SYBASE®

ユーザーズ・ガイド

Adaptive Server® Enterprise ADO.NET Data Provider

1.155

[Microsoft Windows 版]

ドキュメント ID：DC00067-01-0115-02
改訂：2009 年 11 月

マニュアルの注文

マニュアルの注文を承ります。ご希望の方は、サイベース株式会社営業部または代理店までご連絡ください。マニュアルの変更は、弊社の定期的なソフトウェア・リリース時にのみ提供されます。

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目次

はじめに

対象読者

このマニュアルは、サポートされている .NET プログラミング言語を使用して Adaptive Server® Enterprise からデータをアクセスする必要のあるアプリケーション開発者を対象としています。このマニュアルを使用する方は、Microsoft ADO.NET テクノロジに精通しており、ADO.NET 仕様をコード化できる必要があります。

このマニュアルの内容

このマニュアルは、次のように構成されています。

- 「第 1 章 Adaptive Server Enterprise ADO.NET Data Provider の理解と配備」では、Adaptive Server Enterprise ADO.NET Data Provider の概要について説明します。
- 「第 2 章 サンプル・アプリケーションの使用」では、Adaptive Server ADO.NET Data Provider に付属するサンプル・プロジェクトの使用方法について説明します。
- 「第 3 章 アプリケーションの開発」では、Adaptive Server ADO.NET Data Provider を使用してアプリケーションの開発と展開を行う上で役立つヒントを紹介します。
- 「第 4 章 Adaptive Server の高度な機能」では、Adaptive Server ADO.NET Data Provider で使用できる Adaptive Server の機能について説明します。
- 「第 5 章 Adaptive Server ADO.NET Data Provider API リファレンス」では、Adaptive Server ADO.NET Data Provider API について説明します。

関連マニュアル

詳細については、これらのマニュアルを参照できます。

- 『Software Developer's Kit リリース・ノート Microsoft Windows 版』には、Adaptive Server ADO.NET Data Provider および Software Developer's Kit (SDK) に関する重要な最新情報が記載されています。
- 『Software Developer's Kit/Open Server インストール・ガイド』には SDK および Adaptive Server ADO.NET Data Provider コンポーネントのインストールについて説明します。
- Adaptive Server Enterprise の『インストール・ガイド』には、Adaptive Server のインストールについて説明します。
- 使用しているプラットフォームの Adaptive Server Enterprise の『リリース・ノート』では、既知の問題および更新の詳細について説明します。

その他の情報

Sybase® Getting Started CD、SyBooks™ CD、Sybase Product Manuals Web サイトを利用すると、製品について詳しく知ることができます。

- Getting Started CD には、PDF 形式のリリース・ノートとインストール・ガイド、SyBooks CD に含まれていないその他のマニュアルや更新情報が収録されています。この CD は製品のソフトウェアに同梱されています。Getting Started CD に収録されているマニュアルを参照または印刷するには、Adobe Acrobat Reader が必要です (CD 内のリンクを使用して Adobe の Web サイトから無料でダウンロードできます)。
- SyBooks CD には製品マニュアルが収録されています。この CD は製品のソフトウェアに同梱されています。Eclipse ベースの SyBooks ブラウザを使用すれば、使いやすい HTML 形式のマニュアルにアクセスできます。

  一部のマニュアルは PDF 形式で提供されています。これらのマニュアルは SyBooks CD の PDF ディレクトリに収録されています。PDF ファイルを開いたり印刷したりするには、Adobe Acrobat Reader が必要です。

  SyBooks をインストールして起動するまでの手順については、Getting Started CD の『SyBooks インストール・ガイド』、または SyBooks CD の『*README.txt*』ファイルを参照してください。
- Sybase Product Manuals Web サイトは、SyBooks CD のオンライン版であり、標準の Web ブラウザを使用してアクセスできます。また、製品マニュアルのほか、EBFs/Updates、Technical Documents、Case Management、Solved Cases、ニュース・グループ、Sybase Developer Network へのリンクもあります。

  Technical Library Product Manuals Web サイトにアクセスするには、Product Manuals (http://www.sybase.com/support/manuals/) にアクセスしてください。

Web 上の Sybase 製品の動作確認情報

Sybase Web サイトの技術的な資料は頻繁に更新されます。

❖ **製品認定の最新情報にアクセスする**

1. Web ブラウザで Technical Documents を指定します。(http://www.sybase.com/support/techdocs/)
2. [Partner Certification Report] をクリックします。
3. [Partner Certification Report] フィルタで製品、プラットフォーム、時間枠を指定して [Go] をクリックします。
4. [Partner Certification Report] のタイトルをクリックして、レポートを表示します。

❖ **コンポーネント認定の最新情報にアクセスする**

1 Web ブラウザで Availability and Certification Reports を指定します。(http://certification.sybase.com/)

2 [Search By Base Product] で製品ファミリとベース製品を選択するか、[Search by Platform] でプラットフォームとベース製品を選択します。

3 [Search] をクリックして、入手状況と認定レポートを表示します。

❖ **Sybase Web サイト (サポート・ページを含む) の自分専用のビューを作成する**

MySybase プロファイルを設定します。MySybase は無料サービスです。このサービスを使用すると、Sybase Web ページの表示方法を自分専用にカスタマイズできます。

1 Web ブラウザで Technical Documents を指定します。(http://www.sybase.com/support/techdocs/)

2 [MySybase] をクリックし、MySybase プロファイルを作成します。

Sybase EBF とソフトウェア・メンテナンス

❖ **EBF とソフトウェア・メンテナンスの最新情報にアクセスする**

1 Web ブラウザで Sybase Support Page を指定します。(http://www.sybase.com/support)

2 [EBFs/Maintenance] を選択します。MySybase のユーザ名とパスワードを入力します。

3 製品を選択します。

4 時間枠を指定して [Go] をクリックします。EBF/Maintenance リリースの一覧が表示されます。

鍵のアイコンは、「Technical Support Contact」として登録されていないため、一部の EBF/Maintenance リリースをダウンロードする権限がないことを示しています。未登録でも、Sybase 担当者またはサポート・コンタクトから有効な情報を得ている場合は、[Edit Roles] をクリックして、「Technical Support Contact」の役割を MySybase プロファイルに追加します。

5 EBF/Maintenance レポートを表示するには [Info] アイコンをクリックします。ソフトウェアをダウンロードするには製品の説明をクリックします。

表記規則

このマニュアルで使用されている表記規則は次のとおりです。

- クラス、コマンド名、コマンド・オプション名、メソッド、プログラム名、プログラム・フラグ、プロパティ、キーワード、関数、文、ストアド・プロシージャは次の形式で表記されます。

  ExecuteNonQuery メソッドでは、Insert、Update、または Delete 文を使用できます。

- 変数、パラメータ、ユーザが指定する語は、構文内と本文中では次のように斜体で表記されます。

  set password *new_passwd* 句では新しいパスワードを指定します。

- データベース、テーブル、カラム、データ型などのデータベース・オブジェクトの名前は、次のように表記されます。

  pubs2 オブジェクトの値。

- コマンドの構文やオプションを示す文は、次のように表記されます。

  AseDataAdapter *adapter*
  string *connectionString*
  AseCommand *selectCommand*

  コマンドの用途を示す例は、次のように表記されます。

```
AseConnection conn = new AseConnection(
   "Data Source='mango';" +
   "Port=5000;" +
   "UID='sa';" +
   "PWD='';" +
   Database='pubs2';" );
```

次の表は、構文の表記規則をまとめたものです。

表 *1:* 構文の表記規則

キー	定義
{ }	中カッコは、その中のオプションを 1 つ以上選択しなければならないことを意味する。コマンドには中カッコは入力しない。
[ ]	角カッコは、オプションを選択しても省略してもよいことを意味する。コマンドには角カッコは入力しない。
|	縦線は、中カッコまたは角カッコの中の複数のオプションのうち 1 つだけを選択できることを意味する。
,	カンマは、中カッコまたは角カッコの中のオプションをいくつでも選択できることを意味する。複数のオプションを選択する場合には、オプションをカンマで区切る。 カンマは他の構文内容で必須になることもある。
( )	このカッコはコマンドの一部として入力する。
...	省略記号 (...) は、直前の要素を必要な回数だけ繰り返し指定できることを意味する。省略記号はコマンドには入力しない。

アクセシビリティ機能

このマニュアルには、アクセシビリティを重視した HTML 版もあります。この HTML 版マニュアルは、スクリーン・リーダーで読み上げる、または画面を拡大表示するなどの方法により、その内容を理解できるよう配慮されています。

Adaptive Server Enterprise ADO.NET Data Provider マニュアルは、連邦リハビリテーション法第 508 条のアクセシビリティ規定に準拠していることがテストにより確認されています。第 508 条に準拠しているマニュアルは通常、World Wide Web Consortium (W3C) の Web サイト用ガイドラインなど、米国以外のアクセシビリティ・ガイドラインにも準拠しています。

注意 アクセシビリティ・ツールを効率的に使用するには、設定が必要な場合もあります。一部のスクリーン・リーダーは、テキストの大文字と小文字を区別して発音します。たとえば、すべて大文字のテキスト (ALL UPPERCASE TEXT など) はイニシャルで発音し、大文字と小文字の混在したテキスト (Mixed Case Text など) は単語として発音します。構文規則を発音するようにツールを設定すると便利かもしれません。詳細については、ツールのマニュアルを参照してください。

Sybase のアクセシビリティに対する取り組みについては、Sybase Accessibility (http://www.sybase.com/accessibility) を参照してください。Sybase Accessibility サイトには、第 508 条と W3C 標準に関する情報へのリンクもあります。

