



.Net Data Provider

リリース **4.1.3.1**

EDB Postgres™ Advanced Server .NET Data Provider Guide

2020 年 06 月 30 日

目次

第 1 章	新機能	3
第 2 章	要件の概要	5
2.1	サポートされているサーバーのバージョン	5
2.2	サポートされているプラットフォーム	5
第 3 章	AdvancedServer.NET コネクタ-概要	7
3.1	.NET クラス階層	7
第 4 章	.NET コネクタのインストールと設定	9
4.1	.NET コネクタのインストール	9
4.2	.NET コネクタの構成	10
4.2.1	ライブラリファイルの参照	12
4.2.2	.NETFramework のセットアップ	14
4.2.2.1	.NETFramework4.0	14
4.2.2.2	.NETFramework4.5	15
4.2.2.3	.NETFramework4.5.1	16
4.2.2.4	.NETFramework4.6.1	17
4.2.2.5	.NETStandard2.0	18
4.2.2.6	.NET 標準 2.1	19
4.2.2.7	.NETCore3.0	20
4.2.3	エンティティフレームワーク 5/6	21
4.2.4	EnterpriseDBVSIXforVisualStudio2015/2017/2019	23
4.2.4.1	VisualStudio2015/2017/2019 のインストールと構成	23
4.2.4.2	モデルファーストとデータベースファーストの使用法	25
第 5 章	セキュリティと暗号化	27
5.1	スクラムの互換性	27
第 6 章	.NET コネクタの使用	29
第 7 章	データベース接続を開く	31
7.1	接続文字列パラメータ	31
7.2	例-ASP.NET を使用してデータベース接続を開く	34

7.3	例-コンソールアプリケーションからデータベース接続を開く	35
7.4	例-Windows フォームアプリケーションからデータベース接続を開く	37
第 8 章	データベースレコードの取得	39
8.1	単一のデータベースレコードの取得	41
第 9 章	パラメータ化されたクエリ	43
第 10 章	データベースにレコードを挿入	45
第 11 章	データベースのレコードを削除する	47
第 12 章	.NET アプリケーションでの SPL ストアドプロシージャの使用	49
12.1	例-パラメータなしのストアドプロシージャの実行	50
12.2	例-IN パラメータを使用したストアドプロシージャの実行	52
12.3	例-IN、OUT、および INOUT パラメータを使用したストアドプロシージャの実行	56
第 13 章	アドバンスドキューイングの使用	61
13.1	メッセージのエンキューまたはデキュー	62
13.1.1	サーバー側の設定	62
13.1.2	クライアント側の例	63
13.2	EDBAQ クラス	70
第 14 章	.NET アプリケーションでの参照カーソルの使用	75
第 15 章	プラグインの使用	79
15.1	GeoJSON	80
15.2	Json.NET	81
15.3	LegacyPostGIS	82
15.4	NetTopologySuite	83
15.5	NodaTime	84
15.6	RawPostGIS	85
第 16 章	API リファレンス	87
第 17 章	結論	89
索引		91

EDBPostgres™AdvancedServer (AdvancedServer) と共に配布される EDBPostgres™AdvancedServer.NET コネクタは、.NET クライアントアプリケーションと AdvancedServer データベースサーバー間の接続を提供します。このガイドでは、AdvancedServer.NET コネクタの機能を示すインストール手順、使用手順、および例を示します。

- AdvancedServer のインスタンスに接続する方法。
- AdvancedServer データベースから情報を取得する方法。
- AdvancedServer データベースに格納されている情報を更新する方法。

このドキュメントは、C #と.NET の両方に関する確かな実用知識があることを前提としています。AdvancedServer.NET コネクタ機能は、Npgsql オープンソースプロジェクトのコア機能に基づいて構築されています。

Npgsql ユーザーズマニュアルは`オンラインで入手可能です<<http://www.npgsql.org/doc/index.html>>`。

第 1 章

新機能

AdvancedServer.NET コネクタ 4.1.3.1 を作成するために、次の機能が追加されました。

- .NET コネクタは、アップストリームコミュニティドライババージョン 4.1.3 と統合されました。マージの更新の詳細については、<https://www.nuget.org/packages/Npgsql/4.1.3> にアクセスしてください。
- コネクタは .NET Framework 4.6.1、.NET Core 3.0、.NET Standard 2.1 をサポートします。

第 2 章

要件の概要

次のセクションでは、AdvancedServer.NET コネクタでサポートされているプラットフォームについて詳しく説明します。

2.1 サポートされているサーバーのバージョン

AdvancedServer.NET コネクタは、AdvancedServer バージョン 9.5 以降で認定されています。

2.2 サポートされているプラットフォーム

AdvancedServer.NET コネクタのグラフィカルインストーラは、次の Windows プラットフォームでサポートされています。

64 ビット Windows:

- Windows サーバー 2019
- Windows サーバー 2016
- Windows サーバー 2012R2
- ウィンドウズ 10
- ウィンドウズ 8.1

32 ビット Windows:

- ウィンドウズ 10
- ウィンドウズ 8.1

第 3 章

AdvancedServer.NET コネクタ-概要

AdvancedServer.NET コネクタは、クライアントアプリケーションが AdvancedServer ホストに格納されているデータベースに接続できるようにする .NET データプロバイダーです。 .NET コネクタはデータに直接アクセスし、クライアントアプリケーションの最適なパフォーマンス、幅広い機能、および AdvancedServer 機能へのアクセスを可能にします。

.NET コネクタは、次のフレームワークをサポートしています。

- .NET Framework versions 4.0, 4.5.1 and 4.6.1
- .NET Standard 2.0 and 2.1
- .NET Core 3.0
- Entity Framework 5/6
- EnterpriseDB VSIX

3.1 .NET クラス階層

.NET クラス階層には、AdvancedServer データベースへの接続を制御し、サーバーに格納されているデータを操作するオブジェクトを作成するために使用できるいくつかのクラスが含まれています。以下は、最も一般的に使用されるオブジェクトクラスのほんの一部です。

EDBConnection

EDBConnection クラスは AdvancedServer への接続を表します。 EDBConnection オブジェクトには、2 クライアントが AdvancedServer データベースへの接続方法を指示する ConnectionString が含まれています。

EDBCommand

EDBCommand オブジェクトには、クライアントが AdvancedServer に対して実行する SQL コマンドが含ま

れています。 `EDBCommand` オブジェクトを実行する前に、それを `EDBConnection` オブジェクトにリンクする必要があります。

`EDBDataReader`

An `EDBDataReader` object provides a way to read an Advanced Server result set. You can use an `EDBDataReader` object to step through one row at a time, forward-only.

`EDBDataAdapter`

An `EDBDataAdapter` object links a result set to the Advanced Server database. You can modify values and use the `EDBDataAdapter` class to update the data stored in an Advanced Server database.

第 4 章

.NET コネクタのインストールと設定

この章では、AdvancedServer.NET コネクタをインストールして構成する方法について説明します。

4.1 .NET コネクタのインストール

EnterpriseDB.NET コネクタインストーラー（EnterpriseDBWeb サイトから入手可能）を使用できます。<<https://www.enterprisedb.com/advanced-downloads>>_）.NET コネクタをシステムに追加します。インストーラーをダウンロードしたら、インストーラーのアイコンを右クリックし、コンテキストメニューから [0] を選択します。プロンプトが表示されたら、インストール言語を選択し、OK をクリックして ``2`` ウィンドウに進みます。

Next をクリックして続行します。

Installation Directory ダイアログを使用してコネクタをインストールするディレクトリを指定し、Next をクリックして続行します。

Ready to Install ダイアログの Next をクリックしてインストールを開始します。ポップアップダイアログは、インストールウィザードの進行状況を確認します。

セットアップが完了したことがウィザードから通知されたら、Finish ボタンをクリックしてダイアログを終了します。

StackBuilderPlus を使用して、既存の AdvancedServer インストールでコネクタを追加または更新することもできます。StackBuilderPlus を開くには、Windows の Apps メニューから StackBuilderPlus を選択します。

StackBuilderPlus が開いたら、画面の指示に従います。ツリーコントロールの Database Drivers ノードから EnterpriseDB.Net Connector オプションを選択します。

画面上のウィザードの指示に従って、EnterpriseDB コネクタのインストールを追加または更新します。

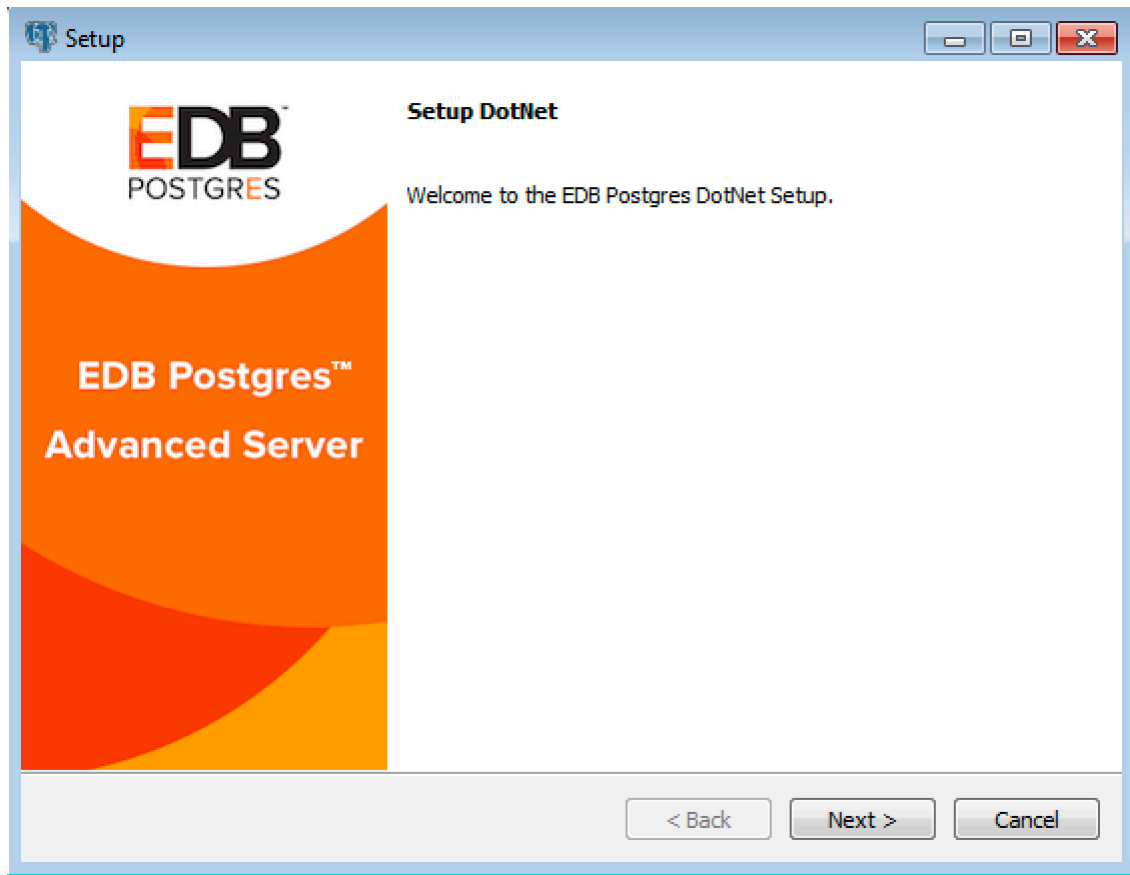


図 1 .NET コネクタインストールウィザード

4.2 .NET コネクタの構成

.NET コネクタの構成については、以下の環境固有のセクションを参照してください。

- ライブラリファイルの参照。:ref:`一般的な構成情報<referencing\_the\_library\_files>`すべてのコンポーネントに適用できます。
- **。NETFramework4.0。** *.NETFramework4.0* で使用するための構成手順。
- **。NETFramework4.5。** *.NETFramework4.5* で使用するための構成手順。
- **。NETFramework4.5.1。** *.NETFramework4.5.1* で使用するための構成手順。
- **。NETFramework4.6.1** *.NETFramework4.6.1* で使用するための構成手順。
- **。NETStandard2.0。** *.NETStandard2.0* で使用するための構成手順。
- **。NETStandard2.1。** *.NETStandard2.1* で使用するための構成手順。
- **。NETCore3.0** *.NETCore3.0* で使用するための構成手順。
- ****EntityFramework5/6。** *EntityFramework* で使用するための構成手順。

