ACTIAN™

Zen Direct Access Components Guide

Zen v16

Activate Your Data™

目次

Direct Access Components の使用方法

以下のトピックでは、Delphi および C++Builder で使用する Zen Direct Access Components（PDAC）について説明します。

- Zen Direct Access Components の概要
- Zen Direct Access Components の使用法
- PDAC に基づくアプリケーションの配布

この機能の実装方法については、Direct Access Components リファレンスを参照してください。

Zen Direct Access Components の概要

Zen Direct Access Components は、Borland Delphi と C++Builder 環境内で、Zen データベース エンジンへ直接トランザクショナルおよびリレーショナル アクセスできるようにする、一連の Visual Component Library（VCL）コンポーネントです。この SDK は、以前は "Pervasive Direct Access Components（PDAC）" と呼ばれていました。この略称 PDAC は現在も SDK で使用されています。Zen v15 より前のリリースでは、これらのランタイム コンポーネントは統合され、Zen データベースの 32 ビットおよび 64 ビットの ¥bin ディレクトリに .bpl ファイルとしてインストールされていました。v15 以降、これらのコンポーネンは Zen インストールから削除されましたが、ダウンロードにより入手される SDK で引き続き利用することができます。これには、RAD Studio 用の設計時コンポーネントも含まれています。

サポートされる開発環境については、PDAC のリリース ノートを参照してください。

PDAC のすべてのバージョンには、次の非ビジュアル コンポーネントが含まれています。

- TPvSession
- TPvDatabase
- TPvTable
- TPvBatchMove
- TPvQuery

- TPvStoredProc
- TPvUpdateSQL
- TPvSqlDatabase

これらの 32 ビット コンポーネントのサポート クラスも提供されています。これらのコンポーネントは、Embarcadero の Data Access Components のプロパティ、メソッド、およびバインド機能を再現でき、実行時に Borland Database Engine（BDE）が存在している必要はありません。

これらのコンポーネントはパッケージ形式で提供され、組み込みコントロールの設計時および実行時に必要なすべての機能を提供します。これらは BDE コンポーネントと同様の方法で Embarcadero データ アウェア コントロールにバインドします。また、完全に互換性のある、サード パーティのバウンド コントロールにもバインドします。

Woll2Woll InfoPower コンポーネントとの相互運用性のための特別なコンポーネント（wwPvTable、wwPvQuery および wwPvStoredProc）が装備されています。

エンジンのバージョンのチェック

PDAC では、クエリまたはテーブルを開いたときにデータベース エンジンのバージョンを確認して、エンジンのバージョン依存に伴うエラーが発生しないようにしています。エンジンのバージョンが不適切な場合は、必要なバージョンを示す例外を返します。

TPvQuery.PassThrough

SQL クエリ テキストを事前解析しないで、エンジンへ直接送ることができます。

TPvDatabase.OEMConversion

このプロパティは、データベースに格納されている文字は OEM（DOS）コード ページによってエンコードされているため、データベースを使用する前にこれらの文字を ANSI（Windows）コード ページに変換する必要があることを示します。

詳細情報を入手するには

PDAC 用の SDK ダウンロードを参照してください。これは弊社 Web サイトにあります。

Zen Direct Access Components の使用法

PDAC は、アーカイブからそのコンポーネントをディレクトリ（*installdir*）に展開することでインストールします。システム上の Delphi または C++Builder IDE が、その展開されたコンポーネントの場所を認識できることを確認してください。

C++Builder のプロジェクト オプションに、PDAC のためのインクルード パスとライブラリ パスを反映させるためには以下の手順に従います。これらはデフォルトのオプションとして追加することができます。そのため、Zen コンポーネントを使用しているすべてのプロジェクトに必要というわけではありません。

PDAC を C++Builder のインクルード パスとライブラリ パスに追加するには

1. C++Builder で［**プロジェクト**］>［**オプション**］を選択して、［**ディレクトリ / 条件**］タブをクリックします。
2. ［インクルード パス］に *installdir*`¥sdk¥pdac¥Delphi?¥include` を追加します。ここで、*installdir* は PDAC をインストールしたディレクトリを示します。疑問符（?）は使用している RAD Studio のバージョンに置き換えます。
3. さらに［ライブラリ パス］に、*installdir*`¥sdk¥pdac¥Delphi?¥lib` と、*installdir*`¥sdk¥pdac¥Delphi?¥lib¥dcu` をセミコロン（;）で区切って追加します。疑問符（?）は使用している RAD Studio のバージョンに置き換えます。

インクルード パスとライブラリ パスを正しく設定すると、Direct Access Components を対応する Embarcadero コンポーネントとまったく同様に使用できるようになります。

Delphi および C++Builder

Zen では Delphi 6 以降のすべての Delphi バージョンをサポートします。お使いの IDE に Delphi または C++Builder を組み入れるには、次の手順を使用します。

Delphi または C++Builder パッケージとパス情報をお使いの環境に追加するには

1. Delphi または C++Builder で［**コンポーネント**］>［**パッケージのインストール**］>［**追加**］をクリックします。
2. 設計時パッケージを選択します。

3. Delphi compiled units（DCU）とインターフェイス ファイル（INT）にパスを追加します。

お使いの環境に合ったコンポーネントを選択すると、VCL プロジェクトに Zen が表示されるようになります。これで、PvTable または PvQuery を追加して、PDAC を使用するアプリケーションを構築することが可能になりました。

PDAC に基づくアプリケーションの配布

Embarcadero などのランタイム ライブラリ用にはいくつかのバリエーションがありますが、エンド ユーザーに配布する PDAC アプリケーションは 2 つの方法で構築できます。1 つは実行時パッケージを使用する方法、もう 1 つはアプリケーションの実行可能ファイルに PDAC ライブラリを静的にリンクする方法です。また、C++Builder では Embarcadero ライブラリの動的または静的リンクも提供します。PDAC アプリケーションを正常に実行させるためには、Zen のクライアントまたはエンジン（Enterprise Server、Cloud Server、Workgroup）のいずれかがターゲット コンピューターに適切にインストールされている必要があります。

実行時パッケージの使用の決定

実行時パッケージを使用する場合はいくつかの利点があります。最小の実行可能ファイルを提供することができます。これは、リンクの遅い環境で実行可能ファイルまたは更新のダウンロードや転送を行う必要がある場合には重要になります。このオプションを使用すれば、PDAC および Embarcadero のランタイム バイナリを共有することができ、多くの Embarcadero および PDAC アプリケーションを使用する環境内のディスク容量やメモリを大幅に節約することができます。

欠点は、より多くのファイルを配布し、それらを最新のものにしておく必要があること、またその共有ファイルは、共有サブディレクトリにサード パーティ製の不適切なバージョンの共有ファイルがインストールされることによって影響を受ける恐れがあることです。実行時パッケージを使用しない場合は、アプリケーションと drm.dll のみを配布する必要があります。PDAC ライブラリは実行可能ファイルに静的にリンクされるため、それに伴ってサイズは大きくなります。

実行時パッケージを使用する場合、開発者はアプリケーションの実行可能ファイルおよび PDAC ライブラリ（PDAC インストールの redist サブディレクトリ内）の両方と、Embarcadero ランタイム ライブラリまたはその他のランタイム ライブラリ（コンパイ

ラの bin ディレクトリにある borlndmm.dll が必要になることが多い）をエンド ユーザーに配布する必要があります。

Redist サブディレクトリ

Embarcadero の各コンパイラには、redist ディレクトリがあります。このディレクトリには、.bpl ライブラリが含まれており、これらのライブラリは［**実行時パッケージを使って構築**］オプションを使ってコンパイルされたアプリケーションと一緒に配布する必要があります。

メモ： 各 redist ディレクトリには .dcp または .bcp ファイルも含まれています。これらのファイルは、派生クラスを使用した開発が許可されるライセンスを持つ組織以外には再配布されません。.dpl パッケージと .bpl パッケージのみをアプリケーションと一緒に配布することができます。

Delphi または C++Builder の構築と配布手順

Delphi IDE で以下のように適切なオプションを設定します。

1. ［**プロジェクト**］メニューから［**オプション**］を選択します。
2. ［**パッケージ**］タブをクリックします。

   実行時にアプリケーションと PDAC ライブラリとを動的にリンクさせるには、［**実行時パッケージを使って構築**］のチェックをオンにし、適切な PDAC パッケージを編集ボックスのリストへ追加します。

メモ： インストール時に PDAC パッケージが構築リストに追加されていた場合は、上記の手順は必要ありません。

3. Delphi のバージョンに応じて、MicroKernel エンジンに必要となる 32 ビットまたは 64 ビット実行時パッケージ（.bpl ファイル）を選択します。

メモ： PDAC パッケージをリストに追加しない場合、これらは実行可能ファイルにコンパイルされ、［実行時パッケージを使って構築］のチェックがオンになっていても共有ライブラリは使用されません。