不明な点があるときは

Sybase ソフトウェアがインストールされているサイトには、Sybase 製品の保守契約を結んでいるサポート・センタとの連絡担当の方 (コンタクト・パーソン) を決めてあります。マニュアルだけでは解決できない問題があった場合には、担当の方を通して Sybase のサポート・センタまでご連絡ください。

第 1 章

Adaptive Server Enterprise ADO.NET Data Provider の理解と配備

この章では、Adaptive Server Enterprise ADO.NET Data Provider の概要について説明します。

Adaptive Server ADO.NET Data Provider とは

Adaptive Server Enterprise ADO.NET Data Provider は、Sybase Adaptive Server データベース用の ADO.NET プロバイダです。Adaptive Server バージョン 12.5.x、15.0.x、15.5.x と Adaptive Server CE の全バージョンでサポートされています。

Adaptive Server ADO.NET Data Provider を使用すると、C#、Visual Basic .NET、マネージ拡張を備えた C++、J# など、.NET でサポートされる任意の言語を使用して Adaptive Server 内のデータにアクセスできます。.NET 共通言語ランタイム (CLR: Common Language Runtime) アセンブリであり、ADO.NET インタフェース全般の機能を提供する、一連の必要なクラスをすべて含んだクラス・ライブラリに相当します。すべてのクラスはマネージ・コードで、任意のマネージ・クライアント・コードからアクセスできます。このような各言語間の通信は、Microsoft .NET Framework によって実現します。

Adaptive Server ADO.NET Data Provider を使用する主な利点としては、次のものが挙げられます。

- Adaptive Server ADO.NET Data Provider は、OLE DB プロバイダよりも高速である。
- .NET 環境において、Adaptive Server ADO.NET Data Provider は Adaptive Server に対するネイティブ・アクセスを提供する。サポートされるその他のプロバイダとは異なり、Adaptive Server と直接通信できるため、ブリッジ技術を必要としない。

Adaptive Server ADO.NET Data Provider の配備

以降の各項では、Adaptive Server ADO.NET Data Provider を配備するための要件について説明します。サポートされているプラットフォームのリストについては、Sybase platform certifications page を参照してください。(http://certification.sybase.com/ucr/search.do)

システム稼働条件

Adaptive Server ADO.NET Data Provider を使用するには、コンピュータに次のものがインストールされている必要があります。

- **開発時 ー** NET Framework SDK 1.1 と、Visual Studio .NET 2003、または C# などの .NET 言語コンパイラ
- **配備時 ー** NET Framework 1.1

必要なファイル

Adaptive Server ADO.NET Data Provider は、以下のファイルで構成されています。

- *Sybase.Data.AseClient.dll* は、クライアント・コードで参照されるプロバイダ・アセンブリです。
- *sbgse2.dll*、*sybcsi_certicom_fips26.dll*、*sybcsi_core26.dll*、*sybcsi_profiler26.dll* には、SSL をサポートするためのコードが含まれます。
- *sybdrvado115.dll* には、ユーティリティ・コードが含まれます。
- *sybdrvkrb.dll* には、Kerberos 認証のためのコードが含まれます。

グローバル・アセンブリ・キャッシュへの Adaptive Server ADO.NET Data Provider アセンブリの配備

多くの場合、1 台のコンピュータ上の複数のアプリケーションで、Adaptive Server ADO.NET Data Provider アセンブリが共有されています。この結果、アセンブリのコピーが重複して存在することになり、互換性やバージョン管理の問題が生じます。このような状況を回避するため、Adaptive Server ADO.NET Data Provider アセンブリをグローバル・アセンブリ・キャッシュ (GAC: Global Assembly Cache) に配備することをおすすめします。グローバル・アセンブリ・キャッシュはコンピュータ全体を対象範囲とするキャッシュであり、同じコンピュータ上の複数のアプリケーションによって共有されているアセンブリを格納および管理します。このような配備を行うことができない場合は、プロバイダを使用するアプリケーションが実行されるすべてのディレクトリに、Adaptive Server ADO.NET Data Provider アセンブリのコピーをインストールしてください。

システム上に .NET Framework SDK 1.1 がインストールされていることが検出されると、Adaptive Server ADO.NET Data Provider のインストール・プログラムによって自動的にアセンブリが GAC に配備されます。SDK がインストールされていない場合、またはインストール・プログラムを使用していない場合は、アセンブリを手動で配備する必要があります。これを行うには、.NET Framework 構成ツールを使用します。

❖ **.NET Framework 構成ツールを使用してアセンブリを配備する**

1. .NET Framework 構成ツールを起動します。構成ツールの起動方法については、各オペレーティング・システムの Microsoft のマニュアルを参照してください。
2. 左側のツリー・ビューで、[アセンブリ キャッシュ] を選択します。
3. パネルの [アセンブリ キャッシュにアセンブリを追加する] リンクをクリックします。
4. [アセンブリの追加] ダイアログ・ボックスで、インストール・ディレクトリにある Adaptive Server ADO.NET Data Provider アセンブリを検索して、[開く] をクリックします。デフォルトのインストール・ディレクトリは、*C:¥Sybase¥DataAccess¥ADONET¥dll* です。

   これで、Adaptive Server ADO.NET Data Provider アセンブリが GAC に配備されます。キャッシュ内のアセンブリのリストを確認するには、パネルの [アセンブリ キャッシュのアセンブリ一覧の表示] リンクを選択します。

GAC からのアセンブリの削除

GAC からアセンブリを削除するには、.NET Framework 構成ツールを使用します。

❖ **.NET Framework 構成ツールを使用してアセンブリを削除する**

1. .NET Framework 構成ツールを起動します。構成ツールの起動方法については、各オペレーティング・システムの Microsoft のマニュアルを参照してください。
2. 左側のツリー・ビューで、[アセンブリ キャッシュ] を選択します。
3. パネルの [アセンブリ キャッシュのアセンブリ一覧の表示] リンクをクリックします。
4. アセンブリ名のリストで、*Sybase.Data.AseClient* を探します。システムに複数のバージョンが配備されている場合は、このアセンブリのエントリがバージョンに対応して複数表示される場合もあります。
5. 削除するアセンブリを 1 つ以上選択します。右クリックして [削除] を選択します。[はい] をクリックして操作を確定します。

6 削除したバージョンに対応する発行者ポリシー・ファイルがないか確認し、これらのファイルも削除します。

注意 GAC には、他のアセンブリから特定のアセンブリへの参照も格納されます。この場合、これらの参照が削除されるまで、この参照先のアセンブリは削除できません。これらの参照は、削除コマンドの一部として強制的に削除できます。システムによっては、ユーティリティがアセンブリの削除に失敗して、Windows インストーラで保留中の参照に関するエラーが発生することがあります。これは、レジストリに値が残っているために発生するものです。この問題が発生した場合は、Microsoft のサポートに連絡して解決策を入手してください。

Adaptive Server ADO.NET Data Provider を使用するアプリケーションの配備

以下の手順では、アプリケーションを配備する方法について説明します。

❖ インストール・プログラムと GAC を使用してアプリケーションを配備する

1 エンド・ユーザのコンピュータのインストール・プログラムを使用して Adaptive Server ADO.NET Data Provider をインストールします。

2 コンピュータに .NET Framework SDK 1.1 がインストールされていない場合は、手動でプロバイダ・アセンブリを GAC に配備します。

3 *exe* や *dll* などのアプリケーション固有ファイルを、システムのアプリケーション固有フォルダにコピーします。

❖ GAC を使用してアプリケーションを配備する

1 Adaptive Server ADO.NET Data Provider を構成する *dll* ファイルを、ターゲット・コンピュータの *C:¥Sybase¥DataAccess¥ADONET¥dll* などのディレクトリにコピーします。

2 このディレクトリをシステム・パスに追加します。

3 プロバイダ・アセンブリを GAC に配備します。詳細については、「グローバル・アセンブリ・キャッシュへの Adaptive Server ADO.NET Data Provider アセンブリの配備」(2 ページ) を参照してください。

4 *exe* や *dll* などのアプリケーション固有ファイルを、システムのアプリケーション固有フォルダにコピーします。

5 アプリケーションを実行します。

❖ **GAC を使わずにアプリケーションを配備する**

1　ターゲット・システムで、*exe* や *dll* などのアプリケーション固有ファイルに加えて、Adaptive Server ADO.NET Data Provider を構成する *dll* ファイルもアプリケーション固有フォルダにコピーします。

2　アプリケーションを実行します。

ADO.NET Data Provider の新しいバージョンへの更新

Adaptive Server ADO.NET Data Provider の更新は、EBF/ESD またはメンテナンス・リリースを通じて入手します。この項では、Data Provider を新しいバージョンに更新する場合の問題点について説明します。更新に関する Microsoft .NET コンセプトの詳細については、Microsoft Developer Network (http://msdn.microsoft.com) で公開されている『.NET Framework Deployment Guide』と『.NET Framework Developer's Guide』を参照してください。

新しいバージョンの Adaptive Server Data Provider にアプリケーションを移行するには、次のいずれかを実行してください。

- アプリケーション設定ファイルを作成して、新しいバージョンの Adaptive Server ADO.NET Data Provider を使用するようにアプリケーションをリダイレクトします。「アプリケーション設定ファイルの使用」(6 ページ) を参照してください。
- 新しいバージョンの Adaptive Server ADO.NET Data Provider に対してアプリケーションを再度ビルドし、配備します。Sybase では、この手順を選択することをおすすめします。