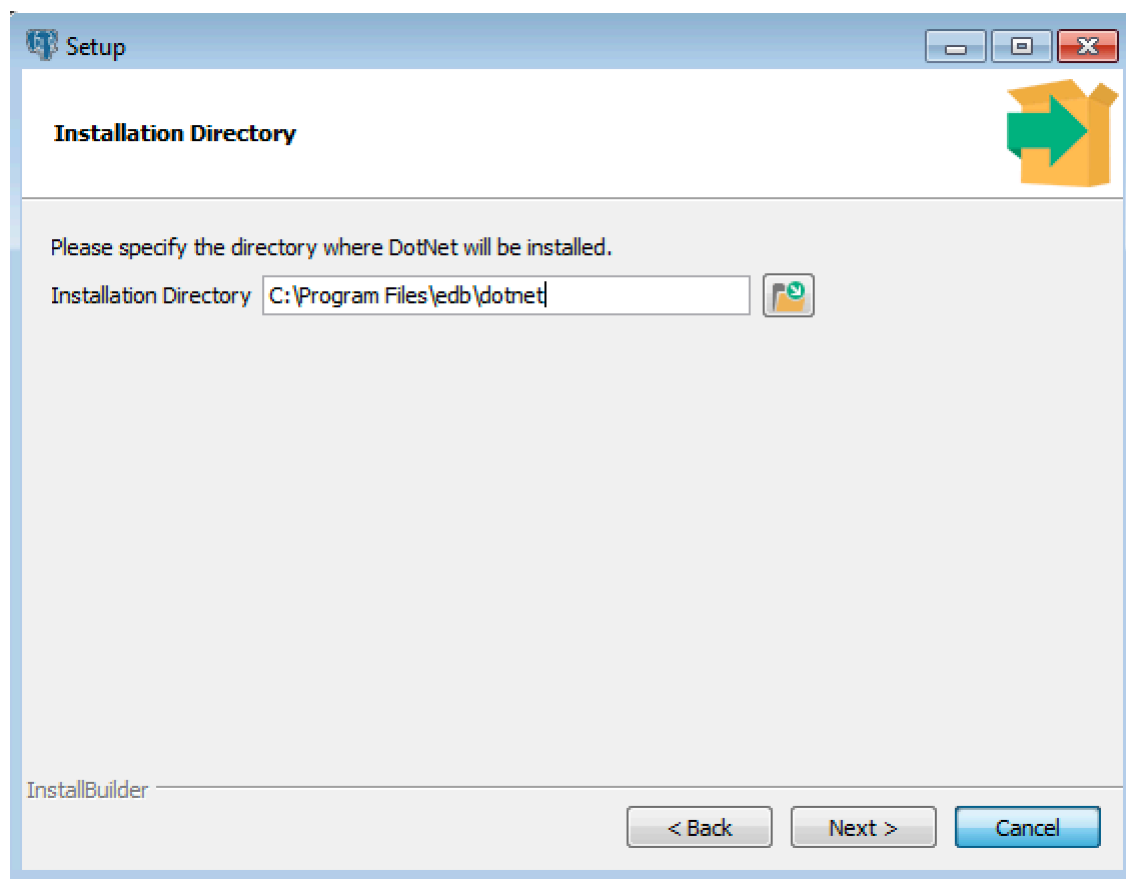


図 2 インストールダイアログ

- ****EnterpriseDBVSIX。** : *EnterpriseDBVSIX* で使用するための構成手順。

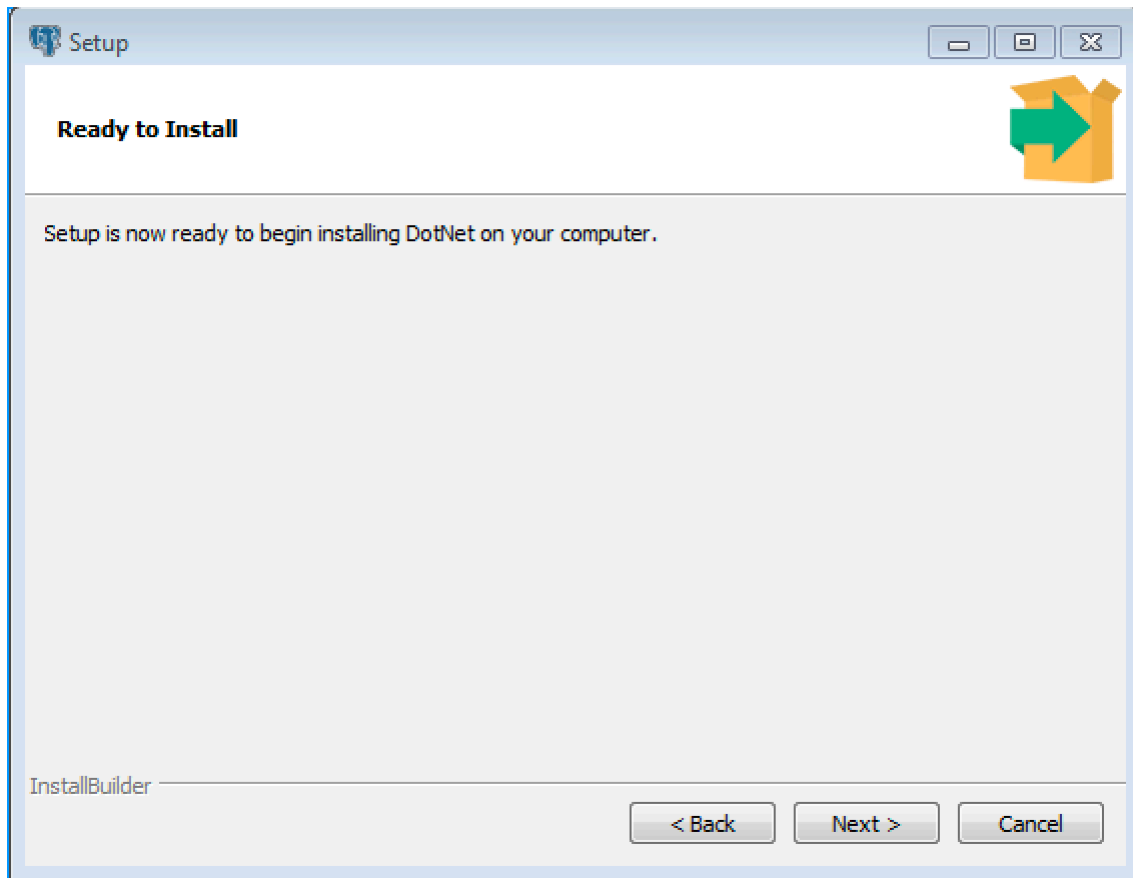


図 3 インストールの準備ダイアログ

4.2.1 ライブラリファイルの参照

Microsoft Visual Studio でライブラリファイルを参照するには:

1. Solution Explorer でプロジェクトを選択します。
2. Project メニューから Add Reference を選択します。
3. Add Reference ダイアログボックスが開いたら、適切なライブラリファイルを参照して選択します。

オプションで、ライブラリファイルを指定した場所にコピーできます。

AdvancedServer.NET クラスを使用する前に、名前空間をプログラムにインポートする必要があります。名前空間をインポートすると、コンパイラは名前空間内で使用可能なクラスを認識します。名前空間は次のとおりです。

```
EnterpriseDB.EDBClient
```

EntityFramework6 を使用している場合は、次の追加の名前空間が必要です。

```
EntityFramework6.EnterpriseDB.EDBClient
```

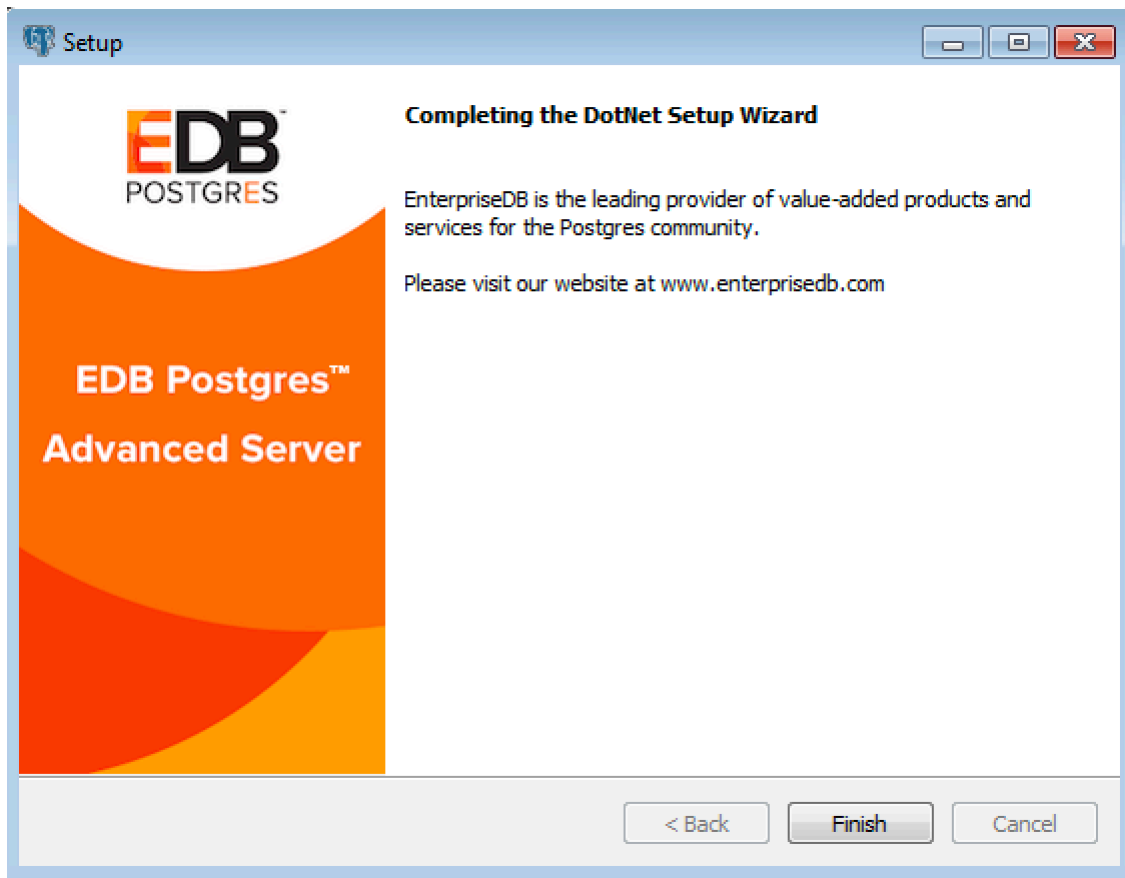


図 4 インストールが完了しました

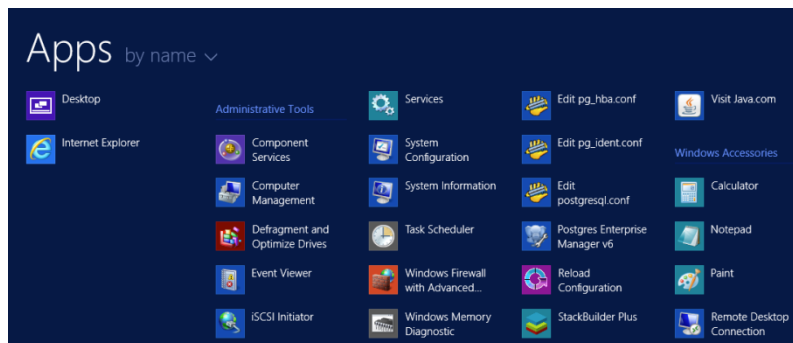


図 5 StackBuilderPlus の起動

名前空間を含めるために使用する方法は、作成しているアプリケーションのタイプによって異なります。たとえば、次のコマンドは名前空間を ASP.NET ページにインポートします:

```
<% import namespace=/"EnterpriseDB.EDBClient/" %>
```

名前空間を C # アプリケーションにインポートするには、次のように記述します。

```
using EnterpriseDB.EDBClient;
```

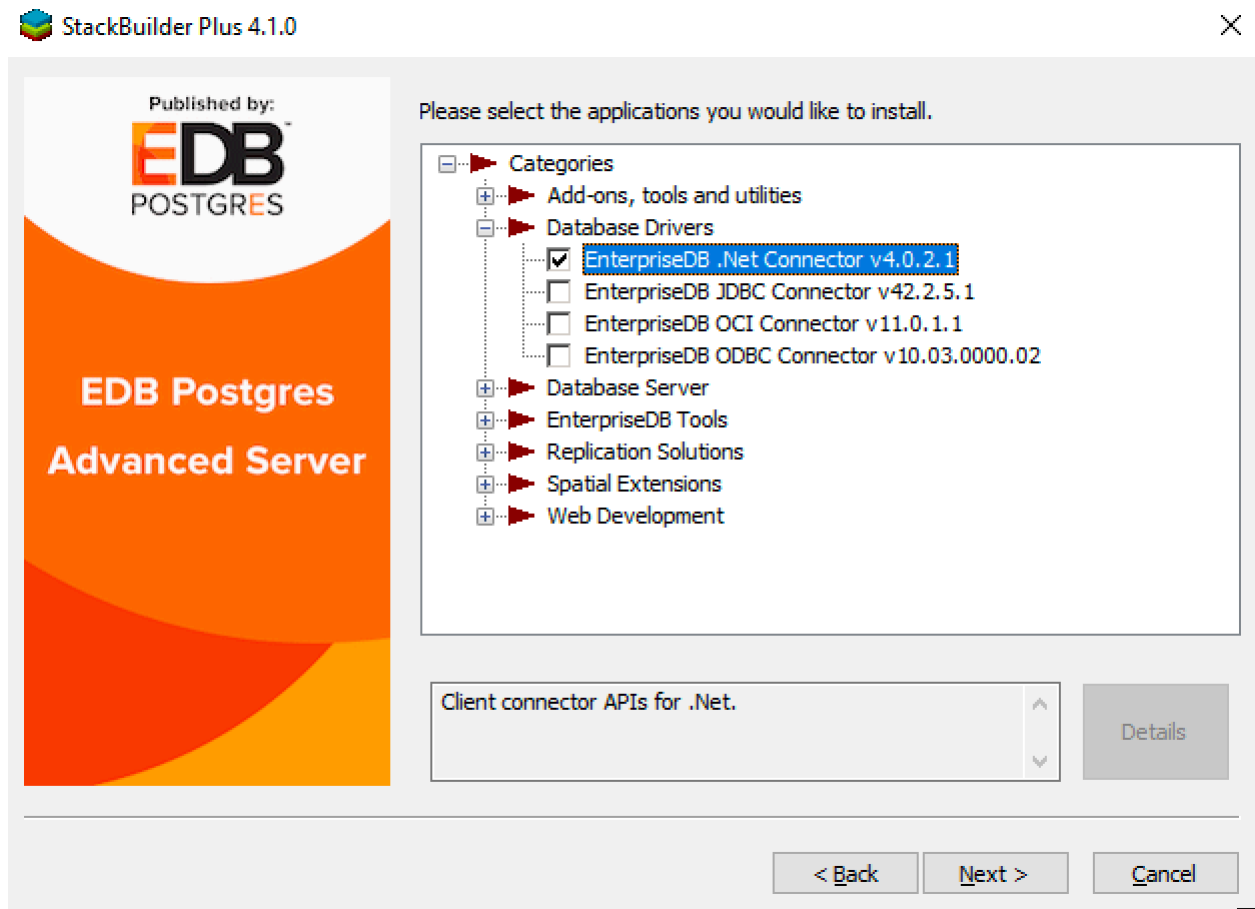


図 6 コネクターインストーラーの選択

4.2.2 .NETFramework のセットアップ

以下のセクションでは、さまざまな.NET バージョンのセットアップについて説明します。

4.2.2.1 .NETFramework4.0

.NETFramework バージョン 4.0 を使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/net40/
```

次の依存関係をプロジェクトに追加する必要があります。