4. PDAC ライブラリを実行可能ファイルに静的にリンクさせるには、［**実行時パッケージを使って構築**］チェック ボックスをオフにしておきます。コンパイル時にこの設定をオンにしていた場合は、アプリケーションを配布するときに、ライブラリとその他必要なパッケージや DLL も配布する必要があります。

これらのファイルは、アプリケーションの実行可能ファイルと同じサブディレクトリにインストールすることができますが、他のアプリケーションと共有できるよう、ターゲット コンピューターのパスに置くことをお勧めします。Zen インストール場所の ¥bin ディレクトリが Zen 実行モジュールの主ディレクトリである場合は、このディレクトリを使用するのが適切です。

さらに、Delphi の PDAC アプリケーションでは、Zen エンジン（サーバーまたはワークグループ）または Zen クライアントのいずれかと、適切にインストールされ設定されているリモート サーバーが必要です。

Delphi 6 より更新されたパッケージ名

Delphi 6 以降、弊社では Embarcadero 標準との互換性をより強化するようパッケージの名前付け規則を変更しました。また、Embarcadero 要件を満たすために、実行時パッケージを設定時パッケージから分けました。

旧バージョンの Delphi と PDAC から移行する場合の、パッケージ名の解釈方法を以下の表に示します。

ファイル名の桁位置	可能な値	説明
1	p	PSQL（Zen 製品の以前の名前）
2	c、b または s	Common、Btrieve または SQL
3 － 5	リリースによって異なる	PDAC コンポーネントのメジャーおよびマイナー バージョン
6	r、d	実行時または設計時
7 － 9	リリースによって異なる	これらのコンポーネントを適用する Delphi バージョン

メモ： Delphi 6 では、InfoPower はメイン パッケージに結合されており、Woll2Woll は独立していません。

Direct Access Components リファレンス

以下のトピックでは、Delphi および C++Builder 開発者向けの詳細な情報を提供します。

- MicroKernel エンジン コンポーネント
- リレーショナル エンジン コンポーネント
- PDAC と Embarcadero のコンポーネントの相違点
- PDAC クラス、プロパティ、イベント、およびメソッド
- Zen および Embarcadero のデータ型

MicroKernel エンジン コンポーネント

Btrieve コンポーネントのセットには以下のものが含まれています。

- TPvSession
- TPvDatabase
- TPvTable
- TPvBatchMove
- TwwPvTable

これらのコンポーネントはリレーショナル機能を使用しておらず、実行時に リレーショナル エンジンまたは ODBC を必要としません。

コンポーネントは、次の表に示すように Embarcadero のコンポーネントに置き換わります。

Zen Direct Access Component	置き換える Embarcadero コンポーネント
TPvBatchMove	TBatchMove
TPvDatabase	TDatabase
TPvSession	TSession
TPvTable、TwwPvTable	TTable

これらのコンポーネントはスタンドアロン（つまり、Borland Database Engine（BDE）なし）で使用することができます。また、単一アプリケーション内で BDE と同時に使用することもできます。コンポーネントの使用法は対応する BDE コンポーネントと同じです。例外については、適用可能な特定のコンポーネント下、および PDAC クラス、プロパティ、イベント、およびメソッドに記載されています。

TPvSession

スレッドに対する安全性とクライアント ID のサポートを提供します。この機能は、TSession VCL コンポーネントと類似しています。

関連情報

TPvSession については、本章の以下のセクションを参照してください。

- TPvSessionList/TpvSqlSessionList クラス（サポート クラス）
- グローバル変数（VCL との全般的な違い）
- TPvSession と TPvSqlSession（VCL と異なる特定のクラス）
- TPvSession に関する注記（VCL と異なる特定のクラス）

TPvDatabase

ログイン コントロール、トランザクション サポート、データベース接続持続性などの、非 SQL データベースのための接続に特化した機能を提供します。この機能は、TDatabase VCL コンポーネントと類似しています。

TPvDatabase と Embarcadero のコンポーネントの機能の相違点

- DriverName、Locale、および TTraceFlags プロパティは削除されています。
- ハンドルのタイプが DRM_DBID に変更されます。
- *IsSQLBased* は常に False を返します。
- TransIsolation に指定できる値は、tiDirtyRead と tiReadCommitted（デフォルト）の 2 つのみです。

セキュリティで保護されたテーブル、およびユーザー名とパスワードの入力要求

セキュリティで保護されたテーブルに接続しようとすると、PDAC にユーザー名とパスワードの入力を求められます。このようなプロンプトが表示されないようにしたい場合は、次のコードを例として用いて、TPvDatabase で接続パラメーターを設定します。（TPvTable のテーブル セキュリティも参照してください。）

```
PvSession.ServerName:='ServerName';
PvSession.SessionName:='session1';
PvSession.Active:=True;

PvDatabase.AliasName:= 'DatabaseName';
PvDatabase.DatabaseName:='DB';
PvDatabase.SessionName:='session1';
PvDatabase.Params.Clear();

// ここでユーザー名とパスワードを指定して
// データベースに接続する

PvDatabase.Params.Add('User Name=UserName');
PvDatabase.Params.Add('password=Password');
PvDatabase.Connected:=True;
PvTable.DatabaseName:='DB';
PvTable.SessionName:='session1';
PvTable.TableName:='person';
PvTable.Active:=True;
```

関連情報

TPvDatabase については、本章の以下のセクションを参照してください。

- TransIsolation プロパティの相違点（PDAC と Embarcadero のコンポーネントの相違点）
- TPvDatabase と TPvSqlDatabase（VCL と異なる特定のクラス）
- TPvDatabase に関する注記（VCL と異なる特定のクラス）

TPvTable

シングル テーブル アクセスと操作を提供します。この機能は、TTable VCL コンポーネントと類似しています。

TPvTable と Embarcadero のコンポーネントの機能の相違点

- すべてのエイリアス管理機能で「名前付きデータベース」管理を行います。
- TDBDataSet からではなく TPvDataSet から継承します。

- プロパティ TableLevel、UpdateObject、UnlockTable、OpenIndexFile、CloseIndexFile は削除されました。TableType は無視されます。
- 設計時と実行時のどちらででも、DSN、Zen 名前付きデータベース、あるいはデータベースへの完全修飾パスとして、Database Name プロパティを利用できます。

テーブル セキュリティ

MicroKernel エンジン用の PDAC セキュリティは、個々のテーブルのオーナー ネームを使用します。このセキュリティ モデルを使用する場合は、テーブルのオーナー ネームを提供します。詳細については、『*Btrieve API Guide*』の Set Owner（29）を参照してください。

TPvTable には `Owner: string` プロパティと次のメソッドがあります。

```
SetOwnerOnTable(AOwner: string; AccessMode: integer)
```

Owner プロパティを使用すれば、オーナー ネームを設定することができます。*SetOwnerOnTable* メソッドを使用すれば、テーブルのオーナー ネームを設定したりクリアしたりすることができます。

コード	説明
0	すべてのアクセス モードでオーナー ネームが必要（データ暗号化なし）
1	読み取り専用アクセスにはオーナー ネームは必要なし（データ暗号化なし）
2	すべてのアクセス モードでオーナー ネームが必要（データ暗号化あり）
3	読み取り専用アクセスにはオーナー ネームは必要なし（データ暗号化あり）

これらの PDAC アクセス モードによって制御されるように、Btrieve テーブルにアクセスするには、有効なオーナー ネームを指定します。

テーブルの作成

実行時に CreateTable メソッドを呼び出し、このデータセットの現在の定義を使用してテーブルを作成します。

FieldDefs プロパティに値がある場合は、これらの値を使ってフィールド定義を作成します。値がない場合は、Fields プロパティを使用します。データセットを再作成するには、これらのプロパティの一方または両方に値がなければなりません。

Add メソッドを使用してフィールドのプロパティを割り当てます。

```
procedure Add(const Name: string; DataType: TFieldType; Size: Word; Required: Boolean);
```

Add メソッドは以前のバージョンとの互換性を保つために提供されます。新しいフィールド定義を Item プロパティの配列に追加する場合は、AddFieldDef メソッドを使用することをお勧めします。また、これを使って ftBCD データ型の小数位と桁数も指定します。

Add メソッドは、Name、DataType、Size および Required パラメーターに渡された値を使用し、これらを新しいフィールド定義オブジェクトの個別のプロパティに割り当てます。

メモ： フィールドがヌル値を許可する場合、**Required** に False を設定します。

メモ： AutoInc フィールドで autoincrement プロパティをアクティブにするには、このフィールドに固有のインデックスを作成します。また、現在のリリースでは、PDAC は BIGIDENTITY データ型をサポートしていません。