CLR のリダイレクト

.NET 共通言語ランタイム (CLR) は、アプリケーション・プログラムの実行時、Data Provider などのアセンブリに対する参照を見つけてバインドします。CLR はデフォルトで、アプリケーションの構築に使用された同じバージョンのアセンブリに対する参照をバインドしようとします。そのため、配備しただけで更新バージョンのアセンブリが自動的に使用されるわけではありません。新しいバージョンのアセンブリを使用してアプリケーションを再構築するか、新しいバージョンを使用するように設定ファイルで CLR をリダイレクトする必要があります。

一般的に、Data Provider の EBF/ESD リリースのリリース・レベル (メジャー／マイナー) が同じであれば、前のリリースとの間にバイナリの互換性があります。このような更新ではアプリケーションを再構築をしないことも可能です。Data Provider の更新ごとにアプリケーションの再構築や再配備を行う代わりに、アプリケーション設定や発行者ポリシー・ファイルを使用することもできます。Sybase では通常、ESD/EBF リリースに、適切なリダイレクトを設定した発行者ポリシー・ファイルを収録しています。下位互換性の問題の詳細については、ESD/EDF のドキュメントを参照してください。

アプリケーション設定ファイルの使用

アプリケーション設定ファイルを使用すると、CLR をリダイレクトして、呼び出し側のアプリケーションのマニフェストに格納されているアセンブリとバージョンの異なるアセンブリをロードできます。

次の例は、以前の Data Provider 1.0.x で構築されたアプリケーションで Data Provider 1.0.159 を使用するように CLR をリダイレクトする方法を示しています。

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="Sybase.Data.AseClient"
                          publicKeyToken="26e0f1529304f4a7"
                          culture="neutral" />
        <bindingRedirect oldVersion="1.0.0.0-1.0.158.65535"
                         newVersion="1.0.159.0"/>
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

設定ファイル・スキーマの詳細については、MSDN Library を参照してください。(http://msdn2.microsoft.com/en-us/default.aspx)

注意 アプリケーションごとに独自の設定ファイルが必要です。

発行者ポリシー・ファイルの使用

アセンブリの発行元は、共有アセンブリの更新とともに発行者ポリシー・ファイルを配布できます。このファイルによって、古いバージョンのアセンブリに対するすべての参照が、新しくインストールされたバージョンにリダイレクトされます。アプリケーション設定ファイルとは異なり、発行者ポリシー・ファイルを機能させるには、グローバル・アセンブリ・キャッシュ (GAC) に配備する必要があります。

発行者ポリシー・ファイルの設定は、アプリケーションまたはアプリケーション設定ファイルのバージョン情報よりも優先されます。ただし、「セーフ・モード」を強制し、特定のアプリケーションで発行者ポリシー・ファイルを無視するように設定することもできます。セーフ・モードを使用するようにアプリケーションを設定する方法については、MSDN ライブラリを参照してください。

一般的に、Adaptive Server ADO.NET Data Provider の更新には、最後にインストールされたバージョンの Data Provider アセンブリにアプリケーションをリダイレクトする発行者ポリシー・ファイルが組み込まれています。インストール・プログラムによりシステム上に .NET Framework SDK 1.1 がインストールされていることが検出されると、新しいプロバイダ・アセンブリと発行者ポリシー・ファイルも GAC に配備されます。

Data Provider の更新の配備

以降の各項では、Data Provider の更新の配備に関連する問題点について説明します。

Data Provider を GAC に配備する

更新された Data Provider アセンブリとポリシー・アセンブリが GAC に配備されると、システム上のすべてのアプリケーションが自動的にこの Provider の使用を開始します。

更新された Data Provider を使用しないように特定のアプリケーションを除外する

更新された Data Provider を使用しないように特定のアプリケーションを除外する場合は、そのアプリケーションのアプリケーション設定ファイルを、発行者ポリシー・ファイルを無効にするセーフ・モードを強制するように設定します。

コア・ファイルのロケーション

Data Provider は、Sybase.Data.AseClient.dll と sybdrvado115.dll という 2 つのコア・ファイルで構成されます。GAC には Sybase.Data.AseClient.dll の複数のバージョンをインストールできます。*sybdrvado115.dll* は GAC にインストールされるのではなく、システムの PATH を使用して実行時に検索されます。このファイルは、Data Provider のインストール・ディレクトリにインストールされます。Sybase は、更新リリースでこの DLL の名前やバージョン文字列を変更する場合があります。たとえば、このファイルの名前はリリース 1.0 では *aseado.dll* でしたが、リリース 1.1 では *sybdrvado115.dll* と呼ばれています。このような更新をインストールするときは、このファイルを使用する Data Provider のバージョンが GAC から削除されるまで、古いバージョンのファイルを削除しないでください。削除すると、古いバージョンのプロバイダを使用するアプリケーションを実行できなくなります。

GAC にない場合の Data Provider の配備

コンピュータの GAC に Data Provider アセンブリがインストールされていない場合、Data Provider の構成要素であるファイルをアプリケーション・フォルダにコピーする必要があります。

更新されたバージョンの Data Provider をアプリケーションで使用するには、次のいずれかを実行します。

- 適切な redirect を使用してアプリケーション設定ファイルを作成する。
- 新しいバージョンの Data Provider に合わせてアプリケーションを再構築する。
- 発行者ポリシー・ファイルのみを GAC に配備する。これにより、アプリケーションで特に除外されていない限り、コンピュータ上の Data Provider に対するすべての参照で、発行者ポリシー・ファイルの redirect が使用されます。

サンプル・プロジェクトの実行

Adaptive Server ADO.NET Data Provider には次の 3 つのサンプル・プロジェクトが組み込まれています。

- **Simple** – データベースへ接続し、クエリを実行して、返された resultsets を読み込む方法を示すサンプル・プログラム。
- **Table Viewer** – AseDataAdapter オブジェクトを使用して結果を DataGrid コントロールにバインドする方法を示すサンプル・プログラム。
- **Advanced** – 入力、出力、および入出力パラメータとともにストアド・プロシージャを呼び出す方法を示すサンプル・プログラム。ストアド・プロシージャの戻り値を読み取り、パラメータを渡すための 2 つのサポートされたメカニズムと、Data Provider のトレース機能を使用します。

Simple サンプルと Table Viewer サンプルを説明するチュートリアルについては、「第 2 章 サンプル・アプリケーションの使用」を参照してください。

ただし、デフォルトでは、Adaptive Server ADO.NET Data Provider サンプルを実行するのに必要な pubs2 データベースは Adaptive Server にインストールされません。pub2 データベースをインストールする方法については、Adaptive Server Enterprise の『インストール・ガイド』を参照してください。

第 2 章

サンプル・アプリケーションの使用

この章では、Adaptive Server ADO.NET Data Provider に付属するサンプル・プロジェクトの使用方法について説明します。

注意 サンプル・プログラムを実行するには、pubs2 サンプル・データベースがインストールされた Adaptive Server にアクセスする必要があります。また、システムに Visual Studio .NET 2003 または .NET Framework 1.1 がインストールされている必要があります。

サンプル・プログラムは、Adaptive Server ADO.NET Data Provider インストール・ディレクトリの次のディレクトリにあります。

- *Samples¥CSharp*。C# プログラミング言語で作成された 3 つのサンプルがあります。
- *Samples¥VB.NET*。Visual Basic .NET プログラミング言語で作成された 3 つのサンプルがあります。

デフォルトのインストール・ディレクトリは、*C:¥Sybase¥DataAccess¥ADONET¥dll* です。

チュートリアル：Simple コード・サンプルの使用

Simple プロジェクトでは、次の機能について説明します。

- データベースへの接続
- AseCommand オブジェクトを使用したクエリの実行
- AseDataReader オブジェクトの使用
- 基本的なエラー処理

サンプルの動作の詳細については、「Simple サンプル・プロジェクトの理解」(13 ページ) を参照してください。

❖ Visual Studio .NET での Simple コード・サンプルの実行

1 Visual Studio .NET を起動します。

2 [ファイル] － [開く] － [プロジェクト] を選択します。

3 サンプル・プロジェクトを指定します。

C# の場合は、<インストール・ディレクトリ>*¥Samples¥CSharp¥Simple* を参照して、*Simple.csproj* を開きます。

Visual Basic .NET の場合は、<インストール・ディレクトリ>*¥Samples¥VB.NET¥Simple* を参照して、*Simple.vbproj* を開きます。

4 インストール・プログラムを使用して Adaptive Server ADO.NET Data Provider をインストールしている場合は、手順 7 に進みます。

5 インストール・プログラムを使用しなかった場合は、プロジェクトの Adaptive Server ADO.NET Data Provider に対する参照を修正する必要があります。これには、まず既存の参照を削除します。

a [ソリューション エクスプローラ] ウィンドウで、Simple プロジェクトが展開されていることを確認します。

b [参照設定] フォルダを展開します。

c *Sybase.AseClient.Data.dll* を右クリックして、[削除] を選択します。

6 Adaptive Server ADO.NET Data Provider アセンブリに対する参照を追加します。

詳細については、「Data Provider アセンブリへの参照の追加」(29 ページ) を参照してください。

7 Simple サンプルを実行するには、[デバッグ] － [デバッグなしで開始] を選択するか、[Ctrl] キーを押しながら、[F5] キーを押します。

[AseSample] ダイアログ・ボックスが表示されます。

8 [AseSample] ダイアログ・ボックスで、サンプルの pubs2 データベースのある Adaptive Server への接続情報を指定して、[接続] をクリックします。