```
EDBDataProvider.2.0.2.dll
```

```
Mono.Security.dll
```

使用するアプリケーションのタイプによっては、ネームスペースをソースコードにインポートする必要がある場合があります。 :ref:`ライブラリファイルの参照`を参照してください<referencing_the_library_files>`これとライブラリファイルの参照に関するその他の情報。

4.2.2.2 .NETFramework4.5

.NETFramework バージョン 4.5 を使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/net45/
```

次の依存関係をプロジェクトに追加する必要があります。

```
EnterpriseDB.EDBClient.dll
```

```
System.Threading.Tasks.Extensions.dll
```

```
System.Runtime.CompilerServices.Unsafe.dll
```

```
System.ValueTuple.dll
```

```
System.Memory.dll
```

使用するアプリケーションのタイプによっては、ネームスペースをソースコードにインポートする必要がある場合があります。 `:ref:` ライブラリファイルの参照を参照してください<referencing_the_library_files>`これとライブラリファイルの参照に関するその他の情報。

4.2.2.3 .NETFramework4.5.1

.NETFramework バージョン 4.5.1 を使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/net451/
```

次の依存関係をプロジェクトに追加する必要があります。

```
EnterpriseDB.EDBClient.dll
```

```
System.Threading.Tasks.Extensions.dll
```

```
System.Runtime.CompilerServices.Unsafe.dll
```

```
System.ValueTuple.dll
```

```
System.Memory.dll
```

使用するアプリケーションのタイプによっては、ネームスペースをソースコードにインポートする必要がある場合があります。 `:.ref:` ライブラリファイルの参照を参照してください<referencing_the_library_files>`これとライブラリファイルの参照に関するその他の情報。

4.2.2.4 .NETFramework4.6.1

.NETFramework バージョン 4.6.1 を使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/net461/
```

次の依存関係をプロジェクトに追加する必要があります。

```
EnterpriseDB.EDBClient.dll
```

```
Microsoft.Bcl.AsyncInterfaces.dll
```

```
System.Memory.dll
```

```
System.Numerics.Vectors.dll
```

```
System.Runtime.CompilerServices.Unsafe.dll
```

```
System.Runtime.dll
```

```
System.Text.Json.dll
```

```
System.Threading.Tasks.Extensions.dll
```

```
System.ValueTuple.dll
```

使用するアプリケーションのタイプによっては、ネームスペースをソースコードにインポートする必要がある場合があります。 :ref: ライブラリファイルの参照を参照してください<referencing_the_library_files>`これとライブラリファイルの参照に関するその他の情報。

4.2.2.5 .NETStandard2.0

.NETStandardFramework2.0 の場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/netstandard2.0/
```

次の依存関係をプロジェクトに追加する必要があります。

```
EnterpriseDB.EDBClient.dll  
  
System.Threading.Tasks.Extensions.dll  
  
System.Runtime.CompilerServices.Unsafe.dll  
  
System.ValueTuple.dll
```

注釈: ターゲットフレームワークが .NETCore2.0 の場合は、プロジェクトに次のファイルを含めます。

```
System.Threading.Tasks.Extensions.dll
```

使用するアプリケーションのタイプによっては、ネームスペースをソースコードにインポートする必要がある場合があります。 :ref: ライブラリファイルの参照を参照してください<referencing_the_library_files>`これとライブラリファイルの参照に関するその他の情報。

4.2.2.6 .NET 標準 2.1

.NETStandardFramework2.1 の場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/netstandard2.1/
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.dll
```

```
System.Memory.dll
```

```
System.Runtime.CompilerServices.Unsafe.dll
```

```
System.Text.Json.dll
```

```
System.Threading.Tasks.Extensions.dll
```

```
System.ValueTuple.dll
```

使用するアプリケーションのタイプによっては、ネームスペースをソースコードにインポートする必要がある場合があります。 :ref:`ライブラリファイルの参照を参照してください<referencing\_the\_library\_files>`これとライブラリファイルの参照に関するその他の情報。

4.2.2.7 .NETCore3.0

.NETCore3.0 を使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/netcoreapp3.0/
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.dll
```

```
System.Threading.Tasks.Extensions.dll
```

```
System.Runtime.CompilerServices.Unsafe.dll
```

```
System.ValueTuple.dll
```

```
System.Memory.dll
```

使用するアプリケーションのタイプによっては、ネームスペースをソースコードにインポートする必要がある場合があります。 `:ref:` ライブラリファイルの参照を参照してください<referencing_the_library_files>`これとライブラリファイルの参照に関するその他の情報。

4.2.3 エンティティフレームワーク 5/6

EntityFramework で使用するために .NET コネクタを設定するには、データプロバイダーのインストールパスは次のとおりです。

net45 の場合

```
C:/Program Files/edb/dotnet/EF/net45
```

次の共有ライブラリファイルが必要です。

```
EntityFramework5.EnterpriseDB.EDBClient.dll
```

```
EntityFramework6.EnterpriseDB.EDBClient.dll
```

net461 の場合

```
C:/Program Files/edb/dotnet/EF/net461
```

次の共有ライブラリファイルが必要です。

```
EntityFramework6.EnterpriseDB.EDBClient.dll
```

注釈: EntityFramework は、net45、net451、net461 のサブディレクトリにある EnterpriseDB.EDBClient.dll ライブラリで使用できます。

:ref:`ライブラリファイルの参照を参照してください<referencing\_the\_library\_files>`ライブラリファイルの参照に関する情報。

Postgres の ADO.NET ドライバの <DbProviderFactories> エントリを app.config ファイルに追加します。次のエントリを追加します。

```
<add name="EnterpriseDB.EDBClient"
  invariant="EnterpriseDB.EDBClient"
  description=".NET Data Provider for EnterpriseDB PostgreSQL"
  type="EnterpriseDB.EDBClient.EDBFactory, EnterpriseDB.EDBClient, Version=4.1.3.1,
  ↪Culture=neutral, PublicKeyToken=5d8b90d52f46fda7"
  support="FF"/>
```

プロジェクトの app.config ファイルで、EntityFramework/providers タグの下にプロバイダーサービスの次のエントリを追加します。

```
<provider invariantName="EnterpriseDB.EDBClient"
  type="EnterpriseDB.EDBClient.EDBServices, EntityFramework6.EnterpriseDB.EDBClient">
</provider>
```

以下は app.config ファイルの例です:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.
↪EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral,
↪PublicKeyToken=b77a5c561934e089" requirePermission="false"/>
  </configSections>

  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>

  <entityFramework>
    <providers>
      <provider invariantName="EnterpriseDB.EDBClient" type="EnterpriseDB.EDBClient.
↪EDBServices, EntityFramework6.EnterpriseDB.EDBClient"></provider>
    </providers>
  </entityFramework>

  <system.data>
    <DbProviderFactories>
      <remove invariant="EnterpriseDB.EDBClient"/>
      <add name="EnterpriseDB Data Provider" invariant="EnterpriseDB.EDBClient" support=
↪"FF" description=".Net Framework Data Provider for Postgresql" type="EnterpriseDB.
↪EDBClient.EDBFactory, EnterpriseDB.EDBClient"/>
    </DbProviderFactories>
  </system.data>
</configuration>
```

注釈: <providers> と <DbProviderFactories> の同じエントリは、web.config ファイルと app.config ファイルに有効です。

使用しているアプリケーションの種類によっては、名前空間をソースコードにインポートする必要がある場合があります（[ライブラリファイルの参照を参照](#)）。

EntityFramework の使用方法については、Microsoft のドキュメントを参照してください。

4.2.4 EnterpriseDBVSIXforVisualStudio2015/2017/2019

EDBDataDesignerExtensibilityProvider (EnterpriseDBVSIX) は、AdvancedServer データベースアクセスを VisualStudio に統合するコンポーネントであり、VisualStudio 統合機能を提供します。

EnterpriseDBVSIX を使用すると、VisualStudio のサーバーエクスプローラー内から AdvancedServer に接続し、既存のデータベースからモデルを作成できます。したがって、VisualStudio の機能が必要な場合は、EnterpriseDBVSIX を使用する必要があります。

EnterpriseDBVSIX ファイルは次のディレクトリにあります。

net45 の場合

```
C:/Program Files/edb/dotnet/vsix/net45
```

上記の場所にあるファイルは次のとおりです。

```
EnterpriseDB.vsix  
SSDLToPgSQL.tt  
System.ValueTuple.dll
```

net461 の場合

```
C:/Program Files/edb/dotnet/vsix/net461
```

上記の場所にあるファイルは次のとおりです。

```
EnterpriseDB.vsix  
SSDLToPgSQL.tt  
System.ValueTuple.dll
```

4.2.4.1 VisualStudio2015/2017/2019 のインストールと構成

EnterpriseDBVSIX をインストールして構成するには、次の手順を使用します。

ステップ 1: EnterpriseDBVSIX を EnterpriseDB.vsix インストーラーで VisualStudio の必要なバージョンにインストールします。

以前のバージョンの VSIX がすでにインストールされている場合は、競合を避けるためにアンインストールすることを強くお勧めします。

グローバルアセンブリキャッシュ (GAC) に EnterpriseDB.EDBClient を含める必要はなくなりました。

ステップ 2: VisualStudio を再起動し、Tools > Extensions and Updates... メニューから EnterpriseDB 拡張機能がインストールされていることを確認します。

ステップ 3: VisualStudioDevelopers コマンドラインで gacutil ユーティリティを使用して、次のライブラリをグローバルアセンブリキャッシュ（GAC）に追加します。

```
System.ValueTuple.dll  
System.Threading.Tasks.Extensions.dll  
System.Runtime.CompilerServices.Unsafe.dll  
System.Memory.dll
```

例えば:

```
> gacutil.exe /i System.ValueTuple.dll
```

ステップ 4: サーバーエクスプローラーで、Data Connections を右クリックし、Add Connection をクリックして、Enterprisedb Postgres Database データソースが使用可能であることを確認します。

4.2.4.2 モデルファーストとデータベースファーストの使用法

ステップ 1: VisualStudioDevelopers コマンドラインで gacutil ユーティリティを使用して、グローバルアセンブリキャッシュ (GAC) に EntityFramework5.EnterpriseDB.EDBClient.dll ライブラリを追加します。

例えば:

```
> gacutil.exe /i EntityFramework5.EnterpriseDB.EDBClient.dll
```

ステップ 2: ADO.NET ドライバーの <DbProviderFactories> エントリを machine.config ファイルに追加します。次のエントリを含めます。

```
<add name="EnterpriseDB.EDBClient"
  invariant="EnterpriseDB.EDBClient"
  description=".NET Data Provider for EnterpriseDB PostgreSQL"
  type="EnterpriseDB.EDBClient.EDBFactory, EnterpriseDB.EDBClient, Version=4.1.3.1,
  ↪Culture=neutral, PublicKeyToken=5d8b90d52f46fda7"
  support="FF"/>
```

属性と値のペアの場合、二重引用符で囲まれた文字列に余分な空白文字を含めないでください。1 行で構成してください。このセクションに示す例は、明確にするために複数の行に分割されている場合がありますが、実際には次のように 1 行で構成する必要があります。

```
description="/.NET Data Provider for EnterpriseDB PostgreSQL/"
```

64 ビット Windows の場合、machine.config ファイルは次の場所にあります:

```
C:/Windows/Microsoft.NET/Framework64/v4.0.30319/Config/machine.config
```

32 ビット Windows の場合、machine.config ファイルは次の場所にあります:

```
C:/Windows/Microsoft.NET/Framework/v4.0.30319/Config/machine.config
```

ステップ 3: 次の例に示すように、DDL 生成テンプレート SSDLTpPgSQL.tt を VisualStudio EntityFramework Tools/DBGen/ フォルダーに配置します:

```
C:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\IDE\Extensions\Microsoft\
  ↪EntityFramework Tools\DBGen\
```

注釈: EDMX ファイルのプロパティでこのテンプレート SSDLTpPgSQL.tt を選択します。

ステップ 4: プロジェクト参照に EnterpriseDB.EDBClient.dll および EntityFramework6.EnterpriseDB.EDBClient.dll ファイルを追加します。:ref: ライブラリファイルの参照を参照してください<referencing_the_library_files> ライブラリファイルの参照に関する情報。

ステップ 5: プロジェクトの app.config ファイルで、EntityFramework/providers タグの下にプロバイダーサービスの次のエントリを追加します。

```
<provider invariantName="EnterpriseDB.EDBClient"
  type="EnterpriseDB.EDBClient.EDBServices, EntityFramework6.EnterpriseDB.EDBClient">
</provider>
```