Delphi のコード例

```
PvTable1.DatabaseName := 'TestData';
PvTable1.TableName := 'TestData1';
with PvTable1.FieldDefs do
begin
  Clear;
  Add('F_autoinc',  ftAutoInc, 0, True);
  Add('F_currency', ftCurrency, 0, False);
  Add('F_integer',  ftInteger, 0, False);
  Add('F_word',     ftWord, 0, False);
  Add('F_fixchar',  ftFixedchar, 30, False);
  Add('F_varbin',   tString, 25, False);
  Add('F_blob',     ftBlob, 60, False);
end;
with PvTable1.FieldDefs.AddFieldDef do
begin
  Name := 'F_BCD';
  DataType := ftBCD;
  Size:=2;     // 桁数
  Precision := 10;  // 小数位
  Required := false;
end;


with PvTable1.IndexDefs do
begin
  Clear;
  Add('Index1', 'F_autoinc', [ixPrimary, ixUnique]);
```

```
  Add('Index2', 'F_integer', [ixCaseInsensitive]);
end;
PvTable1.CreateTable;
```

C++Builder のコード例

```
PvTable1->DatabaseName="TestData";
PvTable1->TableName="Test1";
PvTable1->FieldDefs->Clear();
PvTable1->FieldDefs->Add("F_autoinc", ftAutoInc, 0, True);
PvTable1->FieldDefs->Add("F_integer", ftInteger, 0, False);
PvTable1->FieldDefs->Add("F_Curr", ftCurrency, 0, False);
PvTable1->FieldDefs->Add("F_Word", ftWord, 0, False);
PvTable1->FieldDefs->Add("F_fixchar", ftFixedChar, 0, False);
PvTable1->FieldDefs->Add("F_String", ftString, 20, False);
PvTable1->FieldDefs->Add("F_blob", ftBlob, 60, False);

TFieldDef *FieldDef = PvTable1->FieldDefs->AddFieldDef();
FieldDef->Name="F_BCD";
FieldDef->DataType=ftBCD;
FieldDef->Size=2;
FieldDef->Precision=10;
FieldDef->Required=False;
PvTable1->IndexDefs->Clear();
PvTable1->IndexDefs-> Add("Index1","F_autoinc",TIndexOptions() <<ixPrimary << ixUnique);
PvTable1->CreateTable();
```

関連情報

TPvTable については、以下のトピックを参照してください。

- TransIsolation プロパティの相違点（PDAC と Embarcadero のコンポーネントの相違点）
- TPvTable、TPvQuery および TPvStoredProc（VCL と異なる特定のクラス）
- TPvTable に関する注記（VCL と異なる特定のクラス）
- Zen および Embarcadero のデータ型

TPvBatchMove

アプリケーションの、レコードのグループあるいはテーブル全体へのデータベース オペレーションを可能にします。この機能は、TBatchMove VCL コンポーネントと類似しています。

関連情報

TPvTable については、本章の以下のセクションを参照してください。

- TPvBatchMove（VCL と異なる特定のクラス）

TwwPvTable

InfoPower 互換性のために PDAC に含まれています。TPvTable から直接派生され、ControlType という追加のプロパティを持ちます。

リレーショナル エンジン コンポーネント

リレーショナル エンジン コンポーネントのセットには以下のものが含まれています。

- TPvSQLSession
- TPvSQLDatabase
- TPvQuery
- TPvUpdateSQL
- TPvStoredProc
- TwwPvQuery
- TwwPvStoredProc

コンポーネントは、次の表に示すように Embarcadero のコンポーネントに置き換わります。

Zen Direct Access Component	置き換える Embarcadero コンポーネント
TPvQuery、TwwPvQuery	TQuery
TPvSqlDatabase	TDatabase
TPvSqlSession	TSession
TpvStoredProc、TwwPvStoredProc	TStoredProc
TPvUpdateSQL	TUpdateSQL

これらのコンポーネントはスタンドアロン（つまり、Borland Database Engine（BDE）なし）で使用することができます。また、単一アプリケーション内で BDE と同時に使用することもできます。コンポーネントの使用法は対応する BDE コンポーネントと同じです。例外については、適用可能な特定のコンポーネント下、および PDAC クラス、プロパティ、イベント、およびメソッドに記載されています。

TPvSQLSession

Zen のスレッドに対する安全性とクライアント ID のサポートを提供します。この機能は、TSession VCL コンポーネントと類似しています。

TPvSQLSession と Embarcadero のコンポーネントの機能の相違点

- すべてのエイリアス管理機能で DSN 管理を行います。
- Handle プロパティのタイプが SQLHENV に変更されます。
- AddStandardAlias メソッドではデフォルトの設定でエンジン DSN を追加します。
- 'UpdateDsnType:TDsnTypes' プロパティが追加されています。TDsnTypes = (dsnSystem, dsnUser)。UpdateDsnType の意味は次のとおりです。
  - DeleteAlias 関数の場合は、DSN ユーザーが System または User DSN のどちらを削除するのか。
  - AddAlias および GetAliasParameters 関数の場合は、どの DSN タイプにするのか。

関連情報

TPvSQLSession については、本章の以下のセクションを参照してください。

- TPvSessionList/TpvSqlSessionList クラス（サポート クラス）
- グローバル変数（VCL との全般的な違い）
- TPvSession と TPvSqlSession（VCL と異なる特定のクラス）
- TPvSqlSession に関する注記（VCL と異なる特定のクラス）

TPvSQLDatabase

ログイン コントロール、トランザクション サポート、データベース接続持続性などの、Zen データベースのための接続に特化した機能を提供します。この機能は、TDatabase VCL コンポーネントと類似しています。

TPvDatabase と Embarcadero のコンポーネントの機能の相違点

- Handle プロパティのタイプが SQLHDBC に変更されています。
- IsSQLBased プロパティは常に True になります。

- Tables と StoredProcs プロパティには、それぞれ TsqlTablesInfoCollection、TsqlStoredProcInfoCollection というタイプがあります。

セキュリティで保護されたデータベース、およびユーザー名とパスワードの入力要求

セキュリティで保護されたデータベースに接続しようとすると、PDAC にユーザー名とパスワードの入力を求められます。このようなプロンプトが表示されないようにしたい場合は、次のコードを例として用いて、TPvSQLDatabase で接続パラメーターを設定します。

```
PvSession.ServerName:='ServerName';
PvSession.SessionName:='session1';
PvSession.Active:=True;

PvDatabase.AliasName:= 'DatabaseName';
PvDatabase.DatabaseName:='DB';
PvDatabase.SessionName:='session1';
PvDatabase.Params.Clear();

// ここでユーザー名とパスワードを指定して
// データベースに接続する

PvDatabase.Params.Add('User Name=UserName');
PvDatabase.Params.Add('password=Password');
PvDatabase.Connected:=True;
PvTable.DatabaseName:='DB';
PvTable.SessionName:='session1';
PvTable.TableName:='person';
PvTable.Active:=True;
```

クライアントからの、DSN を使用しない接続

PDAC のリレーショナル コンポーネントは、DSN または 名前付きデータベースなしでも、クライアント コンピューターからリモート サーバー データベースへ接続できます。サーバーにはデータベースの DSN がなければなりません。

この機能は、TPvSqlDatabase コンポートネントで入手できる **AliasNameIsConnectionString** プロパティを使用することで動作します。詳細については、VCL と異なる特定のクラスを参照してください。

1. Delphi のフォーム上に PvSQLDatabase を配置します。
2. 架空の DatabaseName を設定します。これは何でもかまいません。
3. プロパティ **AliasNameIsConnectionString** を True に設定します。

4. **AliasName** プロパティ（または **AliasName** は空のままで **DatabaseName**）に「接続文字列」を設定します。

   「接続文字列」とは、次の例に示すように、サーバー上の DSN やサーバーの名前などを含む、ODBC への完全な接続文字列です。

   ```
   DRIVER={Pervasive ODBC Client Interface};ServerName=DSLINUX2; ServerDSN=DEMODATA;
   UID=myuserid;PWD=mypassword
   ```

メモ： 接続文字列には引用符や改行はありません。お使いのエディターで行を折り返している場合、プロパティ エディターでは単一行にしてください。

   ユーザー名とパスワードが必要な場合、接続文字列の一部としてこれらを含めることができます。接続文字列にユーザー名とパスワードが含まれない場合、LoginPrompt プロパティが True であれば、標準のデータベース ログイン ダイアログが表示されます。

5. データベースを Connected に設定して通常どおり使用します。

これらの手順はすべて設計時、または実行時にコードから実行することができます。

関連情報

TPvSQLDatabase については、本章の以下のセクションを参照してください。

- TPvDatabase と TPvSqlDatabase（VCL と異なる特定のクラス）
- TPvSqlDatabase に関する注記（VCL と異なる特定のクラス）