アプリケーションがサンプルの pubs2 データベースに接続し、ダイアログ・ボックスに各作家の姓が表示されます。

9 ウィンドウの右上角にある [X] をクリックすると、アプリケーションが終了し、pubs2 データベースとの接続が切断されます。

これでアプリケーションを実行できました。次の項では、アプリケーション・コードについて説明します。

❖ Visual Studio を使用しない Simple サンプル・プロジェクトの実行

1. DOS プロンプトを開き、<インストール・ディレクトリ>*¥Samples* にある適切なサンプル・ディレクトリに移動します。
2. .NET Framework 1.1 バイナリのあるディレクトリをシステム・パスに追加します。
3. Adaptive Server ADO.NET Data Provider インストール・ディレクトリにある *dll* ディレクトリが、システム・パスと LIB 環境変数に含まれていることを確認します。
4. 提供されているビルド・スクリプト *build.bat* を使用してサンプル・プログラムをコンパイルします。
5. プログラムを実行するには、次のように入力します。

   ```
   simple.exe
   ```

   [AseSample] ダイアログ・ボックスが表示されます。
6. [AseSample] ダイアログ・ボックスで、サンプルの pubs2 データベースのある Adaptive Server への接続情報を指定して、[接続] をクリックします。

   アプリケーションがサンプルの pubs2 データベースに接続し、ダイアログ・ボックスに各作家の姓が表示されます。
7. ウィンドウの右上角にある [X] をクリックすると、アプリケーションが終了し、pubs2 データベースとの接続が切断されます。

Simple サンプル・プロジェクトの理解

この項では、Adaptive Server サンプル・データベース pubs2 を使用する Simple コード・サンプルを利用して、Adaptive Server ADO.NET Data Provider の一部の主要機能について説明します。pub2 データベースをインストールする方法については、Adaptive Server Enterprise の『インストール・ガイド』を参照してください。

この項では、コードの一部について説明します。コード全体を参照する場合は、サンプル・プロジェクトを開いてください。

C# の場合：

```
<インストール・ディレクトリ>¥Samples¥CSharp¥Simple¥Simple.csproj
```

Visual Basic .NET の場合：

```
<インストール・ディレクトリ>¥Samples¥VB.NET¥Simple¥Simple.vbproj
```

インポートの宣言

プログラムの始めに、Adaptive Server ADO.NET Data Provider 情報をインポートする import 文を宣言します。

C# の場合：

```
using Sybase.Data.AseClient;
```

Visual Basic .NET の場合：

```
Imports Sybase.Data.AseClient
```

データベースへの接続

btnConnect_Click メソッドは、new AseConnection という接続オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection(
   "Data Source='" + host +
      "';Port='" + port +
      "';UID='" + user +
      "';PWD='" + pass +
      "';Database='pubs2';" );
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection( _
     "Data Source='" + host + _
     "';Port='" + port + _
     "';UID='" + user + _
     "';PWD='" + pass + _
     "';Database='pubs2';")
```

AseConnection オブジェクトは、接続文字列を使用してサンプル・データベースに接続します。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

AseConnection オブジェクトの詳細については、「AseConnection クラス」(113 ページ) を参照してください。

クエリの実行

次のコードは、Command オブジェクト (AseCommand) によって SQL 文を定義して実行します。その後、DataReader オブジェクト (AseDataReader) を返します。

C# の場合：

```
AseCommand cmd = new AseCommand( "select au_lname from
     authors", conn );
AseDataReader reader = cmd.ExecuteReader();
```

Visual Basic .NET の場合：

```
Dim cmd As New AseCommand( _
     "select au_lname from authors", conn)
Dim reader As AseDataReader  = cmd.ExecuteReader()
```

Command オブジェクトの詳細については、「AseCommand クラス」(101 ページ) を参照してください。

結果の表示

次のコードは、AseDataReader オブジェクトに保持されているローをループして ListBox コントロールに追加します。DataReader は、`GetString( 0 )` を使用して、ローの最初の値を取得します。

Read メソッドが呼び出されるたびに、DataReader は結果セットから別のローを取得して返します。読み込まれたそれぞれのローについて、新しい項目が ListBox に追加されます。

C# の場合：

```
listAuthors.BeginUpdate();
while( reader.Read() ) {
     listAuthors.Items.Add( reader.GetString( 0 ) );
}
listAuthors.EndUpdate();
```

Visual Basic .NET の場合：

```
listAuthors.BeginUpdate()
While reader.Read()
     listAuthors.Items.Add(reader.GetString(0))
End While
listAuthors.EndUpdate()
```

AseDataReader オブジェクトの詳細については、「AseDataReader クラス」(131 ページ) を参照してください。

接続の終了

メソッドの最後にある次のコードで、読み込みオブジェクトと接続オブジェクトをクローズします。

C# の場合：

```
reader.Close();
conn.Close();
```

Visual Basic .NET の場合：

```
reader.Close()
conn.Close()
```

エラー処理

実行時に発生したエラーや Adaptive Server ADO.NET Data Provider オブジェクトのエラーはすべて、メッセージ・ボックスに表示されます。次のコードは、エラーを検出してメッセージを表示します。

C# の場合：

```
catch( AseException ex ) {
     MessageBox.Show( ex.Message );
}
```

Visual Basic .NET の場合：

```
Catch ex As AseException
     MessageBox.Show(ex.Message)
End Try
```

AseException オブジェクトの詳細については、「AseException クラス」(150 ページ) を参照してください。

チュートリアル：Table Viewer コード・サンプルの使用

このチュートリアルは、Adaptive Server ADO.NET Data Provider に付属している Table Viewer プロジェクトに基づいています。アプリケーション全体は、Adaptive Server ADO.NET Data Provider のインストール・ディレクトリにあります。

C# の場合：

```
<インストール・ディレクトリ>
¥Samples¥CSharp¥TableViewer¥TableViewer.csproj
```

Visual Basic .NET の場合：

```
<インストール・ディレクトリ>
¥Samples¥VB.NET¥TableViewer¥TableViewer.vbproj
```

Table Viewer プロジェクトは、Simple プロジェクトより複雑です。このサンプルでは、次の機能について説明します。

- データベースへの接続
- AseDataAdapter オブジェクトの使用
- 高度なエラー処理と結果チェック

サンプルの動作の詳細については、「Table Viewer サンプル・プロジェクトの理解」(18 ページ) を参照してください。

❖ Visual Studio .NET での Table Viewer コード・サンプルの実行

1. Visual Studio .NET を起動します。
2. [ファイル] － [開く] － [プロジェクト] を選択します。
3. Adaptive Server ADO.NET Data Provider インストール・ディレクトリにある *Samples* ディレクトリを指定します。*CSharp* または *VB.NET* ディレクトリに移動して、Table Viewer プロジェクトを開きます。
4. インストール・プログラムを使用して Adaptive Server ADO.NET Data Provider をインストールしている場合は、手順 7 に進みます。
5. インストール・プログラムを使用しなかった場合は、プロジェクトの Adaptive Server ADO.NET Data Provider に対する参照を修正する必要があります。これには、まず既存の参照を削除します。
   a. [ソリューション エクスプローラ] ウィンドウで、Simple プロジェクトが展開されていることを確認します。
   b. [参照設定] フォルダを展開します。
   c. *Sybase.AseClient.Data.dll* を右クリックして、[削除] を選択します。
6. Adaptive Server ADO.NET Data Provider アセンブリに対する参照を追加します。

   詳細については、「Data Provider アセンブリへの参照の追加」(29 ページ)を参照してください。
7. Table Viewer サンプルを実行するには、[デバッグ] － [デバッグなしで開始] を選択するか、[Ctrl] キーを押しながら、[F5] キーを押します。
8. [Table Viewer] ダイアログ・ボックスで、pubs2 サンプル・データベースがインストールされた Adaptive Server への接続情報を指定します。[接続] をクリックします。

   アプリケーションが Adaptive Server pubs2 サンプル・データベースに接続します。
9. [Table Viewer] ダイアログ・ボックスで [実行] をクリックします。

   アプリケーションは、サンプル・データベースのauthors テーブルからデータを取得して、クエリの結果を Results DataList に入力します。

   このアプリケーションで、別の SQL 文を実行することもできます。[SQL 文] ペインに SQL 文を入力して、[実行] をクリックします。
10. ウィンドウの右上角にある [X] をクリックすると、アプリケーションが終了し、サンプル・データベースとの接続が切断されます。

❖ **Visual Studio を使用しない Table Viewer サンプル・プロジェクトの実行**

1 DOS プロンプトを開いて、<インストール・ディレクトリ>*¥Samples* にある適切なサンプル・ディレクトリに移動します。

2 .NET Framework 1.1 バイナリのあるディレクトリをシステム・パスに追加します。

3 Adaptive Server ADO.NET Data Provider インストール・ディレクトリにある *dll* ディレクトリが、システム・パスと LIB 環境変数に含まれていることを確認します。

4 提供されているビルド・スクリプト *build.bat* を使用してサンプル・プログラムをコンパイルします。

5 プログラムを実行するには、次のように入力します。