以下は app.config ファイルの例です。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.
↳EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral,
↳PublicKeyToken=b77a5c561934e089" requirePermission="false"/>
  </configSections>

  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>

  <entityFramework>
    <providers>
      <provider invariantName="EnterpriseDB.EDBClient" type="EnterpriseDB.EDBClient.
↳EDBServices, EntityFramework6.EnterpriseDB.EDBClient"></provider>
    </providers>
  </entityFramework>

  <system.data>
    <DbProviderFactories>
      <remove invariant="EnterpriseDB.EDBClient"/>
      <add name="EnterpriseDB Data Provider" invariant="EnterpriseDB.EDBClient"
↳support="FF" description=".Net Framework Data Provider for EDB Postgres" type=
↳"EnterpriseDB.EDBClient.EDBFactory, EnterpriseDB.EDBClient"/>
    </DbProviderFactories>
  </system.data>
</configuration>
```

第 5 章

セキュリティと暗号化

5.1 スクラムの互換性

EDB.NET ドライバーは、AdvancedServer バージョン 12、11、10 に SCRAM-SHA-256 サポートを提供します。
このサポートは、EDB.NET4.0.2.1 リリース以降で利用できます。

第 6 章

.NET コネクタの使用

以下のセクションでは、.NET アプリケーションが AdvancedServer データベースに接続して対話できるようにする AdvancedServer.NET コネクタによって提供される EDB オブジェクトクラスの使用例を示します。

このガイドの例を使用するには、アプリケーションのコンパイル済みフォームと同じディレクトリに .NET ライブラリファイルを配置します。すべての例は C # で記述されており、それぞれが ASP.NET ページに埋め込まれています。同じロジックとコードが他の .NET アプリケーション（WinForm またはコンソールアプリケーションなど）にも適用されます。

以下の web.config ファイルをサンプルコードと同じディレクトリに作成して保存してください。例では、この構成ファイルの DB_CONN_STRING キーを使用して、AdvancedServer ホストから接続文字列を返します。

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <appSettings>
    <add key="DB_CONN_STRING" value="Server=127.0.0.1;Port=5444;
      User Id=enterprisedb;Password=enterprisedb;Database=edb" />
  </appSettings>
</configuration>
```

ASP.NETWeb アプリケーションの AdvancedServer 接続文字列は、web.config ファイルに格納されます。ASP.NET を使用しないアプリケーションを作成している場合は、アプリケーション構成ファイル（app.config など）に接続情報を提供します。

第 7 章

データベース接続を開く

An `EDBConnection` object is responsible for handling the communication between an instance of Advanced Server and a .NET application. Before you can access data stored in an Advanced Server database, you must create and open an `EDBConnection`.

次の例は、`AdvancedServer` のインスタンスに接続するための基本的な手順を示しています。絶対です:

1. 名前空間 `EnterpriseDB.EDBClient` をインポートします。
2. `EDBConnection` のインスタンスを作成します。
3. Initialize the `EDBConnection` object by passing a connection string as a parameter to the constructor for the `EDBConnection` class.
4. `EDBConnection` オブジェクトの `Open` メソッドを呼び出して接続を開きます。

7.1 接続文字列パラメータ

有効な接続文字列は、`AdvancedServer` インスタンスの場所と認証情報を指定する必要があります。接続を開く前に、接続文字列を指定する必要があります。接続文字列には以下が含まれている必要があります。

- サーバーの名前または IP アドレス
- `AdvancedServer` データベースの名前
- `AdvancedServer` ユーザーの名前
- そのユーザーに関連付けられているパスワード

次のパラメータが接続文字列に含まれている場合があります。

`CommandTimeout`

CommandTimeout は、例外をスローする前にコマンドの実行が完了するのを待つ時間（秒単位）を指定します。デフォルト値は 20 です。

ConnectionLifeTime

ConnectionLifeTime を使用して、プール内の未使用の接続を閉じるまでの待機時間（秒単位）を指定します。デフォルト値は 15 です。

Database

Database パラメータを使用して、アプリケーションが接続するデータベースの名前を指定します。データベース名が指定されていない場合、データベース名はデフォルトで接続ユーザーの名前になります。

Encoding

Encoding パラメータは廃止されました; パラメータは常に文字列 `unicode` を返し、暗黙的に設定の試行を無視します。

Integrated Security

デフォルトでは、Integrated Security は `false` に設定されており、Windows 統合セキュリティは無効になっています。Windows 統合セキュリティを使用するには、値 `true` を指定します。

Load Role Based Tables

ロールに基づいてテーブル OID をロードするには Load Role Based Tables を使用します。この変更は、テーブルタイプ OID の読み込みにのみ影響し、複合タイプには影響しません。デフォルト値は `false` です。このパラメータを `true` に設定すると、新しい機能がトリガーされます。

MaxPoolSize

MaxPoolSize は、プールが指定された接続数を超えたときに、プールされた接続を破棄するように EDBConnection に指示します。デフォルト値は 20 です。

MinPoolSize

MinPoolSize は 1 に対して、指定された数の接続をサーバーに事前に割り当てるよう指示します。デフォルト値は 1 です。

Password

クリアテキスト認証を使用する場合は、サーバーとの接続を確立するために使用されるパスワードを指定します。

Pooling

デフォルトでは、接続プールを有効にするために Pooling は `true` に設定されています。接続プールを無効にするには、値 `false` を指定します。

Port

Port パラメータはアプリケーションが接続するポートを指定します。

Protocol

(自動の代わりに) 使用する特定のプロトコルバージョン。2 または 3 の整数値を指定します。

SearchPath

SearchPath パラメータを使用して、検索パスを名前付きスキーマとパブリックスキーマに変更します。

Server

AdvancedServer ホストの名前または IP アドレス。

SSL

デフォルトでは、SSL は false に設定されています。安全な接続を試みるには、値 true を指定します。

sslmode

Use sslmode to specify an SSL connection control preference. sslmode can be:

prefer -可能であれば SSL を使用します。

require -SSL 接続を確立できない場合に例外をスローします。

allow -SSL なしで接続します。このパラメーターはサポートされていません。

disable -SSL 接続を試みません。これがデフォルトの動作です。

SyncNotification

SyncNotification パラメータを使用して、EDBDataprovider が同期通知を使用するように指定します。デフォルト値は false です。

Timeout

Timeout は、開いている接続を待つ時間の長さ（秒単位）を指定します。デフォルト値は 15 です。

User Id

User Id パラメータは接続に使用されるべきユーザー名を指定します。

7.2 例-ASP.NET を使用してデータベース接続を開く

The following example demonstrates how to open a connection to an instance of Advanced Server and then close the connection. The connection is established using the credentials specified in the DB_CONN_STRING configuration parameter (see *Using the .Net Connector* for an introduction to connection information and also see *Connection String Parameters* for connection parameters).

```
<% @ Page Language="C#" %>
<% @Import Namespace="EnterpriseDB.EDBClient" %>
<% @Import Namespace="System.Configuration" %>

<script language="C#" runat="server">

private void Page_Load(object sender, System.EventArgs e)
{
    string strConnectionString = ConfigurationSettings.AppSettings
        ["DB_CONN_STRING"];

    EDBConnection conn = new EDBConnection(strConnectionString);

    try
    {
        conn.Open();
        Response.Write("Connection opened successfully");
    }

    catch(EDBException exp)
    {
        exp.ToString();
    }

    finally
    {
        conn.Close();
    }
}

</script>
```

接続が成功すると、ブラウザに次のように表示されます。

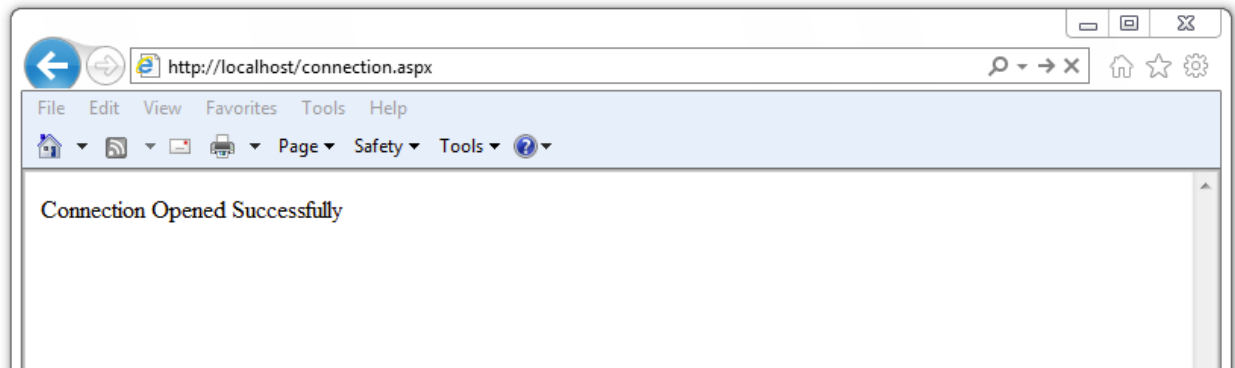


図 1 接続が正常に開かれました

7.3 例-コンソールアプリケーションからデータベース接続を開く

次の例では、コンソールベースのアプリケーションを使用して AdvancedServer データベースとの接続を開きます。

コンソールアプリケーションのコードを書く前に、データベースへの接続文字列を格納する app.config ファイルを作成します。構成ファイルを使用すると、情報が変更された場合に接続文字列を更新するのに便利です。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="DB_CONN_STRING" value = "Server=127.0.0.1;Port=5444;
      User Id=enterprisedb;Password=enterprisedb;Database=edb"/>
  </appSettings>
</configuration>
```

任意のテキストエディターを使用して、次のコードサンプルをファイルに入力します。

```
using System;
using System.Data;
using EnterpriseDB.EDBClient;
using System.Configuration;

namespace EnterpriseDB
{
    class EDB
    {
        static void Main(string[] args)
        {
            string strConnectionString = ConfigurationSettings.AppSettings
                ["DB_CONN_STRING"];

            EDBConnection conn = new EDBConnection(strConnectionString);
```

(次のページに続く)

(前のページからの続き)

```
        try
        {
            conn.Open();
            Console.WriteLine("Connection Opened Successfully");
        }

        catch(Exception exp)
        {
            throw new Exception(exp.ToString());
        }

        finally
        {
            conn.Close();
        }
    }
}
```

ファイルを EDBConnection-Sample.cs として保存し、次のコマンドでコンパイルします:

```
csc /r:EDBDataProvider.dll /out:Console.exe EDBConnection-Sample.cs
```

Compiling the sample should generate a Console.exe file; you can execute the sample code by entering Console.exe. When executed, the console should verify that the:

```
Connection Opened Successfully
```

7.4 例-**Windows** フォームアプリケーションからデータベース接続を開く

次の例は、.NETWinForm アプリケーションを使用してデータベース接続を開く方法を示しています。例を使用するには、次のコードをライブラリファイルを含むディレクトリに WinForm-Example.cs として保存します。

```
using System;
using System.Windows.Forms;
using System.Drawing;
using EnterpriseDB.EDBClient;

namespace EDBTestClient
{
    class Win_Conn
    {
        static void Main(string[] args)
        {
            Form frmMain = new Form();
            Button btnConn = new Button();
            btnConn.Location = new System.Drawing.Point(104, 64);
            btnConn.Name = "btnConn";
            btnConn.Text = "Open Connection";
            btnConn.Click += new System.EventHandler(btnConn_Click);

            frmMain.Controls.Add(btnConn);
            frmMain.Text = "EnterpriseDB";

            Application.Run(frmMain);
        }

        private static void btnConn_Click(object sender, System.EventArgs e)
        {
            EDBConnection conn = null;
            try
            {
                string connectionString = "Server=10.90.1.29;port=5444;
                username=edb;password=edb;database=edb";
                conn = new EDBConnection(connectionString);
                conn.Open();
                MessageBox.Show("Connection Open");
            }
            catch(EDBException exp)
            {
                MessageBox.Show(exp.ToString());
            }
            finally
            {
                conn.Close();
            }
        }
    }
}
```

(次のページに続く)

(前のページからの続き)

```
}  
    }  
}  
}
```

次のコマンドでファイルをコンパイルする前に、接続先のデータベースを指すようにデータベース接続文字列を変更する必要があることに注意してください。

```
csc /r:EDBDataProvider.dll /out:WinForm.exe WinForm-Example.cs
```

このコマンドは、実行可能ファイルがコンパイルされたのと同じフォルダー内に WinForm.exe ファイルを生成します。実行可能ファイルを呼び出すと、次が表示されます。

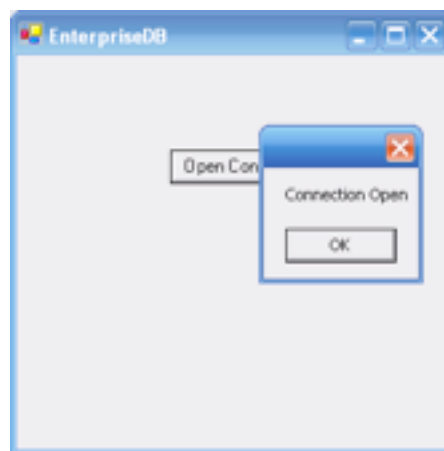


図 2 正常な接続

第 8 章

データベースレコードの取得

You can use a `SELECT` statement to retrieve records from the database via a `SELECT` command. To execute a `SELECT` statement you must:

- データベース接続を作成して開きます。
- `SELECT` ステートメントを表す `EDBCommand` オブジェクトを作成します。
- `EDBDataReader` を返す `EDBCommand` オブジェクトの `ExecuteReader()` メソッドでコマンドを実行します
- Loop through the `EDBDataReader` displaying the results or binding the `EDBDataReader` to some control.

An `EDBDataReader` object represents a forward-only and read-only stream of database records, presented one record at a time. To view a subsequent record in the stream, you must call the `Read()` method of the `EDBDataReader` object.

次の例:

1. `AdvancedServer` ネームスペースをインポートします:`EnterpriseDB.EDBClient`
2. `SELECT` ステートメントで `EDBCommand` オブジェクトを初期化します。
3. データベースへの接続を開きます。
4. Executes the `EDBCommand` by calling the `ExecuteReader` method of the `EDBCommand` object.

SQL ステートメントの結果は `EDBDataReader` オブジェクトに取得されます。
5. `EDBDataReader` オブジェクトの内容をループして、クエリによって返されたレコードを `WHILE` ループ内に表示します。

`Read()` メソッドは次のレコードに進み（レコードが存在する場合）、レコードが存在する場合は `true` を返し、`EDBDataReader` が末尾に達したことを示す場合は `false` を返します結果セット。