TPvQuery

SQL ステートメントに基づいて行セットをカプセル化します。これは、複数のテーブルに対する結合やキャッシュされた更新を含む、完全なリレーショナル アクセスを提供します。この機能は、TQuery VCL コンポーネントと類似しています。

TPvQuery と Embarcadero のコンポーネントの機能の相違点

- TQuery からの継承です。
- Handle プロパティのタイプが SQLHSTMT に変更されています。
- DBHandle プロパティのタイプが SQLHDBC に変更されています。
- StmtHandle プロパティが削除されています。

- Text プロパティは常に SQL プロパティからのテキストを返します。
- CheckOpen メソッドが削除されています。
- インデックスとキーに関連するすべてのプロパティとメソッド（GetIndexInfo など）が削除されています。
- 'LoadBlobOnOpen' プロパティが True のときは、クエリ オープンですべての BLOB がメモリにキャッシュされます。LoadBlobOnOpen が False のときは、BLOB は必要時に読み込まれます。

BookMarksEnabled プロパティ

TPvQuery には BookmarksEnabled というブール型の新しいプロパティが追加されました。アプリケーションでブックマークを使用しない場合は、このプロパティを False に設定すると TPvQuery のパフォーマンスが向上します。デフォルト値は True です。

```
PvQuery.BookmarksEnabled := False
```

カーソル管理

TPvQuery には CursorType プロパティがあり、*ctCached* または *ctDynamic* を設定できます。次の表では、このプロパティの動作を詳しく説明します。

カーソルの種類	動作
ctCached	この設定は旧バージョンで使用可能だった完全キャッシュ、完全静的カーソル マネージャーに相当します。レコードを返す前に ResultSet オブジェクトに各レコードを読み込みます。これはテーブルが大きい場合には速度が遅くなりますが、LookUp などのオペレーション用に開いた後では速くなります。
ctDynamic	この設定がデフォルトです。これはデータベース エンジンの動的カーソルを使用し、ctCached と比較した場合（特に大きなテーブルを扱う場合）、ほとんどのオペレーションでパフォーマンスが大いに向上します。

メモ： 動的カーソルでは、自身またはその他のクライアントによる挿入 / 更新 / 削除を見ることができます。

このプロパティを変更することによって PvQuery の CursorType を変更することはできますが、Active が False になっている場合は実行時のみに限ります。デザイン モードでこれを変更し、クエリが Active である場合は、そのクエリを非アクティブにしてから

カーソルの種類を変更しますが、再アクティブ化はしません。実行時、Active が True になっている場合に PvQuery の CursorType を変更すると、" 開いたデータセットでこのオペレーションを実行できません " という例外が生じる結果となります。

動的カーソルが静的カーソルに変わる状況

動的カーソル（ctDynamic）を要求したとしても、SQL ステートメントに動的カーソルを処理できない構文が含まれている場合、エンジンは接続してデータを返しますが、静的カーソルを使用します。たとえば、次のような構文が挙げられます。

- view
- join
- インデックスが定義されていない列に対する ORDER BY

カーソルが静的カーソルに変わる場合、この静的カーソルはエンジン ベースのカーソルで、ほかのクライアントによる挿入、更新または削除を見ることができません。以前のリリースの静的カーソル（カーソルの種類に *ctCached* を使用する）よりも正しく動作します。

カーソルが動的から静的に変わる場合は、そのことがデータベース エンジンから PDAC コンポーネントに通知され、**EngineCursor** という名前の読み取り専用のパブリック プロパティが設定されます。

EngineCursor プロパティは公開されていないので、設計時に［オブジェクト インスペクター］には表示されません。このプロパティには 2 つの値があります

- *ecDynamic*
- *ecStatic*

このプロパティは変更できませんが、クエリを開いた後にその値を確認することができます。ほかのクライアントによって行われた更新がカーソルに含まれるかどうかをアプリケーション側で認識することが重要な場合などは、このプロパティを確認してください。

関連情報

TPvQuery については、以下の項目を参照してください。

- TPvTable、TPvQuery および TPvStoredProc（VCL と異なる特定のクラス）
- TPvQuery および TPvStoredProc に関する注記（VCL と異なる特定のクラス）

- TPvQuery に関する注記（VCL と異なる特定のクラス）

TPvUpdateSQL

マルチテーブル結合などのほかの方法で更新できない SQL 行セットの（キャッシュ後の更新を含んだ）透過的な更新を可能にします。この機能は、TUpdateSQL VCL コンポーネントと類似しています。

関連情報

TPvTable については、以下の項目を参照してください。

- TPvUpdateSQL（VCL と異なる特定のクラス）

TPvStoredProc

Zen ストアド プロシージャへのアクセスを提供します。TpvQuery コンポーネントと同様に、パラメーター化し、行セットを返すことができます。この機能は、TStoredProc VCL コンポーネントと類似しています。

関連情報

TPvQuery については、以下の項目を参照してください。

- TPvTable、TPvQuery および TPvStoredProc（VCL と異なる特定のクラス）
- TPvQuery および TPvStoredProc に関する注記（VCL と異なる特定のクラス）
- TPvStoredProc に関する注記（VCL と異なる特定のクラス）

TwwPvQuery

InfoPower 互換性のために PDAC に含まれています。TPvQuery から直接派生され、ControlType という追加のプロパティを持ちます。

TwwPvStoredProc

InfoPower 互換性のために PDAC に含まれています。TPvStoredProc から直接派生され、ControlType という追加のプロパティを持ちます。

ODBC と SQL のデータベース セキュリティ

ODBC および SQL のセキュリティは DDF レベルのデータベース セキュリティです。PDAC には SQL セキュリティをセットアップする特別な方法はありません。Zen Control Center または ODBC ツールなどの外部プログラムを使用する必要があります。

データベースがセキュリティで保護されている場合は、テーブルを開くとき、または TpvDatabase コンポーネントにアクセスするときに、毎回ユーザー名とパスワードを入力するダイアログが表示されます。データベースにセキュリティが設定されているかどうかを調べるには、パブリック プロパティの *TPvDatabase.IsSecured* で確認することができます。セキュリティで保護されたテーブル、およびユーザー名とパスワードの入力要求も参照してください。

メモ： MicroKernel エンジン セキュリティはオーナー ネームを使用します（テーブル セキュリティを参照してください）。データベースで SQL セキュリティが有効になっている場合、オーナー ネーム セキュリティは無視されます。

PDAC と Embarcadero のコンポーネントの相違点

次の表では、Zen と Embarcadero のコンポーネントの機能の違いについてまとめています。

TPvTable	すべてのエイリアス管理機能で「名前付きデータベース」管理を行います。 TDBDataSet からではなく TPvDataSet から継承します。 TableLevel、UpdateObject、UnlockTable、OpenIndexFile、CloseIndexFile の各プロパティは既に削除されており、TableType が無視されます。 設計時と実行時のどちらででも、DSN、Zen 名前付きデータベース、あるいはデータベースへの完全修飾パスとして、Database Name プロパティを利用できます。
TPvDatabase	DriverName、Locale、TtraceFlags の各プロパティが既に削除されています。 ハンドルのタイプが DRM_DBID に変更されます。 *IsSQLBased* は常に False を返します。 TransIsolation の値は、"tiDirtyRead" と "tiReadCommitted"（デフォルト）の 2 つのみとなります。

EPvDBEngineError（EPvDBEngineError クラスを参照）	EDBEngineError の代替です。
TPvDBError（TPvDBError クラスを参照）	TDBError の代替です。
TPvSQLSession	すべてのエイリアス管理機能で DSN 管理を行います。 Handle プロパティのタイプが SQLHENV に変更されます。 AddStandardAlias メソッドではデフォルトの設定でエンジン DSN を追加します。 'UpdateDsnType:TDsnTypes' プロパティが追加されています。TDsnTypes = (dsnSystem, dsnUser)。UpdateDsnType の意味は次のとおりです。 • DeleteAlias 関数の場合は、DSN ユーザーが System または User DSN のどちらを削除するのか。 • AddAlias および GetAliasParameters 関数の場合は、どの DSN タイプにするのか。
TPvSQLDatabase	Handle プロパティのタイプが SQLHDBC に変更されています。 IsSQLBased プロパティは常に True になります。 Tables と StoredProcs プロパティには、それぞれ TsqlTablesInfoCollection、TsqlStoredProcInfoCollection というタイプがあります。
TPvQuery	TQuery からの継承です。 Handle プロパティのタイプが SQLHSTMT に変更されています。 DBHandle プロパティのタイプが SQLHDBC に変更されています。 StmtHandle プロパティが削除されています。 Text プロパティは常に SQL プロパティからのテキストを返します。 CheckOpen メソッドが削除されています。 インデックスとキーに関連するすべてのプロパティとメソッド（GetIndexInfo など）が削除されています。 'LoadBlobOnOpen' プロパティが True のときは、クエリ オープンですべての BLOB がメモリにキャッシュされます。LoadBlobOnOpen が False のときは、BLOB は必要時に読み込まれます。