```
tableviewer.exe
```

6 [Table Viewer] ダイアログ・ボックスで、pubs2 サンプル・データベースがインストールされた Adaptive Server への接続情報を指定します。

[接続] をクリックします。

アプリケーションが Adaptive Server pubs2 サンプル・データベースに接続します。

7 [Table Viewer] ダイアログ・ボックスで [実行] をクリックします。

アプリケーションは、サンプル・データベースのauthors テーブルからデータを取得して、クエリの結果を Results DataList に入力します。

このアプリケーションで、別の SQL 文を実行することもできます。[SQL 文] ペインに SQL 文を入力して、[実行] をクリックします。

8 ウィンドウの右上角にある [X] をクリックすると、アプリケーションが終了し、サンプル・データベースとの接続が切断されます。

これでアプリケーションを実行できました。次の項では、アプリケーション・コードについて説明します。

Table Viewer サンプル・プロジェクトの理解

この項では、Table Viewer コード・サンプルを利用して、Adaptive Server ADO.NET Data Provider の一部の主要機能について説明します。Table Viewer プロジェクトは、Adaptive Server サンプル・データベース pubs2 を使用します。このデータベースは、Adaptive Server インストール・ディレクトリにあるスクリプトからインストールできます。

この項では、数行ずつコードを説明します。コード全体を参照するには、Adaptive Server インストール・ディレクトリのサンプル・プロジェクトを開きます。

C# の場合：

```
<インストール・ディレクトリ> ¥Samples¥CSharp¥TableViewer¥
TableViewer.csproj
```

Visual Basic .NET の場合：

```
<インストール・ディレクトリ>¥Samples¥VB.NET¥TableViewer
¥TableViewer.vbproj
```

インポートの宣言

プログラムの始めに、Adaptive Server ADO.NET Data Provider 情報をインポートする import 文を宣言します。

C# の場合：

```
using Sybase.Data.AseClient;
```

Visual Basic .NET の場合：

```
Imports Sybase.Data.AseClient
```

インスタンス変数の宣言

AseConnection クラスを使用して、AseConnection 型のインスタンス変数を宣言します。この接続は、データベースへの初期接続と、[実行] をクリックしてデータベースから結果セットを取得するときに使用されます。

C# の場合：

```
private AseConnection
     _conn;
```

Visual Basic .NET の場合：

```
Private _conn As AseConnection
```

詳細については、「AseConnection コンストラクタ」(114 ページ) を参照してください。

データベースへの接続

次のコードは、Connection String フィールドにデフォルトで表示される接続文字列のデフォルト値を設定します。

C# の場合：

```
txtConnectString.Text = "Data Source='" +
   System.Net.Dns.GetHostName() +
   "';Port='5000';UID='sa';PWD='';Database='pubs2';";
```

Visual Basic .NET の場合：

```
txtConnectString.Text = "Data Source='" + _
   System.Net.Dns.GetHostName() + _
   "';Port='5000';UID='sa';PWD='';Database='pubs2';"
```

Connection オブジェクトは、接続文字列を使用してサンプル・データベースに接続します。

C# の場合：

```
_conn = new AseConnection( txtConnectString.Text );
_conn.Open();
```

Visual Basic .NET の場合：

```
_conn = New AseConnection(txtConnectString.Text)
_conn.Open()
```

詳細については、「AseConnection クラス」(113 ページ) を参照してください。

クエリの定義

次のコードは、SQL Statement フィールドに表示されるデフォルトのクエリを定義します。

C# の場合：

```
this.txtSQLStatement.Text = "SELECT * FROM authors";
```

Visual Basic .NET の場合：

```
Me.txtSQLStatement.Text = "SELECT * FROM authors"
```

結果の表示

アプリケーションは、Connection オブジェクトが初期化されているかどうかを確認してから結果セットをフェッチします。初期化されている場合は、接続ステータスがオープンであることを確認します。

C# の場合：

```
if( _conn == null || _conn.State != ConnectionState.Open )
{
     MessageBox.Show( "Connect to a database first.", "Not connected" );
     return;
}
```

Visual Basic .NET の場合：

```
If (_conn Is Nothing) OrElse (_conn.State <> ConnectionState.Open) Then
     MessageBox.Show("Connect to a database first.", "Not connected")
     Return
End If
```

データベースに接続されると、次のコードは DataAdapter オブジェクト (AseDataAdapter) を使用して SQL 文を実行します。新しい DataSet オブジェクトが作成されて、DataAdapter オブジェクトの結果が入力されます。最後に、DataSet の内容がウィンドウの DataGrid コントロールにバインドされます。

C# の場合：

```
using(AseCommand cmd = new AseCommand( txtSQLStatement.Text.Trim(), _conn ))
{
     using(AseDataAdapter   da = new AseDataAdapter(cmd))
     {
          DataSet ds = new DataSet();
          da.Fill(ds, "Table");

          dgResults.DataSource = ds.Tables["Table"];
     }
}
```

Visual Basic .NET の場合：

```
Dim cmd As New AseCommand( _
   txtSQLStatement.Text.Trim(), _conn)
Dim da As New AseDataAdapter(cmd)
Dim ds As New DataSet
da.Fill(ds, "Table")
dgResults.DataSource = ds.Tables("Table")
```

グローバル変数を使用して接続を宣言しているため、SQL 文の実行には以前にオープンした接続が再使用されます。

DataAdapter オブジェクトの詳細については、「AseDataAdapter クラス」(124 ページ) を参照してください。

エラー処理

アプリケーションがデータベースへの接続を試行しているときにエラーが発生した場合は、次のコードによってエラーが検出され、メッセージが表示されます。

C# の場合：

```
catch( AseException ex )
{
   MessageBox.Show( ex.Source + " :" + ex.Message +
      " ("+ ex.ToString() + ")",
      "Failed to connect" );
}
```

Visual Basic .NET の場合：

```
Catch ex As AseException
   MessageBox.Show(ex.Source + " :" + ex.Message + _
   "(" + ex.ToString() + ")" + _
   "Failed to connect")
End Try
```

チュートリアル：Advanced コード・サンプルの使用

このチュートリアルは、Adaptive Server ADO.NET Data Provider に付属している Advanced プロジェクトに基づいています。アプリケーション全体は、Adaptive Server ADO.NET Data Provider のインストール・ディレクトリにあります。

C# の場合：

```
<インストール・ディレクトリ>
¥Samples¥CSharp¥Advanced¥Advanced.csproj
```

Visual Basic .NET の場合：

```
<インストール・ディレクトリ>
¥Samples¥VB.NET¥Advanced¥Advanced.vbproj
```

Advanced プロジェクトでは、次の機能について説明します。

- データベースへの接続
- Adaptive Server ADO.NET Data Provider に対する ADO.NET の呼び出しをトレースするトレース・イベント機能の使用

  トレース・イベント機能を使用すると、実行した ADO.NET の呼び出しをすべてログに記録して、Sybase 製品の保守契約を結んでいるサポート・センタに送る詳細情報の収集やトラブルシューティングに利用できます。
- 名前付きパラメータ (“@param”) の使用
- 次のようなパラメータ・マーカ (“?”) の使用：`{? = call sp_hello(?, ?, ?)}`
- 入力パラメータ、入力／出力パラメータ、出力パラメータ、戻り値を使用したストアド・プロシージャの呼び出し。Adaptive Server では次の 2 つの方法でストアド・プロシージャを呼び出すことができます。
  - CommandText としてストアド・プロシージャ名を使用して、AseCommand.CommandType を CommandType.StoredProcedure に設定する。
  - 呼び出し構文を使用する。この構文は、ODBC および JDBC プログラムと互換性があります。

❖ Visual Studio .NET での Advanced コード・サンプルの実行

1　Visual Studio .NET を起動します。

2　[ファイル] － [開く] － [プロジェクト] を選択します。

3　Adaptive Server ADO.NET Data Provider インストール・ディレクトリにある *Samples* ディレクトリを指定します。*CSharp* または *VB.NET* ディレクトリに移動して、Advanced プロジェクトを開きます。

4　インストール・プログラムを使用して Adaptive Server ADO.NET Data Provider をインストールしている場合は、手順 7 に進みます。

5　インストール・プログラムを使用しなかった場合は、プロジェクトの Adaptive Server ADO.NET Data Provider に対する参照を修正する必要があります。これには、まず既存の参照を削除します。

   a　[ソリューション エクスプローラ] ウィンドウで、Simple プロジェクトが展開されていることを確認します。

   b　[参照設定] フォルダを展開します。

   c　*Sybase.AseClient.Data.dll* を右クリックして、[削除] を選択します。

6　Adaptive Server ADO.NET Data Provider アセンブリに対する参照を追加します。

7　[デバッグ] － [デバッグなしで開始] を選択して、Advanced プロジェクトを実行します。

   [Form1] ダイアログ・ボックスが表示されます。

8　[Form1] ダイアログ・ボックスで [接続] をクリックします。

   アプリケーションが Adaptive Server サンプル・データベースに接続します。

9　[Form1] ダイアログボックスで [実行] をクリックします。

   アプリケーションはストアド・プロシージャを実行し、入力／出力パラメータ、出力パラメータ、戻り値を返します。

10　ウィンドウの右上角にある [X] をクリックすると、アプリケーションが終了し、サンプル・データベースとの接続が切断されます。

これでアプリケーションを実行できました。次の項では、アプリケーションについて説明します。

❖ Visual Studio を使用しない Advanced サンプル・プロジェクトの実行

1 DOS プロンプトを開いて、<インストール・ディレクトリ>*¥Samples* ディレクトリにある適切なサンプル・ディレクトリに移動します。

2 .NET Framework 1.1 バイナリのあるディレクトリをシステム・パスに追加します。

3 Adaptive Server ADO.NET Data Provider インストール・ディレクトリにある *dll* ディレクトリが、システム・パスと LIB 環境変数に含まれていることを確認します。

4 提供されているビルド・スクリプト *build.bat* を使用してサンプル・プログラムをコンパイルします。

5 プログラムを実行するには、次のように入力します。