```
<% @ Page Language="C#" %>
<% @Import Namespace="EnterpriseDB.EDBClient" %>
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Configuration" %>

<script language="C#" runat="server">

private void Page_Load(object sender, System.EventArgs e)
{
    string strConnectionString = ConfigurationSettings.AppSettings
        ["DB_CONN_STRING"];
    EDBConnection conn = new EDBConnection(strConnectionString);

    try
    {
        conn.Open();
        EDBCommand cmdSelect = new EDBCommand("SELECT * FROM dept", conn);
        cmdSelect.CommandType = CommandType.Text;
        EDBDataReader drDept = cmdSelect.ExecuteReader();

        while(drDept.Read())
        {
            Response.Write("Department Number: " + drDept["deptno"]);
            Response.Write("\tDepartment Name: " + drDept["dname"]);
            Response.Write("\tDepartment Location: " + drDept["loc"]);
            Response.Write("<br>");
        }

    }

    catch(Exception exp)
    {
        Response.Write(exp.ToString());
    }
    finally
    {
        conn.Close();
    }
}
</script>
```

サンプルコードを実行するには、次の名前のファイルのデフォルトの **Web** ルートディレクトリにコードを保存します。

selectEmployees.aspx

プログラムを呼び出すには、**Web** ブラウザーを開き、以下を参照します。

<http://localhost/selectEmployees.aspx>

8.1 単一のデータベースレコードの取得

クエリから単一の結果を取得するには、EDBCommand オブジェクトの ExecuteScalar() メソッドを使用します。ExecuteScalar() メソッドは、指定されたクエリによって生成された DataSet の最初の行の最初の列の最初の値を返します。

```
<% @ Page Language="C#" %>
<% @Import Namespace="EnterpriseDB.EDBClient" %>
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Configuration" %>

<script language="C#" runat="server">

private void Page_Load(object sender, System.EventArgs e)
{
    string strConnectionString = ConfigurationSettings.AppSettings
        ["DB_CONN_STRING"];
    EDBConnection conn = new EDBConnection(strConnectionString);
    try
    {
        conn.Open();
        EDBCommand cmd = new EDBCommand("SELECT MAX(sal) FROM emp", conn);

        cmd.CommandType = CommandType.Text;

        int maxSal = Convert.ToInt32(cmd.ExecuteScalar());

        Response.Write("Emp Number: " + maxSal);

    }
    catch(Exception exp)
    {
        Response.Write(exp.ToString());
    }
    finally
    {
        conn.Close();
    }
}
</script>
```

次の名前の Web ルートディレクトリのファイルにサンプルコードを保存します。

selectscalar.aspx

サンプルコードを呼び出すには、Web ブラウザーを開き、以下を参照します。

<http://localhost/selectScalar.aspx>

サンプルには `ExecuteScalar()` メソッドによって返された値の明示的な変換が含まれていることに注意してください。 `ExecuteScalar()` メソッドはオブジェクトを返します。オブジェクトを表示するには、 `Convert.ToInt32` メソッドを使用してオブジェクトを整数値に変換する必要があります。

第 9 章

パラメータ化されたクエリ

parameterized query は SQL ステートメントに埋め込まれた 1 つ以上のパラメーターマーカーを持つクエリです。パラメータ化されたクエリを実行する前に、SQL ステートメントのテキストで見つかった各マーカーの値を指定する必要があります。

パラメータ化されたクエリは、コードの作成時にクエリの完全なテキストがわからない場合に役立ちます。たとえば、WHERE 節で参照される値は、ユーザー入力から計算される場合があります。

次の例に示すように、EDBParameter オブジェクトを作成し、そのオブジェクトをコマンドのパラメーターコレクションに追加することにより、パラメーター化されたクエリで指定された各パラメーターのデータ型を宣言する必要があります。次に、パラメータの Value() 関数を呼び出して、各パラメータに value を指定する必要があります。

この例では、従業員の給与を増やす UPDATE ステートメントでパラメーター化クエリを使用する方法を示しています:

```
<% @ Page Language="C#" Debug="true"%>
<% @Import Namespace="EnterpriseDB.EDBClient" %>
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Configuration" %>

<script language="C#" runat="server" >

private void Page_Load(object sender, System.EventArgs e)
{
    string strConnectionString = ConfigurationSettings.AppSettings
        ["DB_CONN_STRING"];
    EDBConnection conn = new EDBConnection(strConnectionString);

    string updateQuery = "UPDATE emp SET sal = sal+500 where empno = :ID";

    try {
        conn.Open();
```

(次のページに続く)

(前のページからの続き)

```
EDBCommand cmdUpdate = new EDBCommand(updateQuery, conn);

cmdUpdate.Parameters.Add
(new EDBParameter(":ID", EDBTypes.EDBDbType.Integer));

cmdUpdate.Parameters[0].Value = 7788;

cmdUpdate.ExecuteNonQuery();

Response.Write("Record Updated");

}
catch(Exception exp) {
    Response.Write(exp.ToString());
}
finally {
    conn.Close();
}
}
</script>
```

次の名前の **Web** ルートディレクトリのファイルにサンプルコードを保存します。

updateSalary.aspx

サンプルコードを呼び出すには、**Web** ブラウザーを開き、以下を参照します。

http://localhost/updateSalary.aspx

第 10 章

データベースにレコードを挿入

EDBCommand の ExecuteNonQuery() メソッドを使用して、INSERT コマンドで AdvancedServer ホストに格納されているデータベースにレコードを追加できます。

次の例では、INSERT コマンドが変数 cmd に格納されています。コロン (:) が前に付いた値は、インスタンス化されて値が割り当てられ、その後に続くステートメントの INSERT コマンドのパラメーターコレクションに追加される EDBParameters のプレースホルダーです。INSERT コマンドは cmdInsert オブジェクトの ExecuteNonQuery() メソッドによって実行されます。

この例では、新しい従業員を emp テーブルに追加します:

```
<% @ Page Language="C#" Debug="true"%>
<% @Import Namespace="EnterpriseDB.EDBClient" %>
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Configuration" %>

<script language="C#" runat="server" >

private void Page_Load(object sender, System.EventArgs e)
{
    string strConnectionString = ConfigurationSettings.AppSettings
        ["DB_CONN_STRING"];
    EDBConnection conn = new EDBConnection(strConnectionString);

    try
    {
        conn.Open();

        string cmd = "INSERT INTO emp(empno,ename) VALUES (:EmpNo, :ENAME)";

        EDBCommand cmdInsert = new EDBCommand(cmd,conn);

        cmdInsert.Parameters.Add(new EDBParameter(":EmpNo",
            EDBTypes.EDBDbType.Integer));
```

(次のページに続く)

(前のページからの続き)

```
cmdInsert.Parameters[0].Value = 1234;

cmdInsert.Parameters.Add(new EDBParameter(":ENAME",
    EDBTypes.EDBDbType.Text));

cmdInsert.Parameters[1].Value = "Lola";

cmdInsert.ExecuteNonQuery();
Response.Write("Record inserted successfully");
}

catch(Exception exp)
{
    Response.Write(exp.ToString());
}
finally
{
    conn.Close();
}
}
</script>
```

次の名前の **Web** ルートディレクトリのファイルにサンプルコードを保存します。

insertEmployee.aspx

サンプルコードを呼び出すには、**Web** ブラウザーを開き、以下を参照します。

http://localhost/insertEmployee.aspx

第 11 章

データベースのレコードを削除する

EDBCommand の ExecuteNonQuery() メソッドを使用して、DELETE ステートメントで AdvancedServer ホストに格納されているデータベースからレコードを削除できます。

次の例では、DELETE コマンドは変数 strDeleteQuery に格納されています。このコードは、従業員番号を Delete コマンド (EmpNo で指定) に渡します。次に、コマンドは ExecuteNonQuery() メソッドを使用して実行されます。次の例では、前の例で挿入された従業員を削除します。

```
<% @ Page Language="C#" Debug="true"%>
<% @Import Namespace="EnterpriseDB.EDBClient" %>
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Configuration" %>

<script language="C#" runat="server" >

private void Page_Load(object sender, System.EventArgs e)
{
    string strConnectionString = ConfigurationSettings.AppSettings
        ["DB_CONN_STRING"];

    EDBConnection conn = new EDBConnection(strConnectionString);

    string strDeleteQuery = "DELETE FROM emp WHERE empno = :ID";

    try
    {
        conn.Open();

        EDBCommand deleteCommand = new EDBCommand(strDeleteQuery, conn);

        deleteCommand.Parameters.Add
            (new EDBParameter(":ID", EDBTypes.EDBDbType.Integer));

        deleteCommand.Parameters[0].Value = 1234;
```

(次のページに続く)

(前のページからの続き)

```
        deleteCommand.ExecuteNonQuery();

        Response.Write("Record Deleted");

    }

    catch(Exception exp)
    {

        Response.Write(exp.ToString());

    }

    finally
    {

        conn.Close();

    }
}
</script>
```

次の名前の **Web** ルートディレクトリのファイルにサンプルコードを保存します。

```
deleteEmployee.aspx
```

サンプルコードを呼び出すには、**Web** ブラウザーを開き、以下を参照します。

```
http://localhost/deleteEmployee.aspx
```

第 12 章

.NET アプリケーションでの SPL ストアドプロシージャの使用

アプリケーションに SQL ステートメントを含めるには、次の 2 つの方法があります。

- .NET アプリケーションコードに SQL ステートメントを直接追加する。
- ストアドプロシージャに SQL ステートメントをパッケージ化し、.NET アプリケーションからストアドプロシージャを実行する。

場合によっては、ストアドプロシージャが埋め込み SQL ステートメントよりも優れていることがあります。ストアドプロシージャは、アプリケーションに直接埋め込まれた SQL ステートメントで複製するのが難しい複雑な条件およびループ構造をサポートします。

ストアドプロシージャを使用すると、パフォーマンスが大幅に向上することもわかります。ストアドプロシージャは、サーバー側で 1 回だけ解析、コンパイル、最適化する必要がありますが、アプリケーションに含まれる SQL ステートメントは、.NET アプリケーションから実行するたびに解析、コンパイル、最適化できます。

.NET アプリケーションでストアドプロシージャを使用するには、以下を行う必要があります。

1. AdvancedServer ホストに SPL ストアドプロシージャを作成します。
2. EnterpriseDB.EDBClient 名前空間をインポートします。
3. ストアドプロシージャの名前を EDBCommand のインスタンスに渡します。
4. EDBCommand.CommandType を CommandType.StoredProcedure に変更します。
5. Prepare () コマンド。
6. コマンドを実行します。

12.1 例-パラメータなしのストアードプロシージャの実行

サンプル手順では、部門 10 の名前を出力します。プロシージャはパラメータを取らず、パラメータを返しません。サンプルプロシージャを作成するには、EDB-PSQL を呼び出して AdvancedServer ホストデータベースに接続します。コマンドラインで次の SPL コードを入力します。

```
CREATE OR REPLACE PROCEDURE list_dept10
IS
    v_deptname VARCHAR2(30);
BEGIN
    DBMS_OUTPUT.PUT_LINE('Dept No: 10');
    SELECT dname INTO v_deptname FROM dept WHERE deptno = 10;
    DBMS_OUTPUT.PUT_LINE('Dept Name: ' || v_deptname);
END;
```

AdvancedServer がストアードプロシージャを検証すると、CREATE PROCEDURE をエコーします。

EDBCommand オブジェクトを使用してストアードプロシージャを実行する

The CommandType property of the EDBCommand object is used to indicate the type of command being executed. The CommandType property is set to one of three possible CommandType enumeration values:

- 実行のために SQL 文字列を渡す場合、デフォルトの Text 値を使用します。
- 実行するストアードプロシージャの名前を渡して、StoredProcedure 値を使用します。
- テーブル名を渡すときは TableDirect 値を使用します。この値は、指定されたテーブルのすべてのレコードを返します。

CommandText プロパティには、CommandType プロパティの値に応じて、SQL 文字列、ストアードプロシージャ名、またはテーブル名を含める必要があります。

次の例では、ストアードプロシージャを実行します。

```
<% @ Page Language="C#" Debug="true"%>
<% @Import Namespace="EnterpriseDB.EDBClient" %>
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Configuration" %>

<script language="C#" runat="server" >

private void Page_Load(object sender, System.EventArgs e)
{
    string strConnectionString = ConfigurationSettings.AppSettings
    ["DB_CONN_STRING"];
    EDBConnection conn = new EDBConnection(strConnectionString);
```

(次のページに続く)

(前のページからの続き)