TransIsolation プロパティの相違点

PDAC の **TransIsolation** プロパティの動作は、相当する Embarcadero の動きとは違います。

トランザクションの分離レベルでは、同じテーブルで動作するとき、トランザクションがほかで同時に動作しているトランザクションと対話する方法と、ほかのトランザクションによって実行されるオペレーションをトランザクションで監視する程度を決定します。

データベースのトランザクション分離レベルを指定するには、**TPvDatabase.TransIsolation** プロパティを使用します。

PDAC では *tiReadCommitted* モードのみをサポートします。これは、トランザクションが終了するまで、ファイルへの変更をほかのユーザーが見ることができないという意味です。これは、データベース エンジンでサポートされる唯一の設定です。

PDAC クラス、プロパティ、イベント、およびメソッド

このトピックでは、PDAC のすべてのクラス、プロパティ、イベント、およびメソッドの一覧を示します。Pascal 表記法で記載し、対応する Embarcadero のコンポーネントとの相違点も示しています。

- 例外クラス
- サポート クラス
- VCL との全般的な違い
- VCL と異なる特定のクラス
- Zen および Embarcadero のデータ型のマッピング
- Btrieve および Embarcadero のデータ型のマッピング

例外クラス

EpvDatabaseError クラス

このクラスは PDAC のすべての例外クラスの上位クラスです。これには例外を生成するクラスへの参照が含まれる `Owner` プロパティがあります。

EPvDBEngineError クラス

これは、クラスに関連するすべての DB エンジン エラーの抽象ベースのクラスです。`Errors` プロパティでは（TPvDBError クラスから派生するクラスの）エラーを一覧表示し、`ErrorCount` プロパティは `Errors` プロパティに含まれるエラーの総数を示します。

TPvDBError クラス

TPvDBError は、EPvDBEngineError 例外クラスのデータベース エンジン エラーを表すすべてのクラスに関する抽象ベースのクラスです。以下のプロパティがあります。

- `Message` プロパティは、エラー メッセージのテキストを指定します。
- `NativeError` プロパティは、エンジンから返されるステータス コード（Btrieve ステータス コード）を示します。

EpvDrmEngineError クラス

このクラスの例外は PDAC の Btrieve サブセットによって発生します。`ErrorCode` プロパティは DRM エラー コードです。EPvDrmEngineError の `Errors` プロパティの配列には TpvDrmError タイプのオブジェクトが含まれます。

TpvDrmError クラス

TPvDrmError クラスは DRM エラーを示します。`ErrorCode` には DRM エラー コードが含まれ、`NativeError` には Btrieve ステータス コードが含まれます。

EpvSqlEngineError クラス

このクラスの例外は PDAC の SQL（リレーショナル）サブセットによって発生します。`ErrorCode` プロパティは最後の ODBC 呼び出しの戻り値です。EPvSqlEngineError の `Errors` プロパティの配列には TPvSqlError タイプのオブジェクトが含まれます。

TpvSqlError クラス

TPvSqlError クラスは ODBC エラーを示します。

EpvDbAdminEngineError クラス

このクラスの例外は、ローカル サーバーおよびリモート サーバー上で、TPvSqlSession.AddAlias などのデータベース名や DSN 管理関数の実行中に発生します。追加のプロパティ ErrorType: TpvDbAdminEngineErrorTypes があります。ErrorType = dbmeDTI の場合、NativeError には DTI（Distributed Tuning Interface）エラー コードが含まれます。それ以外の場合、NativeError にはデータベース名および DSN 関連の関数のローカル エラー コードが含まれます（EPvSqlInstallerEngineError および EpvOwnSqlInstallerEngineError を参照）。

EpvSqlInstallerEngineError クラス

このクラスの例外は、ローカル サーバー上で、TPvSqlSession.AddAlias などの DSN 管理関数の実行中（TPvSqlSession.ServerName プロパティが空またはローカル サーバーの名前が設定されていた場合）に発生します。`Errors` には、SQLInstallerError() 関数で返される値が含まれます。

EpvOwnSqlInstallerEngineError クラス

DSN 管理関数では適切なエラーがない場合があるので、このクラスの例外を発生して新しいタイプのエラーを示すことができます。`NativeError` の可能な値は以下のとおりです。

- cPvOwnSqlInstallerEngineErrorDsnAlreadyExist － ユーザーは既に存在する DSN を作成しようとしました。
- cPvOwnSqlInstallerEngineErrorDsnNotFound － ユーザーは存在しない DSN を削除しようとしました。
- cPvOwnSqlInstallerEngineErrorInvalidOpenMode － prmOPEN_MODE（'OPEN_MODE'）パラメーターの値が無効です。詳細については、後述の prmOPEN_MODE の説明を参照してください。
- cPvOwnSqlInstallerEngineErrorClientDSNsAreNotSupported － クライアント DSN はリモート サーバー モード（TPvSqlSession.ServerName が空）でサポートされていません。

サポート クラス

サポート クラスは、高レベルのコンポーネントとカプセル化した BDE 固有の機能で必要となります。これらのクラスは、可能な限り少ない変更で PDAC に 複製されています。

TPvSessionList/TpvSqlSessionList クラス

TPvSessionList および TPvSqlSessionList クラスは、複数のセッションを提供するアプリケーション内のセッション コンポーネントを管理します。このクラスでは、含まれるオブジェクトのタイプのみ（TSession に対応する TPvSession または TPvSqlSession）が変更されています。

TPvBlobStream/TPvSQLBlobStream クラス

TPvBlobStream および TPvSQLBlobStream クラスは、BLOB（バイナリ ラージ オブジェクト）フィールドを示すフィールド オブジェクトをアプリケーションで読み書きできるようにするストリーム オブジェクトです。これらは TBlobStream VCL クラスと同様に機能します。

TParam/TParams クラス

TParam クラスはフィールド パラメータを表します。TParam のプロパティはフィールドの値を示すパラメーターの値を設定するのに使用します。TParams は TParam オブジェクトのリストです。PDAC の TParam と TParams ではそれぞれのインターフェイス セクションの変更はありません。これらは新しいファイルに移動するだけです。

TMasterDataLink クラス

メモ： Delphi/C++Builder 3 および 4 用のみです。

TMasterDataLink を使用すれば、データセットでマスター / 詳細関係を設定することができます。このインターフェイス セクションでの変更はありません。新しいファイルに移動するだけです。

VCL との全般的な違い

PDAC コンポーネントで公開するインターフェイスは、BDE で動作する適切な VCL コンポーネントとほぼ一致しています。VCL コンポーネントに関する詳細情報は、Delphi/C++Builder ヘルプ システム（del?vcl.hlp または bcbvcl?.hlp ファイル。「?」は 3、4、または 5 を示します）を参照してください。ただし、BDE のいくつかの機能は Zen に存在せず、また Zen のいくつかの機能は BDE に存在しないので、Zen ではこれらのインターフェイス（削除または追加プロパティ / メソッド / イベント）を修正しました。これらの変更されたインターフェイスのみを以下に列挙します。その他すべてのインターフェイスは同一です。

グローバル変数

BDE の Session および Sessions グローバル変数の代わりに、PDAC では独自のグローバル変数があります。Btrieve サブセット用に BtvSession: TPvSession と BtvSessions: TPvSessionList があり、リレーショナル サブセット用に PvSqlSession: TPvSqlSession と PvSqlSessions: TPvSqlSessionList があります。これらは、Session や Sessions 変数と同様に動作し、アプリケーションの開始時に自動的に作成され、終了時に自動的に破棄されます。

Btrieve サブセット

トランザクショナル サブセットの場合、PDAC では Zen の名前付きデータベースをエイリアスとして使用します。

SQL サブセット

リレーショナル サブセットの場合、PDAC ではデータ ソース名（DSN）をエイリアスとして使用します。

VCL と異なる特定のクラス

以下のセクションでは、Embarcadero VCL とは異なるクラスを個別に説明します。

- TPvSession と TPvSqlSession
  - TPvSession に関する注記
  - TPvSqlSession に関する注記

- TPvDatabase と TPvSqlDatabase
  - TPvDatabase に関する注記
  - TPvSqlDatabase に関する注記
- TPvTable、TPvQuery および TPvStoredProc
  - TPvTable に関する注記
  - TPvQuery および TPvStoredProc に関する注記
  - TPvQuery に関する注記
  - TPvStoredProc に関する注記
- TPvUpdateSQL
- TPvBatchMove