```
advanced.exe
```

6 [Form1] ダイアログ・ボックスが表示されます。[接続] をクリックします。

アプリケーションが Adaptive Server サンプル・データベースに接続します。

7 [Form1] ダイアログボックスで [実行] をクリックします。

アプリケーションはストアド・プロシージャを実行し、入力／出力パラメータ、出力パラメータ、戻り値を返します。

8 ウィンドウの右上角にある [X] をクリックすると、アプリケーションが終了し、サンプル・データベースとの接続が切断されます。

これでアプリケーションを実行できました。次の項では、アプリケーション・コードについて説明します。

Advanced サンプル・プロジェクトの理解

この項では、Advanced コード・サンプルを利用して、Adaptive Server ADO.NET Data Provider の一部の主要機能について説明します。Advanced プロジェクトは、Adaptive Server サンプル・データベース pubs2 を使用します。このデータベースは、Adaptive Server の CD からインストールできます。

この項では、数行ずつコードを説明します。コード全体を参照する場合は、サンプル・プロジェクトを開いてください。

C# の場合：

```
<インストール・ディレクトリ>¥Samples¥CSharp¥Advanced¥Advanced.csproj
```

Visual Basic .NET の場合：

```
<インストール・ディレクトリ>¥Samples¥VB.NET¥Advanced¥Advanced.vbproj
```

トレース・イベント・ハンドラの付加

次のコード行は、トレース・イベント・ハンドラを AseConnection に付加します。

C# の場合：

```
_conn.TraceEnter += new
     TraceEnterEventHandler(TraceEnter);
_conn.TraceExit += new
     TraceExitEventHandler(TraceExit);
```

Visual Basic .NET の場合：

```
AddHandler _conn.TraceEnter, AddressOf TraceEnter
AddHandler _conn.TraceExit, AddressOf TraceExit
```

名前付きパラメータを使用したストアド・プロシージャの呼び出し

メソッド ExecuteCommandUsingNamedParams() は、名前付きパラメータを使用して、名前でストアド・プロシージャを呼び出します。

C# の場合：

```
using(AseCommand cmd = new AseCommand("sp_hello", _conn))
{
   cmd.CommandType = CommandType.StoredProcedure;

   AseParameter inParam = new AseParameter("@inParam", AseDbType.VarChar, 32);
   inParam.Direction = ParameterDirection.Input;
   inParam.Value = textBoxInput.Text;
   cmd.Parameters.Add(inParam);

   AseParameter inoutParam = new AseParameter("@inoutParam",
   AseDbType.VarChar, 64);
   inoutParam.Direction = ParameterDirection.InputOutput;
   inoutParam.Value = textBoxInOut.Text;
   cmd.Parameters.Add(inoutParam);

   AseParameter outParam = new AseParameter("@outParam",
   AseDbType.VarChar, 64);
   outParam.Direction = ParameterDirection.Output;
   cmd.Parameters.Add(outParam);

   AseParameter retValue = new AseParameter("@retValue", AseDbType.Integer);
   retValue.Direction = ParameterDirection.ReturnValue;
   cmd.Parameters.Add(retValue);

   try
   {
      cmd.ExecuteNonQuery();
   }
   catch (AseException ex)
   {
      MessageBox.Show( ex.Source + " :" + ex.Message + " (" + ex.ToString() +
      ")", "Execute Stored Precedure failed.");
   }
}
```

Visual Basic .NET の場合：

```
Dim cmd As New AseCommand("sp_hello", _conn)
' set command type to stored procedure
cmd.CommandType = CommandType.StoredProcedure

' create the input parameter object and bind it to the command
Dim inParam As New AseParameter("@inParam", AseDbType.VarChar, 32)
inParam.Direction = ParameterDirection.Input
inParam.Value = textBoxInput.Text
cmd.Parameters.Add(inParam)

' create the inout parameter object and bind it to the command
Dim inoutParam As New AseParameter("@inoutParam", AseDbType.VarChar, 64)
inoutParam.Direction = ParameterDirection.InputOutput
inoutParam.Value = textBoxInOut.Text
cmd.Parameters.Add(inoutParam)

' create the output parameter object and bind it to the command
Dim outParam As New AseParameter("@outParam", AseDbType.VarChar, 64)
outParam.Direction = ParameterDirection.Output
cmd.Parameters.Add(outParam)

' create the return value object and bind it to the command
Dim retValue As New AseParameter("@retValue", AseDbType.Integer)
retValue.Direction = ParameterDirection.ReturnValue
cmd.Parameters.Add(retValue)

' execute the stored procedure
Try
   cmd.ExecuteNonQuery()

Catch ex As AseException
   MessageBox.Show(ex.Source + " :" + ex.Message + " (" + ex.ToString() + ")",
      "Execute Query failed.")
Finally
   ' dispose the command object
   cmd.Dispose()
End Try
```

呼び出し構文とパラメータ・マーカを使用したストアド・プロシージャの呼び出し

メソッド ExecuteCommandUsingParameterMarkers() は、呼び出し構文とパラメータ・マーカを使用してストアド・プロシージャを呼び出します。

C# の場合：

```
using(AseCommand cmd = new AseCommand("{ ?= call sp_hello(?, ?, ?)}", _conn))
{
   cmd.NamedParameters = false;
   AseParameter retValue = new AseParameter(0, AseDbType.Integer);
   retValue.Direction = ParameterDirection.ReturnValue;
   cmd.Parameters.Add(retValue);

   AseParameter inParam = new AseParameter(1, AseDbType.VarChar, 32);
   inParam.Direction = ParameterDirection.Input;
   inParam.Value = textBoxInput.Text;
   cmd.Parameters.Add(inParam);

   AseParameter inoutParam = new AseParameter(2, AseDbType.VarChar, 64);
   inoutParam.Direction = ParameterDirection.InputOutput;
   inoutParam.Value = textBoxInOut.Text;
   cmd.Parameters.Add(inoutParam);

   AseParameter outParam = new AseParameter(3, AseDbType.VarChar, 64);
   outParam.Direction = ParameterDirection.Output;
   cmd.Parameters.Add(outParam);
   try
   {
      cmd.ExecuteNonQuery();
   }
   catch (AseException ex)
   {
      MessageBox.Show( ex.Source + " :" + ex.Message + " (" + ex.ToString() +
      ")", "Execute Stored Precedure failed.");
   }
}
```

Visual Basic .NET の場合：

```
Dim cmd As New AseCommand("{ ?= call sp_hello(?, ?, ?)}", _conn)
' need to notify Named Parameters are not being used (which is the default)
cmd.NamedParameters = False

' create the return value object and bind it to the command
Dim retValue As New AseParameter(0, AseDbType.Integer)
retValue.Direction = ParameterDirection.ReturnValue
cmd.Parameters.Add(retValue)

' create the input parameter object and bind it to the command
Dim inParam As New AseParameter(1, AseDbType.VarChar, 32)
inParam.Direction = ParameterDirection.Input
inParam.Value = textBoxInput.Text
cmd.Parameters.Add(inParam)

' create the inout parameter object and bind it to the command
Dim inoutParam As New AseParameter(2, AseDbType.VarChar, 64)
inoutParam.Direction = ParameterDirection.InputOutput
inoutParam.Value = textBoxInOut.Text
cmd.Parameters.Add(inoutParam)

' create the output parameter object and bind it to the command
Dim outParam As New AseParameter(3, AseDbType.VarChar, 64)
outParam.Direction = ParameterDirection.Output
cmd.Parameters.Add(outParam)

' execute the stored procedure
Try
   cmd.ExecuteNonQuery()

   ' get the output, inout and return values and display them
   textBoxReturn.Text = cmd.Parameters(0).Value
   textBoxReturn.ForeColor = Color.Blue

   textBoxInOut.Text = cmd.Parameters(2).V

   textBoxOutput.Text = cmd.Parameters(3).Value
   textBoxOutput.ForeColor = Color.Blue
Catch ex As AseException
   MessageBox.Show(ex.Source + " :" + ex.Message + " (" + ex.ToString() + ")",_
   "Execute Query Failed")
Finally
   ' dispose the command object
   cmd.Dispose()
End Try
```

第 3 章

アプリケーションの開発

この章では、Adaptive Server ADO.NET Data Provider を使用したアプリケーションの開発と配備の方法について説明します。

Visual Studio .NET プロジェクトでの Data Provider の使用

Adaptive Server ADO.NET Data Provider をインストールしたら、Visual Studio .NET プロジェクトに次の 2 つの変更を加えて、使用できるようにします。

- Adaptive Server ADO.NET Data Provider アセンブリに対する参照を追加する。
- ソース・コードに Adaptive Server ADO.NET Data Provider クラスを参照する行を追加する。

Adaptive Server ADO.NET Data Provider のインストールと登録の詳細については、「Adaptive Server ADO.NET Data Provider の配備」(2 ページ) を参照してください。

Data Provider アセンブリへの参照の追加

参照を追加して、Adaptive Server ADO.NET Data Provider のコードを検索するために含めるアセンブリを Visual Studio .NET に指示します。

❖ **Visual Studio .NET プロジェクトでの Adaptive Server ADO.NET Data Provider に対する参照の追加**

1 Visual Studio .NET を起動して、プロジェクトを開きます。

2 [ソリューション エクスプローラ] ウィンドウで、[参照設定] フォルダを右クリックし、ポップアップ・メニューから、[参照の追加] を選択します。

[参照の追加] ダイアログ・ボックスが表示されます。

3 [.NET] タブで、Sybase.Data.AseClient コンポーネントが見つかるまでコンポーネントの一覧をスクロールします。このコンポーネントを指定して、[選択] をクリックします。

4 [OK] をクリックします。

コンポーネントの一覧に Adaptive Server ADO.NET Data Provider アセンブリがない場合は、[参照] から <インストール・ディレクトリ>*¥dll* ディレクトリにある *Sybase.Data.AseClient.dll* を探します。この DLL を選択して [開く] をクリックします。次に、[OK] をクリックします。

注意 Adaptive Server ADO.NET Data Provider の場合、デフォルトのロケーションは、*C:¥Sybase¥DataAccess¥ADONET¥dll* です。

プロジェクトの [ソリューション エクスプローラ] ウィンドウの [参照設定] フォルダにアセンブリが追加されます。

Adaptive Server ADO.NET Data Provider クラスの参照

Adaptive Server ADO.NET Data Provider を使用するには、Adaptive Server ADO.NET Data Provider を参照する行もソース・コードに追加します。C# と Visual Basic .NET では追加する行が異なります。

❖ **ソース・コードでの Adaptive Server ADO.NET Data Provider クラスの参照**

1 Visual Studio .NET を起動して、プロジェクトを開きます。

- C# の場合は、プロジェクトの先頭にある `using` ディレクティブの一覧に次の行を追加します。