```
try
{
    conn.Open();

    EDBCommand cmdStoredProc = new EDBCommand("list_dept10",conn);
    cmdStoredProc.CommandType = CommandType.StoredProcedure;

    cmdStoredProc.Prepare();
    cmdStoredProc.ExecuteNonQuery();

    Response.Write("Stored Procedure Executed Successfully");

}
catch(Exception exp)
{
    Response.Write(exp.ToString());
}
finally
{
    conn.Close();
}
}
</script>
```

次の名前の Web ルートディレクトリのファイルにサンプルコードを保存します。

storedProc.aspx

サンプルコードを呼び出すには、Web ブラウザーを開き、以下を参照します。

http://localhost/storedProc.aspx

12.2 例-IN パラメータを使用したストアードプロシージャの実行

次の例は、IN パラメータを含むストアードプロシージャの呼び出しを示しています。サンプルプロシージャを作成するには、EDB-PSQL を呼び出して AdvancedServer ホストデータベースに接続します。コマンドラインで次の SPL コードを入力します。

```
CREATE OR REPLACE PROCEDURE
EMP_INSERT
(
    pENAME IN VARCHAR,
    pJOB IN VARCHAR,
    pSAL IN FLOAT4,
    pCOMM IN FLOAT4,
    pDEPTNO IN INTEGER,
    pMgr IN INTEGER
)
AS
DECLARE
    CURSOR TESTCUR IS SELECT MAX(EMPNO) FROM EMP;
    MAX_EMPNO INTEGER := 10;
BEGIN

    OPEN TESTCUR;
    FETCH TESTCUR INTO MAX_EMPNO;
    INSERT INTO EMP (EMPNO, ENAME, JOB, SAL, COMM, DEPTNO, MGR)
        VALUES (MAX_EMPNO+1, pENAME, pJOB, pSAL, pCOMM, pDEPTNO, pMgr);
    CLOSE testcur;
END;
```

AdvancedServer がストアードプロシージャを検証すると、CREATE PROCEDURE をエコーします。

入力値をストアードプロシージャに渡す

パラメータを含むストアードプロシージャの呼び出しは、パラメータなしのストアードプロシージャの実行と非常に似ています。主な違いは、パラメータ化されたストアードプロシージャを呼び出すときに、EDBCommand オブジェクトの EDBParameter コレクションを使用する必要があることです。コレクション EDBCommand に EDBParameter が追加されると、ParameterName、DbType、Direction、Size、Value などのプロパティ`設定されています。

次の例は、C #からパラメーター化されたストアードプロシージャを実行するプロセスを示しています。

```
<% @ Page Language="C#" Debug="true"%>
<% @Import Namespace="EnterpriseDB.EDBClient" %>
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Configuration" %>

<script language="C#" runat="server" >
```

(次のページに続く)

(前のページからの続き)

```
private void Page_Load(object sender, System.EventArgs e)
{
    string strConnectionString = ConfigurationSettings.AppSettings
        ["DB_CONN_STRING"];
    EDBConnection conn = new EDBConnection(strConnectionString);

    string empName    = "EDB";
    string empJob      = "Manager";
    double salary      = 1000;
    double commission = 0.0;
    int    deptno      = 20;
    int    manager     = 7839;

    try
    {
        conn.Open();

        EDBCommand cmdStoredProc = new EDBCommand
            ("emp_insert (:EmpName, :Job, :Salary, :Commission, :DeptNo,
            :Manager)", conn);
        cmdStoredProc.CommandType = CommandType.StoredProcedure;

        cmdStoredProc.Parameters.Add(new EDBParameter
            ("EmpName", EDBTypes.EDBDbType.VarChar));
        cmdStoredProc.Parameters[0].Value = empName;

        cmdStoredProc.Parameters.Add(new EDBParameter
            ("Job", EDBTypes.EDBDbType.VarChar));
        cmdStoredProc.Parameters[1].Value = empJob;

        cmdStoredProc.Parameters.Add(new EDBParameter
            ("Salary", EDBTypes.EDBDbType.Float));
        cmdStoredProc.Parameters[2].Value = salary;

        cmdStoredProc.Parameters.Add(new EDBParameter
            ("Commission", EDBTypes.EDBDbType.Float));
        cmdStoredProc.Parameters[3].Value = commission;

        cmdStoredProc.Parameters.Add(new EDBParameter
            ("DeptNo", EDBTypes.EDBDbType.Integer));
        cmdStoredProc.Parameters[4].Value = deptno;

        cmdStoredProc.Parameters.Add
            (new EDBParameter("Manager", EDBTypes.EDBDbType.Integer));
        cmdStoredProc.Parameters[5].Value = manager;

        cmdStoredProc.Prepare();
    }
}
```

(次のページに続く)

(前のページからの続き)

```
cmdStoredProc.ExecuteNonQuery();

Response.Write("Following Information Inserted Successfully
<br>");
string empInfo = "Employee Name: " + empName + "<br>";
empInfo += "Job: " + empJob + "<br>";
empInfo += "Salary: " + salary + "<br>";
empInfo += "Commission: " + commission + "<br>";
empInfo += "Manager: " + manager + "<br>";

Response.Write(empInfo);

}

catch(Exception exp)
{
    Response.Write(exp.ToString());
}
finally
{
    conn.Close();
}
}

</script>

</script>
```

次の名前の **Web** ルートディレクトリのファイルにサンプルコードを保存します。

storedProcInParam.aspx

サンプルコードを呼び出すには、**Web** ブラウザーを開き、以下を参照します。

<http://localhost/storedProcInParam.aspx>

この例では、Page_Load メソッドの本体が EDBConnection オブジェクトを宣言してインスタンス化しています。次に、サンプルは、ストアドプロシージャの実行に必要なプロパティを持つ EDBCommand オブジェクトを作成します。

次に、この例では EDBCommand Parameter コレクションの Add メソッドを使用して 6 つの入力パラメーターを追加しています。

```
EDBCommand cmdStoredProc = new EDBCommand
("emp_insert(:EmpName,:Job,:Salary,:Commission,:DeptNo,:Manager)",conn);
cmdStoredProc.CommandType = CommandType.StoredProcedure;
```

EMP_INSERT ストアドプロシージャに渡す前に、各パラメーターに値を割り当てます

Prepare() メソッドは ExecuteNonQuery() メソッドを呼び出す前にステートメントを準備します。

EDBCommand オブジェクトの ExecuteNonQuery メソッドはストアドプロシージャを実行します。ストアドプロシージャの実行後、テストレコードが emp テーブルに挿入され、挿入された値が Web ページに表示されます。

12.3 例-IN、OUT、および INOUT パラメータを使用したストアードプロシージャの実行

前の例では、IN パラメータをストアードプロシージャに渡す方法を示しました。以下の例は、ストアードプロシージャから IN 値を渡し、OUT 値を返す方法を示しています。

ストアードプロシージャの作成

次のストアードプロシージャは、部門番号を渡し、対応する場所と部門名を返します。サンプルプロシージャを作成するには、EDB-PSQL コマンドラインを開き、AdvancedServer ホストデータベースに接続します。コマンドラインで次の SPL コードを入力します。

```
CREATE OR REPLACE PROCEDURE
  DEPT_SELECT
(
  pDEPTNO IN  INTEGER,
  pDNAME  OUT VARCHAR,
  pLOC    OUT  VARCHAR
)
AS
DECLARE
  CURSOR TESTCUR IS SELECT DNAME,LOC FROM DEPT;
  REC RECORD;
BEGIN

  OPEN TESTCUR;
  FETCH TESTCUR INTO REC;

  pDNAME  := REC.DNAME;
  pLOC    := REC.LOC;

  CLOSE testcur;
END;
```

AdvancedServer がストアードプロシージャを検証すると、CREATE PROCEDURE をエコーします。

ストアードプロシージャからの出力値の受信

When retrieving values from OUT parameters you must explicitly specify the direction of out parameters as Output. You can retrieve the values from Output parameters in two ways:

- EDBCommand の ExecuteReader メソッドを呼び出し、返された EDBDataReader を明示的にループして、OUT パラメータの値を検索します。
- EDBCommand の ExecuteNonQuery メソッドを呼び出し、宣言された Output パラメータの値を明示的に取得するには、その EDBParameter 値プロパティを呼び出します。

各メソッドでは、各パラメーターを宣言する必要があり、パラメーターの direction を示します

(ParameterDirection.Input 、 ParameterDirection.Output または ParameterDirection.InputOutput)。プロシージャを呼び出す前に、 IN および INOUT の各パラメータに値を指定する必要があります。プロシージャが戻った後、 command.Parameters[] 配列から OUT および INOUT パラメータ値を取得できます。

次のコードリストは、 ExecuteReader メソッドを使用して結果セットを取得する方法を示しています。

```
<% @ Page Language="C#" Debug="true"%>
<% @Import Namespace="EnterpriseDB.EDBClient" %>
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Configuration" %>

<script language="C#" runat="server" >

private void Page_Load(object sender, System.EventArgs e)
{
    string strConnectionString =
        ConfigurationSettings.AppSettings["DB_CONN_STRING"];
    EDBConnection conn = new EDBConnection(strConnectionString);

    try
    {
        conn.Open();
        EDBCommand command = new EDBCommand("DEPT_SELECT
            (:pDEPTNO,:pDNAME,:pLOC)", conn);
        command.CommandType = CommandType.StoredProcedure;

        command.Parameters.Add(new EDBParameter("pDEPTNO",
            EDBTypes.EDBDbType.Integer,10,"pDEPTNO",
            ParameterDirection.Input,false ,2,2,
            System.Data.DataRowVersion.Current,1));

        command.Parameters.Add(new EDBParameter("pDNAME",
            EDBTypes.EDBDbType.Varchar,10,"pDNAME",
            ParameterDirection.Output,false ,2,2,
            System.Data.DataRowVersion.Current,1));

        command.Parameters.Add(new EDBParameter("pLOC",
            EDBTypes.EDBDbType.Varchar,10,"pLOC",
            ParameterDirection.Output,false ,2,2,
            System.Data.DataRowVersion.Current,1));

        command.Prepare();

        command.Parameters[0].Value = 10;
        EDBDataReader result = command.ExecuteReader();
    }
}
```

(次のページに続く)

(前のページからの続き)

```
int fc = result.FieldCount;

while(result.Read())
{
    for(int i = 0; i < fc; i++)
    {
        Response.Write("RESULT["+i+"]="+ Convert.ToString
            (command.Parameters[i].Value));
        Response.Write("<br>");
    }
}

catch(EDBException exp)
{
    Response.Write(exp.ToString());
}
finally
{
    conn.Close();
}
}

</script>
```

次のコードリストは、ExecuteNonQuery メソッドを使用して結果セットを取得する方法を示しています。

```
<% @ Page Language="C#" Debug="true"%>
<% @Import Namespace="EnterpriseDB.EDBClient" %>
<% @Import Namespace="System.Data" %>
<% @Import Namespace="System.Configuration" %>

<script language="C#" runat="server" >

private void Page_Load(object sender, System.EventArgs e)
{
    string strConnectionString = ConfigurationSettings.AppSettings
        ["DB_CONN_STRING"];
    EDBConnection conn = new EDBConnection(strConnectionString);

    try
    {
        conn.Open();
        EDBCommand command = new EDBCommand("DEPT_SELECT
            (:pDEPTNO,:pDNAME,:pLOC)", conn);
```

(次のページに続く)

(前のページからの続き)

```
command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add(new EDBParameter("pDEPTNO",
    EDBTypes.EDBDbType.Integer,10,"pDEPTNO",
    ParameterDirection.Input,false ,2,2,
    System.Data.DataRowVersion.Current,1));

command.Parameters.Add(new EDBParameter("pDNAME",
    EDBTypes.EDBDbType.Varchar,10,"pDNAME",
    ParameterDirection.Output,false ,2,2,
    System.Data.DataRowVersion.Current,1));

command.Parameters.Add(new EDBParameter("pLOC",
    EDBTypes.EDBDbType.Varchar,10,"pLOC",
    ParameterDirection.Output,false ,2,2,
    System.Data.DataRowVersion.Current,1));

command.Prepare();
command.Parameters[0].Value = 10;
command.ExecuteNonQuery();

Response.Write(command.Parameters["pDNAME"].Value.ToString());
Response.Write(command.Parameters["pLOC"].Value.ToString());
}

catch(EDBException exp)
{
    Response.Write(exp.ToString());
}
finally
{
    conn.Close();
}
}
</script>
```


第 13 章

アドバンスドキューイングの使用

EDBPostgresAdvancedServerAdvancedQueuing は、AdvancedServer データベースにメッセージキューイングとメッセージ処理を提供します。ユーザー定義のメッセージはキューに格納されます。キューのコレクションは、キューテーブルに格納されます。依存するキューを作成する前に、まずキューテーブルを作成する必要があります。

On the server side, procedures in the DBMS_AQADM package create and manage message queues and queue tables. Use the DBMS_AQ package to add or remove messages from a queue, or register or unregister a PL/SQL callback procedure. For more information about DBMS_AQ and DBMS_AQADM, click [here](#).

クライアント側では、アプリケーションは EDB.NET ドライバーを使用してメッセージをエンキュー/デキューします。

13.1 メッセージのエンキューまたはデキュー

13.1.1 サーバー側の設定

.NET アプリケーションでアドバンスドキューイング機能を使用するには、まずユーザー定義型、キューテーブル、およびキューを作成してから、データベースサーバーでキューを開始する必要があります。EDB-PSQL を呼び出して、AdvancedServer ホストデータベースに接続します。コマンドラインで次の SPL コマンドを使用します。

ユーザー定義タイプの作成

RAW データ型を指定するには、ユーザー定義型を作成する必要があります。次の例は、myxml という名前のユーザー定義型の作成を示しています。