TPvSession と TPvSqlSession

- TraceFlags プロパティは削除されました。これは、Zen API 呼び出しのトレースは外部ユーティリティまたは DTI（Distributed Tuning Interface）を使って実行されるからです。
- Zen は以下のプロパティを使用しません（これらは文字列の格納としてのみ存在します)。
  - NetFileDir
  - PrivateDir
- Locale プロパティは削除されました。Zen にはこの BDE 固有の機能に類似する機能がないからです。
- Zen にはドライバーがありません。ドライバーに関するすべてのインターフェイス項目（AddDriver、DeleteDriver、GetAliasDriverName、GetDriverNames、GetDriverParams、および ModifyDriver メソッド）は削除されました。
- パスワード関連のインターフェイスは、文字列のみを格納および取得します（AddPassword、RemovePassword、および RemoveAllPasswords)。OnPassword イベントは、GetPassword メソッド以外では発生しません。
- GetConfigParams メソッドは何も行いません。
- 新たに公開されるプロパティ、ServerName、ServerAdminUser、ServerAdminPassword、および ServerAdminLoginPrompt が追加されました。これら

は、リモート サーバーへの接続方法を提供します。ServerAdminUser と ServerAdminPassword は DTI（Distributed Tuning Interface）のユーザー名とパスワードです。

ヒント： 詳細については、DTI のドキュメントを参照してください。

- ServerAdminLoginPrompt プロパティは TPv(Sql)Database.LoginPrompt の相似型ですが、設計時に ServerAdminLoginPrompt = False とし、サーバーからデータベース名のリストを取得する際に問題が発生した場合、ログイン試行が失敗した後にユーザー名とパスワードが要求される点が異なります。

ヒント： DTI は Pervasive.SQL 2000 以降でのみサポートされます。

- LocalSystem パブリック プロパティ。
  このプロパティに、リモート サーバーの場合は False、ローカルの場合は True を設定します。

TPvSession に関する注記

- すべてのエイリアス管理機能は PDAC の名前付きデータベース管理を実行します。TPvSession を使用して、ローカルおよびリモート データベース名の作成、またローカルおよびリモート データベース名用のパラメーターの取得を行うことができます。サーバー上のデータベースとの接続を確立する場合、ユーザーはサーバー上のデータベースへのパスを使ってデータベース名を作成する必要があります。開発者は最初に以下の値を使用することができます。
  - prmDDF_PATH ('DDF_PATH') ― データ辞書ファイルへのパス。
  - prmPATH ('PATH') ― データ ファイルへのパス。
  - prmBOUND ('BOUND') ― データベースをバインドするかどうか。デフォルトは False です。
  - prmINTEGRITY ('INTEGRITY') ― データベースに参照整合制約を設定するかどうか。デフォルトは True です。
  - prmCREATE_DDF ('CREATE_DDF') ― このパラメーターが True の場合、空のデータベースが作成されます。
- AddStandardAlias メソッドでは、INTEGRITY = True、BOUND = False という標準の設定でデータベース名を追加します。
- Handle プロパティのタイプは DRM_SESID に変更されました。

- 例：データベース名の作成

```
var MyList: TStringList;
begin
    MyList := TStringList.Create;
try
  with MyList do
  begin
      Add('DDF_PATH=D:¥MyDemoData');
      Add('PATH=D:¥MyDemodata');
      Add('BOUND=False');
      Add('INTEGRITY=False');
      Add('Create_DDF=False');
  end;
  PvSession1.AddAlias('TestAlias', MyList);
finally
        MyList.Free;
      end;
```

TPvSqlSession に関する注記

- すべてのエイリアス管理機能で DSN 管理を行います。ユーザーは最初に以下の値を使用することができます。
  - prmDB_NAME (DB) － データベースのデータベース名エンジン DSN のみ。
  - prmDSN_DESCRIPTION (DESCRIPTION) － DSN の説明。
  - prmIS_ENGINE_DSN ('IS_ENGINE_DSN') － 指定した DSN が、エンジン DSN（True）あるいはクライアント DSN（False）のどちらなのかを調べます。
  - prmIS_SYSTEM_DSN ('IS_SYSTEM_DSN') － 指定した DSN が、システム DSN（True）あるいはユーザー DSN（False）のどちらなのかを調べます。
  - prmOPEN_MODE (OPEN_MODE) － DSN のオープン モードを調べます。可能な値は、prmOPEN_MODE_normal（ノーマル）、prmOPEN_MODE_accelerated（アクセラレイティド）、prmOPEN_MODE_readonly（リード オンリー）、および prmOPEN_MODE_exclusive（エクスクルーシブ）です。エンジン DSN のみ。

    - prmSERVER_NAME ('SERVER_NAME') ー データが存在するサーバーのアドレスまたはホスト名、およびポート番号。クライアント DSN のみ。
    - prmTCP_PORT ('TCP_PORT') ー サーバーの TCP ポート。クライアント DSN のみ。
    - prmSERVER_DSN ('SERVER_DSN') ー サーバーのエンジン DSN の名前。クライアント DSN のみ。
    - prmTRANSPORT_HINT ('TRANSPORT_HINT') ー 転送プロトコルが含まれます。クライアント DSN のみ。可能な値は TCP です。
    - prmARRAY_FETCH_ON ('ARRAY_FETCH_ON') ー 配列のフェッチを有効 / 無効（True/False）にします。クライアント DSN のみ。
    - prmARRAY_BUFFER_SIZE ('ARRAY_BUFFER_SIZE') ー 配列バッファーのサイズ。1 KB から 64KB までの値を受け付けることができます。クライアント DSN のみ。
- Handle プロパティのタイプは SQLHENV に変更されました。
- リモート DSN 管理は、Pervasive.SQL 2000 SP2a またはそれ以降でのみサポートされます。リモート クライアントとユーザー DSN はサポートされません。
- AddStandardAlias メソッドでは、デフォルトの設定でエンジン DSN を追加します。2 番目のパラメーターはデータベース名です。
- UpdateDsnType: TDsnTypes プロパティが追加されています。TDsnTypes = (dsnSystem, dsnUser)。UpdateDsnType の意味は次のとおりです。
    - DeleteAlias 関数の場合は、DSN ユーザーが System または User DSN のどちらを削除するのか。
    - AddAlias および GetAliasParameters 関数の場合は、どの DSN タイプにするのか。
- 以下はシステム クライアント DSN を作成するサンプル プログラムです。

```
var MyStringList: TStringList;
begin
  MyStringList := TStringList.Create;
  try
    MyStringList.Clear();
    MyStringList.Add('IS_ENGINE_DSN=False');
    MyStringList.Add('IS_SYSTEM_DSN=True');
```

```
        MyStringList.Add('SERVER_NAME=ServerName');
        MyStringList.Add('SERVER_DSN=DEMO1');
              // サーバーの DSN
        PvSqlSession1.AddAlias('ATest', MyStringList);
      finally
        MyStringList.Free;
      end;
```

TPvDatabase と TPvSqlDatabase

- ODBCPacketSize プロパティは SQLUINTEGER 値で、ネットワーク パッケージのサイズをバイト単位で指定します。これは SQLSetConnectAttr() で、接続について管理する SQL_ATTR_PACKET_SIZE 属性を設定する際に使用します。

ヒント： SQLSetConnectAttr() の詳細については、『Microsoft ODBC Programmer's Reference』を参照してください。

- 以下のプロパティは削除されました。
  - DriverName
  - Locale
  - TraceFlags
- TransIsolation プロパティは 1 つの値（tiReadCommitted）のみを持ちます。これは、Zen ではその他の分離レベルをサポートしないからです。
- Tables プロパティが追加されました。このプロパティにはデータベース内のテーブルのリストが含まれます。
- StoredProcs プロパティが追加されました。このプロパティにはデータベース内のストアド プロシージャのリストが含まれます。
- LoginPromptOnlyIfNecessary が追加されました。以下の状況が考えられます。
  - LoginPrompt=True で、LoginPromptIfNecessary=True。セキュリティで保護されたデータベースの場合にのみ、（ログイン試行の失敗後）ログイン ダイアログが表示されます。これはデフォルトの動作です。
  - LoginPrompt=True で、LoginPromptIfNecessary=False。ログインを試みる前に必ずログイン ダイアログが表示されます。これは VCL の TDatabase の動作です。セ

キュリティで保護されたデータベースに最も適しています。ログインする最も速い方法を提供します。

- LoginPrompt=False で、LoginPromptIfNecessary=True/False（どちらでもかまいません）。ログイン ダイアログは表示されません。ユーザーが独自のログイン ダイアログを実装できます。