```
using Sybase.Data.AseClient;
```

- Visual Basic .NET の場合は、プロジェクトの先頭にある行 `Public Class Form1` の前に次の行を追加します。

```
Imports Sybase.Data.AseClient
```

この行は必須ではありません。ただし、これによって Adaptive Server クラスの省略形を使用できるようになります。このコード行がない場合でも、次のコード行を使用できます。

```
Sybase.Data.AseClient.AseConnection conn = new
Sybase.Data.AseClient.AseConnection();
```

上記のコード行を次のコード行の代わりに使用します。

```
AseConnection conn = new AseConnection();
```

データベースへの接続

データに対して操作を実行するには、最初にアプリケーションをデータベースに接続します。この項では、Adaptive Server データベースに接続するコードの記述方法について説明します。

詳細については、「AseConnection クラス」(113 ページ) と 「ConnectionString プロパティ」(120 ページ) を参照してください。

❖ Adaptive Server データベースへの接続

1　AseConnection オブジェクトを割り付けます。

次のコードは、“conn” という名前の AseConnection オブジェクトを作成します。

C# の場合：

```
AseConnection conn = new AseConnection();
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection()
```

アプリケーションからデータベースへ複数の接続を設定できます。アプリケーションによっては、Adaptive Server データベースに対する接続を 1 つだけ使用して、常時この接続をオープンにします。これを実行するには、接続にグローバル変数を宣言します。

C# の場合：

```
private AseConnection_conn;
```

Visual Basic .NET の場合：

```
Private _conn As AseConnection
```

詳細については、<インストール・ディレクトリ>¥*Samples* にある Table Viewer のサンプル・コードと「Table Viewer サンプル・プロジェクトの理解」(18 ページ) を参照してください。

2 データベースへの接続に使用する接続文字列を指定します。

C# の場合：

```
AseConnection conn = new AseConnection(
   "Data Source='mango';Port=5000;" +
   "UID='sa';PWD='';" +
   "Database='pubs2';" );
```

“mango” には、データベース・サーバが実行されているホスト名を指定します。

Visual Basic .NET の場合：

```
Dim conn As New AseConnection(_
     "Data Source='mango',Port=5000," +_
     "UID='sa';PWD='''';" + _
     "Database='pubs2';")
```

接続パラメータの完全なリストについては、「AseConnection コンストラクタ」(114 ページ) を参照してください。

3 次のコードを使用して、データベースへの接続をオープンします。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

4 接続エラーを検出します。

データベースへの接続試行時に発生したすべてのエラーが検出されるようにアプリケーションを設計してください。次のコードは、エラーを検出し、そのメッセージを表示する方法を示しています。

C# の場合：

```
try {
   _conn = new AseConnection(
      txtConnectString.Text );
   _conn.Open();
}
catch( AseException ex ) {
   MessageBox.Show(
      ex.Message,
      "Failed to connect");
}
```

Visual Basic .NET の場合：

```
Try
   _conn = New AseConnection(_
      txtConnectionString.Text)
   _conn.Open()
Catch ex As AseException
   MessageBox.Show(_
      ex.Message,_
      "Failed to connect")
End Try
```

AseConnection オブジェクトの作成時に接続文字列を渡すのではなく、ConnectionString プロパティを使用して接続文字列を設定することもできます。

C# の場合：

```
AseConnection conn = new AseConnection();
conn.ConnectionString = "Data Source='mango';" +
   "Port=5000;" +
   "UID='sa';" +
   "PWD='';" +
   "Database='pubs2';" ;
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection()
conn.ConnectionString = "Data Source='mango';" + _
   "Port=5000;" + _
   "UID='sa';" + _
   "PWD='';" + _
   "Database='pubs2';"
```

“mango” には、データベース・サーバ名を指定します。

5　データベースへの接続をクローズします。conn.Close() メソッドを使用して明示的にクローズするまで、データベースへの接続はオープンしたままになります。

接続プール

Adaptive Server Enterprise ADO.NET プロバイダは、アプリケーションがプールからの既存の接続を再使用できる接続プールをサポートします。これにより、データベースへの新しい接続を繰り返し作成する代わりに、接続ハンドルをプールに保存し、接続を再使用できるようになります。接続プールは、デフォルトでオンに設定されています。

プール・サイズの最小値と最大値を指定することもできます。次に例を示します。

```
"Data Source='mango';" +
   "Port=5000;" +
   "UID='sa';" +
   "PWD='';" +
   "Database='pubs2';" +
   "Max Pool Size=50;" +
   "Min Pool Size=5";
```

アプリケーションがデータベースに初めて接続しようとするとき、アプリケーションは、指定した接続パラメータと同じパラメータを使用している既存の接続がプールにあるかどうかをチェックします。一致する接続があった場合は、その接続を使用します。一致する接続がなかった場合は、新しい接続を使用します。接続が切断されると、その接続はプールに戻されて再利用できるようになります。

注意 Max Pool Size が指定されている場合は、オープンできる接続の最大数がこの値に制限されます。制限値に達すると、AseConnection.Open() の呼び出しが失敗して AseException が発生します。

接続プールの無効化

接続プールを無効にするには、接続文字列で Pooling=False を指定します。

接続ステータスの確認

データベースへの接続が確立した後は、接続ステータスをチェックして接続がオープンになっていることを確認してから、データをフェッチして更新できます。接続が失われたり、ビジー状態だったり、別のコマンドを処理中の場合は、それに相当するメッセージを返すことができます。

AseConnection クラスには、接続ステータスをチェックする「State プロパティ」があります。使用可能なステータス値は Open と Closed です。

次のコードは、Connection オブジェクトが初期化されているかどうかをチェックし、初期化されている場合は、接続がオープンになっていることを確認します。

C# の場合：

```
if( _conn == null || _conn.State !=
   ConnectionState.Open )
   {
      MessageBox.Show( "Connect to a database first",
      "Not connected" );
                     return;
   }
```

Visual Basic .NET の場合：

```
If (_conn Is Nothing) OrElse (_conn.State <>
ConnectionState.Open) Then
     MessageBox.Show("Connection to a database first",
     "Error")
     Return
End If
```

接続がオープンになっていない場合は、メッセージが返されます。詳細については、「State プロパティ」(123 ページ) を参照してください。

文字セット

Adaptive Server ADO.NET Data Provider は、ネゴシエートされた文字セットを使用して Adaptive Server と通信します。この文字セットは、Adaptive Server ADO.NET Data Provider のユーザ・インタフェースで ClientCharset または Server Default に設定できます。Server Default がデフォルト設定です。

使用法

- ClientCharset を文字セットとして選択する場合は、CodePageType プロパティの値も指定します。有効な値は、ANSI (デフォルト値) および OEM です。
- Other を選択する場合、charset プロパティの値には文字セットの名前を指定します。たとえば、“charset=utf8” いう指定では、ネゴシエートされた文字セットは utf8 に設定されます。
- charset プロパティは、できるだけ使用しないでください。.NET 文字列は unicode であり、マルチバイトに変換してから Adaptive Server に char または varchar パラメータとして送信する必要があります。このため、サーバのデフォルト以外の文字セットを使用した場合、パフォーマンスに影響します。
- サポートされていない文字セットを使用すると、次のようなエラーが発生します。