```
CREATE TYPE myxml AS (value XML);
```

キューテーブルの作成

キューテーブルは、同じペイロードタイプの複数のキューを保持できます。次の例は、MSG_QUEUE_TABLE という名前のテーブルの作成を示しています。

```
EXEC DBMS_AQADM.CREATE_QUEUE_TABLE
    (queue_table => 'MSG_QUEUE_TABLE',
     queue_payload_type => 'myxml',
     comment => 'Message queue table');
END;
```

キューの作成

次の例は、テーブル MSG_QUEUE_TABLE 内に MSG_QUEUE という名前のキューを作成する方法を示しています。

```
BEGIN
DBMS_AQADM.CREATE_QUEUE ( queue_name => 'MSG_QUEUE', queue_table => 'MSG_QUEUE_TABLE',
↳comment => 'This queue contains pending messages. ');
END;
```

キューの開始

キューが作成されたら、コマンドラインで次の SPL コードを呼び出して、EDB データベースのキューを開始します。

```
BEGIN
DBMS_AQADM.START_QUEUE
(queue_name => 'MSG_QUEUE');
END;
```

13.1.2 クライアント側の例

ユーザー定義型を作成し、続いてキューテーブルとキューを作成したら、キューを開始します。次に、EDB.Net ドライバーを使用してメッセージをエンキューまたはデキューできます。

メッセージをキューに入れる:

.NET アプリケーションでメッセージをエンキューするには、以下を行う必要があります。

1. EnterpriseDB.EDBClient 名前空間をインポートします。
2. キューの名前を渡し、EDBAQueue のインスタンスを作成します。
3. エンキューメッセージを作成し、ペイロードを定義します。
4. queue.Enqueue メソッドを呼び出します。

次のコードリストは、Queue.enqueue メソッドの使用を示しています:

```
using EnterpriseDB.EDBClient;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AQXml
{
    class MyXML
    {
        public string value { get; set; }
    }
    class Program
    {
        static void Main(string[] args)
        {
            int messagesToSend = 1;
            if (args.Length > 0 && !string.IsNullOrEmpty(args[0]))
            {
                messagesToSend = int.Parse(args[0]);
            }
            for (int i = 0; i < 5; i++)
            {
                EnqueueMsg("test message: " + i);
            }

            private static EDBConnection GetConnection()
            {
```

(次のページに続く)

(前のページからの続き)

```
        string connectionString = "Server=127.0.0.1;Host=127.0.0.1;Port=5444;User_
↪Id=enterprisedb;Password=test;Database=edb;Timeout=999";
        EDBConnection connection = new EDBConnection(connectionString);
        connection.Open();
        return connection;
    }

    private static string ByteArrayToString(byte[] byteArray)
    {
        // Sanity check if it's null so we don't incur overhead of an exception
        if (byteArray == null)
        {
            return string.Empty;
        }
        try
        {
            StringBuilder hex = new StringBuilder(byteArray.Length * 2);
            foreach (byte b in byteArray)
            {
                hex.AppendFormat("{0:x2}", b);
            }

            return hex.ToString().ToUpper();
        }
        catch
        {
            return string.Empty;
        }
    }

    private static bool EnqueueMsg(string msg)
    {
        EDBConnection con = GetConnection();
        using (EDBAQQueue queue = new EDBAQQueue("MSG_QUEUE", con))
        {
            queue.MessageType = EDBAQMessageType.Xml;
            EDBTransaction txn = queue.Connection.BeginTransaction();
            QueuedEntities.Message queuedMessage = new QueuedEntities.Message() {
↪MessageText = msg };

            try
            {
                string rootElementName = queuedMessage.GetType().Name;
                if (rootElementName.IndexOf(".") != -1)
                {
                    rootElementName = rootElementName.Split('.')[0];
                }
            }
        }
    }
}
```

(次のページに続く)

(前のページからの続き)

```
        string xml = new Utils.XmlFragmentSerializer<QueuedEntities.  
↪Message>().Serialize(queuedMessage);  
        EDBAQMessage queMsg = new EDBAQMessage();  
        queMsg.Payload = new MyXML { value = xml };  
        queue.MessageType = EDBAQMessageType.Udt;  
        queue.UdtTypeName = "myxml";  
        queue.Enqueue(queMsg);  
        var messageId = ByteArrayToString((byte[])queMsg.MessageId);  
        Console.WriteLine("MessageID: " + messageId);  
        txn.Commit();  
        queMsg = null;  
        xml = null;  
        rootElementName = null;  
        return true;  
    }  
    catch (Exception ex)  
    {  
        txn?.Rollback();  
        Console.WriteLine("Failed to enqueue message.");  
        Console.WriteLine(ex.ToString());  
        return false;  
    }  
    finally  
    {  
        queue?.Connection?.Dispose();  
    }  
}  
  
}
```

メッセージをキューから取り出す

.NET アプリケーションでメッセージをデキューするには、次のことを行う必要があります。

1. EnterpriseDB.EDBClient 名前空間をインポートします。
2. キューの名前を渡し、EDBAQQueue のインスタンスを作成します。
3. queue.Dequeue メソッドを呼び出します。

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;
```

(次のページに続く)

(前のページからの続き)

```
using EnterpriseDB.EDBClient;

namespace DequeueXML
{
    class MyXML
    {
        public string value { get; set; }
    }
    class Program
    {
        static void Main(string[] args)
        {
            DequeueMsg();
        }

        private static EDBConnection GetConnection()
        {
            string connectionString = "Server=127.0.0.1;Host=127.0.0.1;Port=5444;User_
↪Id=enterprisedb;Password=test;Database=edb;Timeout=999";
            EDBConnection connection = new EDBConnection(connectionString);
            connection.Open();
            return connection;
        }

        private static string ByteArrayToString(byte[] byteArray)
        {
            // Sanity check if it's null so we don't incur overhead of an exception
            if (byteArray == null)
            {
                return string.Empty;
            }
            try
            {
                StringBuilder hex = new StringBuilder(byteArray.Length * 2);
                foreach (byte b in byteArray)
                {
                    hex.AppendFormat("{0:x2}", b);
                }

                return hex.ToString().ToUpper();
            }
            catch
            {
                return string.Empty;
            }
        }
    }
}
```

(次のページに続く)

(前のページからの続き)

```
public static void DequeueMsg(int waitTime = 10)
{
    EDBConnection con = GetConnection();
    using (EDBAQQueue queueListen = new EDBAQQueue("MSG_QUEUE", con))
    {
        queueListen.UdtTypeName = "myxml";
        queueListen.DequeueOptions.Navigation = EDBAQNavigationMode.FIRST_
MESSAGE;
        queueListen.DequeueOptions.Visibility = EDBAQVisibility.ON_COMMIT;
        queueListen.DequeueOptions.Wait = 1;
        EDBTransaction txn = null;

        while (1 == 1)
        {
            if (queueListen.Connection.State == System.Data.ConnectionState.
CLOSED)
            {
                queueListen.Connection.Open();
            }

            string messageId = "Unknown";
            try
            {
                // the listen function is a blocking function. It will Wait
the specified waitTime or until a
                // message is received.
                Console.WriteLine("Listening...");
                string v = queueListen.Listen(null, waitTime);
                // If we are waiting for a message and we specify a Wait time,
                // then if there are no more messages, we want to just bounce
out.
                if (waitTime > -1 && v == null)
                {
                    Console.WriteLine("No message received during Wait period.
");
                    Console.WriteLine();
                    continue;
                }

                // once we're here that means a message has been detected in
the queue. Let's deal with it.
                txn = queueListen.Connection.BeginTransaction();

                Console.WriteLine("Attempting to dequeue message...");
                // dequeue the message
                EDBAQMessage deqMsg;
                try
```

(次のページに続く)

(前のページからの続き)

```
{
    deqMsg = queueListen.Dequeue();
}
catch (Exception ex)
{
    if (ex.Message.Contains("ORA-25228"))
    {
        Console.WriteLine("Message was not there.  Another
↪process must have picked it up.");
        Console.WriteLine();
        txn.Rollback();
        continue;
    }
    else
    {
        throw;
    }
}

messageId = ByteArrayToString((byte[]) deqMsg.MessageId);
if (deqMsg != null)
{
    Console.WriteLine("Processing received message...");
    // process the message payload
    MyXML obj = new MyXML();
    queueListen.Map<MyXML>(deqMsg.Payload, obj);

    QueuedEntities.Message msg = new Utils.
↪XmlFragmentSerializer<QueuedEntities.Message>().Deserialize(obj.value);

    Console.WriteLine("Received Message:");
    Console.WriteLine("MessageID: " + messageId);
    Console.WriteLine("Message: " + msg.MessageText);
    Console.WriteLine("Enqueue Time" + queueListen.
↪MessageProperties.EnqueueTime);

    txn.Commit();

    Console.WriteLine("Finished processing message");
    Console.WriteLine();
}
else
{
    Console.WriteLine("Message was not dequeued.");
}
}
catch (Exception ex)
```

(次のページに続く)

(前のページからの続き)

```
        {
            Console.WriteLine("Failed To dequeue or process the dequeued_↵
↵message.");

            Console.WriteLine(ex.ToString());
            Console.WriteLine();
            if (txn != null)
            {
                txn.Rollback();
                if (txn != null)
                {
                    txn.Dispose();
                }
            }
        }
    }
}
}
```

13.2 EDBAQ クラス

このアプリケーションでは、次の EDBAQ クラスが使用されています。

EDBAQDequeueMode

EDBAQDequeueMode クラスは利用可能なすべてのデキューモードをリストします。

値	説明
ブラウズ	ロックせずにメッセージを読みます。
ロックされています	メッセージを読み取り、書き込みロックを取得します。
削除する	読んだ後にメッセージを削除します。これがデフォルト値です。
Remove_NoData	メッセージの受信を確認します。

EDBAQDequeueOptions

EDBAQDequeueOptions クラスはメッセージをデキューするときに利用できるオプションをリストします。

プロパティ	説明
Consumer-Name	メッセージをデキューするコンシューマの名前。
Dequeue-Mode	これは EDBAQDequeueMode から設定されます。これは、デキューオプションにリンクされたロック動作を表します。
ナビゲーション	これは EDBAQNavigationMode から設定されます。フェッチされるメッセージの位置を表します。
可視性	これは EDBAQVisibility から設定されます。これは、新しいメッセージが現在のトランザクションの一部としてデキューされるかどうかを表します。
Wait	検索基準に基づくメッセージの待機時間。
Msgid	メッセージ識別子。
相関	相関識別子。
DeqCondition	デキューの状態。ブール式です。
変換	メッセージをデキューする前に適用される変換。
Delivery-Mode	デキューされたメッセージの配信モード。

EDBAQEnqueueOptions

EDBAQEnqueueOptions クラスは、メッセージをエンキューするときに利用できるオプションをリストします。

プロパティ	説明
可視性	これは EDBAQVisibility から設定されます。これは、新しいメッセージが現在のトランザクションの一部としてエンキューされるかどうかを表します。
RelativeMsgid	相対メッセージ識別子。
シーケンス偏差	メッセージをデキューする必要があるときのシーケンス。
変換	メッセージをエンキューする前に適用される変換。
Delivery-Mode	エンキューされたメッセージの配信モード。

EDBAQMessage

EDBAQMessage クラスは、エンキュー/デキューされるメッセージをリストします。

プロパティ	説明
ペイロード	キューに入れられる実際のメッセージ。
MessageId	キューに入れられたメッセージの ID。

EDBAQMessageProperties

EDBAQMessageProperties は利用可能なメッセージプロパティをリストします。

プロパティ	説明
優先	メッセージの優先度。
ディレイ	メッセージがデキューできる期間の投稿。これは秒単位で指定されます。
有効期限	メッセージがデキューに利用できる期間。これは秒単位で指定されます。
相関	相関識別子。
試み	メッセージのデキューを試行した回数。
RecipientList	デフォルトのキューサブスクライバーを破棄する受信者リスト。
ExceptionQueue	未処理のメッセージを移動するキューの名前。
EnqueueTime	メッセージがエンキューされた時刻。
状態	デキュー中のメッセージの状態。
OriginalMsgid	最後のキューのメッセージ ID。
TransactionGroup	デキューされたメッセージのトランザクショングループ。
DeliveryMode	デキューされたメッセージの配信モード。

EDBAQMessageState

EDBAQMessageState クラスは、デキュー中のメッセージの状態を表します。

値	説明
期限切れ	メッセージは例外キューに移動されます。
処理済み	メッセージは処理され、保持されます。
準備ができて	メッセージを処理する準備ができました。
Waiting	メッセージは待機状態です。遅延に達していません。

EDBAQMessageType

EDBAQMessageType クラスはペイロードのタイプを表します。

値	説明
Raw	Raw メッセージタイプ。 注:現在、このペイロードタイプはサポートされていません。
UDT	ユーザー定義タイプのメッセージ。
XML	XML タイプのメッセージ。 注:現在、このペイロードタイプはサポートされていません。

EDBAQNavigationMode

EDBAQNavigationMode クラスは、利用可能なさまざまなタイプのナビゲーションモードを表します。

値	説明
First_Message	検索語に一致する最初の利用可能なメッセージを返します。
Next_Message	検索項目に一致する次の使用可能なメッセージを返します。
Next_Transaction	次のトランザクショングループの最初のメッセージを返します。