TPvDatabase に関する注記

- Handle プロパティは DRM_DBID に変更されました。
- IsSQLBased プロパティは常に False です。
- ローカル サーバー（つまり、TPvSession.ServerName プロパティの値が空であるか、ローカル サーバーの名前が指定されている場合）の Directory プロパティには、データベースのデータ辞書ファイルへのパスが入ります。リモート サーバーの場合、このプロパティは常に空です。どちらの場合も、これを設定しようとすると例外が発生します。作業しているサーバーがローカルかリモートかを知るには、TPvSession.LocalSystem プロパティで調べることができます。
- 以下のプロパティには TDRMTableCollection のタイプがあります。
  - Tables
  - StoredProcs
- IsSecured プロパティ（boolean）が追加されました。これは読み取り専用プロパティです。データベースにセキュリティが設定されている場合は、IsSecured = True です。この場合、TPvTable.Owner プロパティの MicroKernel エンジン オーナー ネームは無視されるので、ユーザーはデータベースにログインする権限を許可する必要があります。
- OEMConversion プロパティが追加されました。

  このプロパティは、データベースに格納されている文字は OEM（DOS）コード ページによってエンコードされているため、データベースを使用する前にこれらの文字を ANSI（Windows）コード ページに変換する必要があることを示します。データベースは OEM コード ページのままですが、文字データのすべての読み書きは PDAC で変換されます。

  この変換では Windows の OemToCharBuff および CharToOemBuff 関数で提供されるマッピングを使用します。すべての文字が往復変換できるわけではないことに注意することが重要です。OEM と ANSI コード ページの両方に存在する文字のみが更新で保持されます。大まかに言うと、ほとんどのアルファベット文字は保持されます

が、ボックス描画文字などその他の種類の文字は保持されません。正確に保持できない文字については、最も類似する文字が選択されます。たとえば、ボックス描画文字はプラス（+)、マイナス（-)、およびパイプ（｜）文字に置き換えられます。

現在のところ、ユーザー テーブルに保存されている文字のみが変換されます。テーブル、列、およびファイルの名前などの（DDF ファイルに保存されている）メタデータは変換されません。

TPvSqlDatabase に関する注記

- Handle プロパティのタイプは SQLHDBC に変更されました。
- IsSQLBased プロパティは常に True です。
- Directory プロパティは常に空です。これを設定しようとすると、例外が発生します。
- 以下のプロパティにはそれぞれ相当する TSqlTablesInfoCollection および TSqlStoredProcInfoCollection タイプがあります。
  - Tables
  - StoredProcs
- Exclusive プロパティは何も行いません。VCL との互換性を保つためだけに提供されます。
- AliasNameIsConnectionString プロパティが追加されました。このプロパティは DSN のない接続を行うために提供されます。AliasNameIsConnectionString=True の場合、AliasName（AliasName が空の場合は DatabaseName）が接続文字列になります。

TPvTable、TPvQuery および TPvStoredProc

- TDBDataSet のすべてのプロパティ、メソッド、イベントのタイプは TPvDataSet に変更されたか、または**派生**しました。
- TSession のすべてのプロパティ、メソッド、イベントのタイプは TPvAbsSession に変更されたか、または**派生**しました。
- TDatabase のすべてのプロパティ、メソッド、イベントのタイプは TPvAbsDatabase に変更されたか、または**派生**しました。
- ExpIndex プロパティは常に False です。
- 以下のプロパティは削除されました。

    - Locale
    - DBLocale
    - ObjectView
- TPvDatabase または TPvSqlDatabase タイプの Database プロパティが追加されました。
- ConstraintCallBack は削除されました。
- CheckOpen メソッドのパラメーター タイプは DRM_ERR に変更されました。

TPvTable に関する注記

- Handle プロパティのタイプは DRM_TABLEID に変更されました。
- TDBHandle プロパティのタイプは DRM_DBID に変更されました。
- Owner プロパティが追加されています。これはテーブルの Btrieve オーナー ネームを表します。
- BtrHandle のプロパティ：TBtrieveInfo が追加されています。これは DirectBtrCall 関数のヘルパー プロパティです。TBtrieveInfo には以下のフィールドがあります。
    - pKeyBuf － キー バッファーへのポインター。
    - KeyLen － キーの長さ。
    - KeyNum － キーの番号。
    - CurFilter － Extended オペレーション用の入力データ バッファー構造体へのポインター（『*Btrieve API Guide*』の GetNextExtended を参照）。また、このバッファーには現在のフィルターも格納されます。
    - CurFilterLen － CurFilter バッファーの長さ。CurFilterLen は、開発者が CurFilter データをバッファーからメモリ内の別の場所にコピーするときに利用できます。
- DirectBtrCall(Op: Smallint function; pDataBuf: Pointer; var DataLen: Word; pKeyBuf: Pointer; KeyLen: Byte; KeyNum: Shortint): integer が追加されました。開発者は TpvTable のポジション ブロックを使用して直接 Btrieve を呼び出すことができます。以下のパラメーターがあります。
    - Op － Btrieve オペレーション。『*Btrieve API Guide*』を参照してください。
    - pDataBuf － BTRCALL のデータ バッファー パラメーターと同一
    - DataLen － BTRCALL のデータ バッファー長パラメーターと同一

- pKeyBuf － BTRCALL のキー バッファー パラメーターと同一
- KeyLen － キー バッファーの長さ
- KeyNum － BTRCALL のキー番号パラメーターと同一

DirectBtrCall は Btrieve ステータス コードを返します。pKeyBuf、KeyLen、および KeyNum パラメーターは PvTable.BtrHandle プロパティから取得する必要があります。

以下に、Btrieve の直接呼び出しによって現在のレコードをロックおよびロック解除する方法を示す小さなサンプルを提供します。

```
procedure TForm1.Lock(Sender: TObject);
var     b: TBookmark;
DataLen: word;
Res: integer;
begin
b := PvTable1.GetBookmark();
try
     DataLen := 4;
      Res := PvTable1.DirectBtrCall(B_GET_DIRECT + 300, b, DataLen, PvTable1.BtrHandle.pKeyBuf,
    PvTable1.BtrHandle.KeyLen, PvTable1.BtrHandle.KeyNum);
    finally
    PvTable1.FreeBookmark(b);
    end;
    end

procedure TForm1.Unlock(Sender: TObject);
var   Res: integer;
vr: Word;
    begin
      vr := 0;
      Res := PvTable1.DirectBtrCall(B_UNLOCK, @vr, vr, @vr, vr, -2);
    end
```

- IndexFiles プロパティは削除されています。
- TableType プロパティは削除されています。
- TableLevel プロパティは無視されます。
- UpdateObject プロパティは削除されています。
- 以下のメソッドが削除されました。
  - CloseIndexFile
  - OpenIndexFile
  - LockTable
  - UnlockTable

- PvCreateTable(PvFieldDefs: TPvFieldDefs) メソッドが追加されています。このメソッドを使えば、開発者はテーブル作成処理をある程度調整することができます。詳細については、適切なセクションを参照してください。
- SetOwnerOnTable(AOwner: string; AccessMode: integer) メソッドが追加されています。開発者はテーブルの Btrieve オーナー ネームを設定することができます。AccessMode には以下の設定が可能です。
  - B_ACCESS_RWOWNER ― すべてのアクセス モードでオーナー ネームが必要（データ暗号化なし）。
  - B_ACCESS_WOWNER ― 読み取り専用アクセスにはオーナー ネームは必要なし（データ暗号化なし）。
  - B_ACCESS_RWOWNERENCRYPT ― すべてのアクセス モードでオーナー ネームが必要（データ暗号化あり）。
  - B_ACCESS_WOWNERENCRYPT ― 読み取り専用アクセスにはオーナー ネームは必要なし（データ暗号化あり）。

TPvQuery および TPvStoredProc に関する注記

- Handle プロパティのタイプは SQLHSTMT に変更されました。
- DBHandle プロパティのタイプは SQLHDBC に変更されました。
- StmtHandle は削除されました。
- Text プロパティは常に SQL プロパティからのテキストを返します。
- CheckOpen メソッドは削除されました。
- インデックスとキーに関連するすべてのプロパティとメソッド（GetIndexInfo など）は削除されました。

TPvQuery に関する注記

以下のプロパティが追加されました。

- LoadBlobOnOpen

  LoadBlobOnOpen が True のときは、クエリ オープンですべての BLOB がメモリにキャッシュされます。LoadBlobOnOpen が False のときは、BLOB は必要に応じて読み取られます。
- PassThrough

このプロパティを True に設定すると、PDAC では SQL テキストを事前解析（通常、パラメーターをバインドするために行う）しないで直接エンジンに渡します。これは、パラメーターを使ってストアド プロシージャを作成する場合などで必要となります。このプロパティは次のように使用します。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  PvQuery1.SQL.Clear;

