();
      findMaxThread.Start();
    }

    static private void FindMin()
    {
      Console.WriteLine("Поиск минимального значения...");
      LinterDbConnection conn = new LinterDbConnection();
      conn.ConnectionString = GetConnectionString();
      conn.Open();
      LinterDbCommand comm = conn.CreateCommand();
      comm.Connection = conn;
      comm.CommandText = "SELECT MIN(X) FROM T";
      object result = comm.ExecuteScalar();
      conn.Close();
      /* Имитируем длительную работу */
      Thread.Sleep(5000);
      Console.WriteLine("Минимальное значение: " + result);
    }

    static private void FindMax()
    {
```


© Поставщики данных ADO.NET. АО НПП «РЕЛЭКС», 1990-2026

```
      Console.WriteLine("Поиск максимального значения...");
      LinterDbConnection conn = new LinterDbConnection();
      conn.ConnectionString = GetConnectionString();
      conn.Open();
      LinterDbCommand comm = conn.CreateCommand();
      comm.Connection = conn;
      comm.CommandText = "SELECT MAX(X) FROM T";
      object result = comm.ExecuteScalar();
      conn.Close();
      /* Имитируем длительную работу */
      Thread.Sleep(7000);
      Console.WriteLine("Максимальное значение: " + result);
    }

    static private void CreateDataTable()
    {
      Console.Write("Создание тестовой таблицы...");
      LinterDbConnection conn = new LinterDbConnection();
      conn.ConnectionString = GetConnectionString();
      conn.Open();
      LinterDbCommand comm = conn.CreateCommand();
      comm.Connection = conn;
      comm.CommandText = "CREATE OR REPLACE TABLE T(X REAL)";
      comm.ExecuteNonQuery();
      comm.CommandText = "INSERT INTO T (X) VALUES (RAND())";
      for (int i = 0; i <= 10000; i++)
      {
        comm.ExecuteNonQuery();
      }
      conn.Close();
      Console.WriteLine("завершено.");
    }

    static private string GetConnectionString()
    {
      /* Чтобы не хранить строку подключения в коде, вы можете
       * получить ее из файла конфигурации */
      return "Data Source=LOCAL;User ID=SYSTEM;Password=MANAGER8";
    }
  }
}
```