EDBAQQueue

EDBAQQueue クラスは PostgreSQL データベースで DMBS_AQ 機能を実行する SQL ステートメントを表します。

プロパティ	説明
接続	使用する接続。
名前	キューの名前。
MessageType	このキューからエンキューまたはデキューされるメッセージタイプ。たとえば、EDBAQMessageType.Udt です。
UdtTypeName	メッセージタイプのユーザ定義タイプ名。
EnqueueOptions	使用するエンキューオプション。
DequeueOptions	使用するデキューオプション。
MessageProperties	使用するメッセージプロパティ。

EDBAQVisibility

EDBAQVisibility クラスは利用可能な可視性オプションを表します。

値	説明
Immediate	エンキュー/デキューは、進行中のトランザクションの一部ではありません。
On_Commit	エンキュー/デキューは、現在のトランザクションの一部です。

注釈:

- 上記のパラメータのデフォルトオプションを確認するには、[ここをクリックしてください<https://www.enterprisedb.com/docs/en/11.0/EPAS_BIP_Guide_v11/Database_Compatibility_for_Oracle_Developers_Built-in_Package_Guide.1.14.html#pID0E01HG0HA/>`\\_](https://www.enterprisedb.com/docs/en/11.0/EPAS_BIP_Guide_v11/Database_Compatibility_for_Oracle_Developers_Built-in_Package_Guide.1.14.html#pID0E01HG0HA/>`_)。
- EDBAQ 機能は、ユーザー定義タイプを使用して、エンキュー/デキュー操作を呼び出します。 NoTypeLoading はユーザー定義型をロードしないので、 Server Compatibility Mode=NoTypeLoading は EDBAQ では使用できません。

第 14 章

.NET アプリケーションでの参照カーソルの使用

A `ref cursor` is a cursor variable that contains a pointer to a query result set. The result set is determined by the execution of the `OPEN FOR` statement using the cursor variable. A cursor variable is not tied to a particular query like a static cursor. The same cursor variable may be opened a number of times with the `OPEN FOR` statement containing different queries and each time, a new result set will be created for that query and made available via the cursor variable. There are two ways to declare a cursor variable:

- 弱く型付けされた `ref` カーソルを宣言するには、`SYS_REFCURSOR` 組み込みデータ型を使用します。
- その型の変数を宣言する強く型付けされた `ref` カーソルを定義します。

`SYS_REFCURSOR` は、任意の結果セットに関連付けることができる `ref` カーソルタイプです。これは、弱く型付けされた参照カーソルと呼ばれます。次の例は、弱く型付けされた `ref` カーソルの宣言です。

```
name SYS_REFCURSOR;
```

以下は、強く型付けされた `ref` カーソルの例です。

```
TYPE <cursor_type_name> IS REF CURSOR RETURN emp%ROWTYPE;
```

ストアドプロシージャの作成

次のサンプルコードは、`refcur_inout_callee` というストアドプロシージャを作成します。サンプルプロシージャを作成するには、`EDB-PSQL` を呼び出して `AdvancedServer` ホストデータベースに接続します。コマンドラインで次の `SPL` コードを入力します。

```
CREATE OR REPLACE PROCEDURE
  refcur_inout_callee(v_refcur IN OUT SYS_REFCURSOR)
IS
BEGIN
  OPEN v_refcur FOR SELECT ename FROM emp;
END;
```

上記で定義された手順を.NET コードから使用するには、上記のスクリプトに示すように、IN パラメーターとして渡される参照カーソルのデータ型を指定する必要があります。

次の C #コードは、ストアードプロシージャを使用して、emp テーブルから従業員名を取得します。

```
using System;
using System.Data;
using EnterpriseDB.EDBClient;
using System.Configuration;

namespace EDBRefCursor
{
    class EmpRefcursor
    {
        [STAThread]
        static void Main(string[] args)
        {
            string strConnectionString =
                ConfigurationSettings.AppSettings["DB_CONN_STRING"];
            EDBConnection conn = new EDBConnection(strConnectionString);
            conn.Open();
            EDBTransaction tran = conn.BeginTransaction();
            try
            {
                EDBTransaction tran = conn.BeginTransaction();
                EDBCommand command = new EDBCommand("refcur_inout_callee",
                    conn);
                command.CommandType = CommandType.StoredProcedure;
                command.Transaction = tran;
                command.Parameters.Add(new EDBParameter("refCursor",
                    EDBTypes.EDBDbType.Refcursor, 10, "refCursor",
                    ParameterDirection.InputOutput, false, 2, 2,
                    System.Data.DataRowVersion.Current, null));

                command.Prepare();
                command.Parameters[0].Value = null;

                command.ExecuteNonQuery();
                String cursorName = command.Parameters[0].Value.ToString();
                command.CommandText = "fetch all in \"\" + cursorName + \"\"";
                command.CommandType = CommandType.Text;

                EDBDataReader reader =
                    command.ExecuteReader(CommandBehavior.SequentialAccess);
                int fc = reader.FieldCount;
                while (reader.Read())
                {
                    for (int i = 0; i < fc; i++)
```

(次のページに続く)

(前のページからの続き)

```
        {
            Console.WriteLine(reader.GetString(i));
        }
    }
    reader.Close();
    tran.Commit();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message.ToString());
}
}
```

次の.NET コードスニペットは、コンソールに結果を表示します。

```
for(int i = 0; i < fc; i++)
{
    Console.WriteLine(reader.GetString(i));
}
```

ref カーソルパラメータを使用している場合は、`EDBDbType.RefCursor` 型を `EDBParameter()` にバインドする必要があることに注意してください。

第 15 章

プラグインの使用

EDB.Net ドライバープラグインが導入され、他の方法では.Net では利用できないさまざまなデータ型の拡張機能をサポートします。利用可能なさまざまなプラグインのサポート:

- GeoJSON
- Json.NET
- レガシー PostGIS
- NetTopologySuite
- NodaTime
- Rawpostgis

プラグインは、空間、データ/時間、および Json タイプの使用をサポートします。以下のセクションでは、これらのプラグインでサポートされているフレームワークとデータプロバイダーのインストールパスについて詳しく説明します。

15.1 GeoJSON

.NETFramework4.5 で GeoJSON プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/GeoJSON/net45
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.GeoJSON.dll
```

```
GeoJSON.Net.dll
```

```
Newtonsoft.Json.dll
```

.NETFramework4.6.1 で GeoJSON プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/GeoJSON/net461
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.GeoJSON.dll
```

```
GeoJSON.Net.dll
```

```
Newtonsoft.Json.dll
```

.NETStandard2.0 で GeoJSON プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/GeoJSON/netstandard2.0
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.GeoJSON.dll
```

GeoJSON プラグインの使用の詳細については、`Npgsql` のドキュメントを参照してください
<<http://www.npgsql.org/doc/types/geojson.html>>`。

15.2 Json.NET

.NETFramework4.5 で Json.NET プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/Json.NET/net45
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.Json.NET.dll  
Newtonsoft.Json.dll
```

.NETFramework4.6.1 で Json.NET プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/Json.NET/net461
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.Json.NET.dll  
Newtonsoft.Json.dll
```

.NETStandard2.0 で Json.NET プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/Json.NET/netstandard2.0
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.Json.NET.dll
```

Json.NET プラグインの使用に関する詳細については、`Npgsql` のドキュメントを参照してください
<<http://www.npgsql.org/doc/types/jsonnet.html>>`。

15.3 LegacyPostGIS

.NetFramework4.5 で LegacyPostGIS プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/LegacyPostgis/net45
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.LegacyPostgis.dll
```

.NetFramework4.6.1 で LegacyPostGIS プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/LegacyPostgis/net461
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.LegacyPostgis.dll
```

.NetStandard2.0 で LegacyPostGIS プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/LegacyPostgis/netstandard2.0
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.LegacyPostgis.dll
```

LegacyPostGIS プラグインの使用に関する詳細については、`Npgsql のドキュメントを参照してください`
<[<http://www.npgsql.org/doc/types/legacy-postgis.html/>](http://www.npgsql.org/doc/types/legacy-postgis.html)>`。

15.4 NetTopologySuite

.NetFramework4.5 で NetTopologySuite プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/ plugins/NetTopologySuite/net45
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.NetTopologySuite.dll  
GeoAPI.dll  
NetTopologySuite.dll  
NetTopologySuite.IO.PostGis.dll
```

.NetFramework4.6.1 で NetTopologySuite プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/ plugins/NetTopologySuite/net461
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.NetTopologySuite.dll  
NetTopologySuite.dll  
NetTopologySuite.IO.PostGis.dll
```

.NetStandard2.0 で NetTopologySuite プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/NetTopologySuite/netstandard2.0
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.NetTopologySuite.dll
```

NetTopologySuite タイプのプラグインの使用に関する詳細については、`Npgsql` のドキュメントを参照してください<<http://www.npgsql.org/doc/types/nts.html>>`_。

15.5 NodaTime

.NetFramework4.5 で NodaTime プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/NodaTime/net45
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.NodaTime.dll  
NodaTime.dll
```

.NetFramework4.6.1 で NodaTime プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/NodaTime/net461
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.NodaTime.dll  
NodaTime.dll
```

.NetStandard2.0 で NodaTime プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/NodaTime/netstandard2.0
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.NodaTime.dll
```

NodaTime プラグインの使用に関する詳細については、`Npgsql` のドキュメントを参照してください [<http://www.npgsql.org/doc/types/nodatime.html/>](http://www.npgsql.org/doc/types/nodatime.html)。

15.6 RawPostGIS

.NetFramework4.5 で RawPostGIS プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/RawPostgis/net45
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.RawPostgis.dll
```

.NetFramework4.6.1 で RawPostGIS プラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program Files/edb/dotnet/plugins/RawPostgis/net461
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.RawPostgis.dll
```

.NetStandard2.0 で RawPostGIS タイプのプラグインを使用している場合、データプロバイダーのインストールパスは次のとおりです。

```
C:/Program/Files/edb/dotnet/plugins/RawPostGis/netstandard2.0
```

次の共有ライブラリファイルが必要です。

```
EnterpriseDB.EDBClient.RawPostgis.dll
```

RawPostGIS プラグインの使用に関する詳細については、`ドキュメント`
<<https://www.nuget.org/packages/Npgsql.RawPostgis/>>`。

第 16 章

API リファレンス

API の使用については、`Npgsql` のドキュメントを参照してください
<<http://www.npgsql.org/doc/api/Npgsql.html>>`。

使用上の注意:

- API を使用する場合、Npgsql への参照を EnterpriseDB.EDBClient に置き換えます。
- クラスを参照するときは、Npgsql を EDB に置き換えます。たとえば、NpgsqlBinaryExporter クラスの代わりに EDBBinaryExporter クラスを使用します。

第 17 章

結論

EDBPostgres.NET データプロバイダーガイド

Copyright©2007-2020EnterpriseDBCorporation。全著作権所有。

EnterpriseDB®Corporation

34CrosbyDrive、Suite201、Bedford、MA01730、USA

T+17813573390F+19784671307E

info@enterprisedb.com

`[www.enterprisedb.com](https://www.enterprisedb.com/) で Web 上の EnterpriseDB にアクセスします<https://www.enterprisedb.com/>`。

- EnterpriseDB および PostgresEnterpriseManager は、EnterpriseDBCorporation の登録商標です。EDB および EDBPostgres は EnterpriseDBCorporation の商標です。Oracle は Oracle、Inc. の登録商標です。その他の商標は、各所有者の商標です。
- EDB は、EDB 製品が存在する場合、ログオン UI の設計、コーディングのベストプラクティスの確立、レビュー、および検証を行います。EDB は、追加の入力コンポーネントについても同じアプローチに従いますが、製品の性質上、制限付きの検証が可能な信頼できるユーザーが入力および送信するフリーフォーム SQL、WMI、またはその他の文字列を受け入れることが必要になる場合があります。このような場合、ユーザーが誤った入力や危険な入力をするのを防ぐことはできません。
- EDB は、認証済みの信頼できるユーザーからのフリーフォーム SQL、WMI、またはその他の潜在的に危険な入力を受け入れる製品に機能を追加する権利を留保しますが、そのようなすべての機能が最小限のリスクを提供できるように設計およびテストされることを保証します。可能、スーパーユーザーまたは同等の権限が必要です。
- EDB は、すべての潜在的な脅威を予測できること、または予測できることを保証するものではありません。したがって、私たちのプロセスでは、すべての潜在的な脆弱性が対処または検討されたことを完全に保証できません。

索引

.NET framework setup, 14
 .NET アプリケーションでの SPL ストアドプロシージャの使用, 49
 .NET アプリケーションでの参照カーソルの使用, 75
 .NET コネクタのインストールと設定, 9
 .NET コネクタの使用, 29

AdvancedServer.NET コネクタ-概要, 7
 API リファレンス, 87

configuring the .NET connector, 10

dequeue message, 62

edbaq classes, 70
 enqueue message, 62
 executing stored procedure with in, 56
 executing stored procedure with in parameters, 52
 executing stored procedure without parameters, 50

opening database connection from console application, 35
 opening database connection from windows form application, 37
 opening database connection using asp.net, 34
 out and inout parameters, 56

referencing the library files, 12

Supported platforms, 5
 Supported server versions, 5

the .net class hierarchy, 7

アドバンスドキューイングの使用, 61

スクラムの互換性, 27

セキュリティと暗号化, 27

データベースにレコードを挿入, 45
 データベースのレコードを削除する, 47
 データベースレコードの取得, 39
 データベース接続を開く, 31

パラメータ化されたクエリ, 43

プラグインの使用, 79

結論, 89
 新機能, 3
 要件の概要, 5