  PvQuery1.SQL.Add('CREATE PROCEDURE TestPr(IN :A INTEGER) AS');
  PvQuery1.SQL.Add('BEGIN');
  PvQuery1.SQL.Add('PRINT :A;');
  PvQuery1.SQL.Add('END');
  PvQuery1.PassThrough := True;
  PvQuery1.ExecSQL;
  PvQuery1.PassThrough := False;
end;
```

PassThrough プロパティは IDE 内で設計時にも使用できます。

TPvStoredProc に関する注記

- Overload プロパティは削除されました。

TPvUpdateSQL

- TQuery プロパティのタイプ（すべてのプロパティ）が TPvQuery で変更されています。

TPvBatchMove

- Transliterate プロパティは削除されました。
- Destination プロパティのタイプは TPvTable に変更されました。

Zen および Embarcadero のデータ型

以下のトピックでは、データ型のマッピングについて詳しく説明します。

Zen および Embarcadero のデータ型のマッピング

次の表は、Zen の列のデータ型から Delphi のデータ型へのマッピングを示しています。Zen データベースに格納されている左側のデータ型は、PDAC コンポーネントにより、右側に記載されているデータ型で表されます。

Zen データ型	Delphi データ型
BigInt	ftBCD
Binary	ftBytes
Bit	ftBoolean
Char	ftString
Currency	ftCurrency
Date	ftDate
Decimal	ftBCD
Double	ftFloat
Float	ftFloat
Identity	ftAutoInc
Integer	ftInteger
Longvarbinary	ftBlob
Longvarchar	ftMemo
Numeric	ftBCD
Real	ftFloat
Smallidentity	ftAutoInc
Smallint	ftSmallInteger
Time	ftTime
TimeStamp	ftDateTime
Tinyint	ftSmallInteger

Zen データ型	Delphi データ型
Ubigint	ftBCD
Uint	ftInteger
Usmallint	ftWord
Utinyint	ftWord
Varbinary	ftVarBytes
Varchar	ftString

メモ： 現在のリリースでは、PDAC は BIGIDENTITY データ型をサポートしていません。

次の表は、Delphi データ型から Zen データ型へのマッピングを示しています。PDAC を使って新しいデータベース テーブルが作成されると、左側の列に示されるフィールドのデータ型として定義された列が、Zen によって、右側に示されるデータ型で保存されます。

Delphi データ型	Zen データ型
ftAutoInc	Identity
ftBCD	Numeric
ftBlob,	Longvarbinary
ftBoolean	Bit
ftBytes	Binary
ftCurrency	Currency
ftDate	Date
ftDateTime	DateTime
ftFixedChar	Char
ftFloat	Double
ftFmtMemo	LongVarChar
ftGraphic	Blob
ftInteger	Integer
ftLargeInt	BigInt

Delphi データ型	Zen データ型
ftMemo	Longvarchar
ftSmallInteger	Smallint
ftString	Varchar
ftTime	Time
ftTypedBinary	Binary
ftVarBytes	VarChar
ftWord	Smallint

Btrieve および Embarcadero のデータ型のマッピング

次の表は、Btrieve から VCL へのデータ型のマッピングを示しています。

メモ： バイナリ フラグは X$Fields.Xe$Flags のフラグを指します。

Btrieve データ型	バイナリ フラグ	長さ（バイト単位）	VCL データ型
AUTOINCREMENT (15)			ftAutoInc
BFLOAT (9)			ftFloat
BIT (16)			ftBoolean
BLOB (21)	+		ftBlob
BLOB (21)	-		ftMemo
CURRENCY (19)			ftBCD
DATE (3)			ftDate
DECIMAL (5)			ftBCD
FLOAT (2)			ftFloat
INTEGER (1)		1	ftSmallint
INTEGER (1)		2	ftSmallint
INTEGER (1)		4	ftInteger
INTEGER (1)		8	ftBCD
LOGICAL (7)		1	ftBoolean

Btrieve データ型	バイナリ フラグ	長さ（バイト単位）	VCL データ型
LOGICAL (7)		2	ftSmallint
LSTRING (10)	+		ftVarBytes
LSTRING (10)	-		ftString
LVAR (13)	+		ftBCD
LVAR (13)	-		ftMemo
MONEY (6)			ftBCD
NOTE (12)	+		ftBlob
NOTE (12)	-		ftMemo
NUMERIC (8)			ftBCD
NUMERICSA (18)			ftBCD
NUMERICSTS (17)			ftBCD
STRING (0)	+		ftBytes
STRING (0)	-		ftString
TIME (4)			ftTime
TIMESTAMP (20)			ftDateTime
UNSIGNED BINARY (14)		1	ftWord
UNSIGNED BINARY (14)		2	ftWord
UNSIGNED BINARY (14)		4	ftInteger
UNSIGNED BINARY (14)		8	ftBCD
ZSTRING (11)			ftString

次の表は、VCL から Btrieve へのデータ型のマッピングを示しています。

VCL データ型	Btrieve 型	バイナリ フラグ	長さ（バイト単位）
ftAutoInc	AUTOINCREMENT (15)		4
ftBCD	NUMERIC (8)		
ftBlob	BLOB (21)	+	FieldDefs[].Size

VCL データ型	Btrieve 型	バイナリ フラグ	長さ（バイト単位）
ftBoolean	LOGICAL (7)		1
ftBytes	STRING (0)	+	FieldDefs[].Size
ftCurrency	CURRENCY (19)		
ftDate	DATE (3)		
ftDateTime	TIMESTAMP (20)		
ftFixedChar	STRING (0)	-	FieldDefs[].Size
ftFloat	FLOAT (2)		8
ftFmtMemo	BLOB (21)	-	FieldDefs[].Size
ftGraphic	BLOB (21)	+	FieldDefs[].Size
ftInteger	INTEGER (1)		4
ftLargeint	INTEGER (1)		8
ftMemo	BLOB (21)	-	FieldDefs[].Size
ftSmallint	INTEGER (1)		2
ftString	ZSTRING (11)	-	FieldDefs[].Size
ftTime	TIME (4)		
ftTypedBynary	STRING (0)	+	FieldDefs[].Size
ftVarBytes	LSTRING (10)	+	FieldDefs[].Size
ftWord	UNSIGNED BINARY (14)		2

メモ： バイナリ フラグは X$Fields.Xe$Flags のフラグを指します。

フィールド タイプの追加情報

Zen 固有のテーブル作成メソッド（TPvTable.PvCreateTable）があります。これを使えば、フィールド タイプに関する追加パラメーターを調整することができます。以下の定義があります。

```
Procedure PvCreateTable(PvFieldDefs: TPvFieldDefs)
```

PvFieldDefs でいくつかのパラメーターを調整することができます。

```
TPvFieldDef = class(TCollectionItem)
public
  FieldNum: integer;
  BtrType: integer;
  DrmType: word;
  ColumnSize: integer;
  DefaultValue: string;
  IsColumnCaseInsensitive: boolean;
  ACS_FileName: string;
  ACS_Name: string;
  ACS_ID: string;
end;
```

上記の要素は次のような意味があります。

- FieldNum － 主 FieldDefs 内のフィールド番号。
- BtrType － Btrieve 型。
- DrmType － データ レコード マネージャー型。開発者はこのフィールドを使う必要はありません。TPvBatchMove で使用する場合のみ追加されます。
- ColumnSize － 列のサイズ（バイト単位）。
- DefaultValue － 列のデフォルト値（文字列）。
- IsColumnCaseInsensitive － フィールドのインデックスで大文字と小文字を区別する場合、True を設定します。
- ACS_FileName － ACS データを含んでいるファイル名を、.alt 拡張子を付けないで設定します。
- ACS_Name － ACS データの名前を設定します。
- ACS_ID － ACS データの ID を設定します。

特定のフィールドを設定しない場合は、0（DrmType、ColumnSize の場合）、-1（BtrType の場合）、False（IsColumnCaseInsensitive の場合）、または "（すべての文字列フィールド）を設定することができます。この場合、デフォルト値を使用します。FieldNum フィールドは必須です。

IsColumnCaseInsensitive、ACS_FileName、ACS_Name、および ACS_ID フィールドは互いに排他的になっています。つまり、これらのうち 1 つのみ設定することができます。

Btrieve 動作と一致させるために、インデックス オプションの ixCaseInsensitive は無視されます。

次に PvCreateTable メソッドの使用例を示します。

```
with PvTable1.FieldDefs do
begin
  Clear;
  Add('F_AutoInc', ftAutoInc, 0, true);
  Add('F_Bytes', ftBytes, 10, False);
end;

PvFieldDefs := TPvFieldDefs.Create(TPvFieldDef);
try
  PvFieldDef := PvFieldDefs.Add();
  PvFieldDef.FieldNum := 1; // F_Bytes
  PvFieldDef.BtrType := 10;
  PvFieldDef.DrmType := DRM_coltypVarText;
  PvFieldDef.ColumnSize := 20;
  PvFieldDef.IsColumnCaseInsensitive := true;

  PvTable1.PvCreateTable(PvFieldDefs);
finally
  PvFieldDefs.Free();
end;
```