  ```
  [Sybase][driver] Could not load code page for requested
  charset
  ```

  このエラーを回避するには、次のいずれかの手順を実行します。

- ドライバのユーザ・インタフェースで、[詳細設定] タブを開き、ClientCharset を文字セットとして選択します。Other を選択した場合、指定した文字セットがサポートされることを確認してください。
- 接続文字列では、charset プロパティで、サポートされている文字セットが指定されていることを確認してください。

データに対するアクセスと操作

Adaptive Server ADO.NET Data Provider のデータ・アクセス方法には、AseCommand オブジェクトを使用する方法と AseDataAdapter オブジェクトを使用する方法の 2 つがあります。

- **AseCommand オブジェクト**：AseCommand オブジェクトを使用する方法は、プログラマが接続をより制御しやすくなるため、.NET でデータ・アクセスとデータ操作を行う場合におすすめします。ただし、AseDataAdapter を使用すればオフラインでの作業が可能になります。

  AseCommand オブジェクトを使用すると、SQL 文を実行してデータベースのデータを直接取得したり修正したりできます。また AseCommand オブジェクトを使用すると、データベースに対して直接 SQL 文を発行してストアド・プロシージャを呼び出すことができます。

  AseCommand オブジェクト内では、AseDataReader クラスを使用して、クエリまたはストアド・プロシージャから読み込み専用の結果セットを返すことができます。

  詳細については、「AseCommand クラス」(101 ページ) と 「AseDataReader クラス」(131 ページ) を参照してください。

- **AseDataAdapter オブジェクト**：AseDataAdapter オブジェクトは、結果セット全体を DataSet に取得します。DataSet は接続が切断された記憶領域で、データベースから取得されたデータが格納されます。DataSet に格納されたデータは編集できます。編集操作が終了すると、AseDataAdapter オブジェクトは、DataSet に対して行われた変更でデータベースを更新します。AseDataAdapter を使用する場合、DataSet に取得したローを他のユーザが変更しないように防ぐ方法はありません。そのため、発生する可能性のある競合すべてを解決するロジックをアプリケーションに組み込む必要があります。

  詳細については、「AseDataAdapter を使用する場合の競合の解決」(51 ページ) を参照してください。

  AseDataAdapter オブジェクトの詳細については、「AseDataAdapter クラス」(124 ページ) を参照してください。

AseCommand を使用したデータの取得と操作

以降の各項では、AseDataReader を使用したデータの取得と、ローの挿入、更新、削除の方法について説明します。

AseCommand オブジェクトを使用したデータの取得

AseCommand オブジェクトを使用すると、Adaptive Server データベースに対して SQL 文を発行したり、ストアド・プロシージャを呼び出したりできます。データベースからデータを取得するには、次の種類のコマンドを発行します。

- **ExecuteReader**：単一の結果セットを返すコマンドを発行します。デフォルトではカーソルは使用されません。結果セット全体がクライアント側でフェッチされ、ユーザは 1 度に 1 つのローを前方向へのみフェッチできます。カーソルの使用をオンにするには、次の行を ConnectString に追加します。

  ```
  "Use Cursor=true;"
  ```

  これにより、データベース・サーバから結果セット全体をフェッチする代わりに、前方向への読み込み専用カーソルが使用されるようになります。

  カーソルを使用すると、クエリによって大規模な resultset が返されることが予測される場合にパフォーマンスを改善できますが、クライアントは必然的に resultset 全体を使用できなくなります。

  いずれの場合も、ユーザは一方向にのみ結果セットのローをすばやくループできます。

  詳細については、「ExecuteReader メソッド」(104 ページ) を参照してください。

- **ExecuteScalar**：単一の値を返すコマンドを発行します。結果セットの最初のローの最初のカラムや、COUNT や AVG など、集計値を返す SQL 文が返される場合もあります。

  詳細については、「ExecuteScalar メソッド」(104 ページ) を参照してください。

- **ExecuteXmlReader**：単一の結果セットを XML フォーマットで返すコマンドを発行します。通常、このメソッドは FOR XML 句のある select 文で使用されます。

  詳細については、「ExecuteXmlReader メソッド」(105 ページ) を参照。

以降の説明では、Adaptive Server ADO.NET Data Provider に含まれている Simple コード・サンプルを使用します。

Simple コード・サンプルの詳細については、「Simple サンプル・プロジェクトの理解」(13 ページ) を参照してください。

❖ 結果セット全体を返すコマンドの発行

1 Connection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection(connStr);
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection(connStr)
```

C# の場合：

```
try {
     conn.Open();
}
catch (AseException ex)
{
     <error handling>
}
```

Visual Basic .NET の場合：

```
Try
     conn.Open()
Catch ex As AseExecption
     <error handling>
End Try
```

2 SQL 文を定義して実行する Command オブジェクトを追加します。

C# の場合：

```
AseCommand cmd = new AseCommand(_
   "select au_lname from authors", conn );
```

Visual Basic .NET の場合：

```
Dim cmd As New AseCommand("select au_lname from authors",
conn)
```

注意 ストアド・プロシージャを使用してデータベースからデータを取得する場合、ストアド・プロシージャから出力パラメータ値と結果セットの両方が返されると、結果セットはリセットされ、出力パラメータ値が参照されると同時に結果セットのローは参照できなくなります。Sybase では、このような場合、結果セットのローすべてを参照して使い終わるまで、参照側の出力パラメータ値を最後までそのままにしておくことをおすすめします。

詳細については、「ストアド・プロシージャの使用」(72 ページ) と「AseParameter クラス」(153 ページ) を参照してください。

3　ExecuteReader メソッドを呼び出して、DataReader オブジェクトを返します。

C# の場合：

```
AseDataReader reader = cmd.ExecuteReader();
```

Visual Basic .NET の場合：

```
Dim reader as AseDataReader = cmd.ExecuteReader()
```

4　結果を表示します。

C# の場合：

```
listAuthors.BeginUpdate();
while( reader.Read() ) {
   listAuthors.Items.Add( reader.GetString( 0 ) );
}
listAuthors.EndUpdate();
```

Visual Basic .NET の場合：

```
 listAuthors.BeginUpdate()
While reader.Read()
     listAuthors.Items.Add(reader.GetString(0))
End While
listAuthors.EndUpdate()
```

5　DataReader オブジェクトと Connection オブジェクトをクローズします。

C# の場合：

```
reader.Close();
conn.Close();
```

Visual Basic .NET の場合：

```
reader.close()
conn.close()
```

❖ 単一の値のみを返すコマンドの発行

1　AseConnection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection(
   "Data Source='mango';" +
   "Port=5000;" +
   "UID='sa';" +
   "PWD='';" +
   "Database='pubs2';" );
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection( _
   "Data Source='mango';" + _
   "Port=5000;" + _
   "UID='sa';" + _
   "PWD='';" + _
   "Database='pubs2';")
```

“mango” には、データベース・サーバ名を指定します。

2 接続をオープンします。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

3 SQL 文を定義して実行する AseCommand オブジェクトを追加します。

C# の場合：

```
AseCommand cmd = new AseCommand(
  "select count(*) from authors where state = 'CA'",
  conn );
```

Visual Basic .NET の場合：

```
Dim cmd As New AseCommand(
 "select count(*) from authors where state = 'CA'"_,
 conn );
```

4 ExecuteScalar メソッドを呼び出して、値を含むオブジェクトを返します。

C# の場合：

```
int count = (int) cmd.ExecuteScalar();
```

Visual Basic .NET の場合：

```
Dim count As Integer = cmd.ExecuteScalar()
```

5 AseConnection オブジェクトをクローズします。

C# の場合：

```
conn.Close();
```

Visual Basic .NET の場合：

```
conn.Close()
```

AseDataReader には、任意のデータ型で結果を返すことのできる Get メソッドがいくつかあります。

詳細については、「AseDataReader クラス」(131 ページ) を参照してください。

❖ XmlReader オブジェクトを返すコマンドの発行

1　Connection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection(connStr);
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection(connStr)
```

2　接続をオープンします。

C# の場合：

```
try {
   conn.Open();
}
catch (AseException ex)
{
   <error handling>
}
```

Visual Basic .NET の場合：

```
Try
   conn.Open()
Catch ex As AseExecption
   <error handling>
End Try
```

3　SQL 文を定義して実行する Command オブジェクトを追加します。

C# の場合：

```
AseCommand cmd = new AseCommand(
   "select * from authors for xml",
   conn );
```

Visual Basic .NET の場合：

```
Dim cmd As New AseCommand( _
   "select au_lname from authors for xml", _
   conn
```

4　ExecuteReader メソッドを呼び出して、DataReader オブジェクトを返します。

C# の場合：

```
XmlReader reader = cmd.ExecuteXmlReader();
```

Visual Basic .NET の場合：

```
Dim reader as XmlReader = cmd.ExecuteXmlReader()
```

5 XML の結果を使用します。

C# の場合：

```
reader.read();
<process xml>
```

Visual Basic .NET の場合：

```
reader.read()
<process xml>
```

6 DataReader オブジェクトと Connection オブジェクトをクローズします。

C# の場合：

```
reader.Close();
conn.Close();
```

Visual Basic .NET の場合：

```
reader.close()
conn.close()
```

AseCommand オブジェクトを使用したローの挿入、更新、削除

AseCommand オブジェクトを使用して、Insert、Update、または Delete 操作を実行するには、ExecuteNonQuery 関数を使用します。ExecuteNonQuery 関数は、結果セットを返さないコマンド (SQL 文またはストアド・プロシージャ) を発行します。

詳細については、「ExecuteNonQuery メソッド」(103 ページ) を参照してください。

自動インクリメント・プライマリ・キーのプライマリ・キー値を取得する方法については、「プライマリ・キー値の取得」(63 ページ) を参照してください。

コマンドの独立性レベルを設定する場合は、AseCommand オブジェクトを AseTransaction オブジェクトの一部として使用してください。AseTransaction オブジェクトを使用せずにデータを修正すると、Adaptive Server ADO.NET Data Provider は autocommit モードで動作し、加えられたすべての変更がただちに適用されます。

詳細については、「トランザクション処理」(75 ページ) を参照してください。

❖ ローを挿入するコマンドの発行

1 AseConnection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection( c_connStr );
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection(c_connStr)
```

2　接続をオープンします。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

3　Insert 文を定義して実行する AseCommand オブジェクトを追加します。

C# の場合：

```
AseCommand insertCmd = new AseCommand(
   "INSERT INTO publishers " +
   " ( pub_id, pub_name, city, state) " +
   " VALUES( @pub_id, @pub_name, @city, @state )",
   conn);
```

Visual Basic .NET の場合：

```
Dim insertCmd As new AseCommand( _
   "INSERT INTO publishers " + _
   " ( pub_id, pub_name, city, state) " + _
   " VALUES (@pub_id, @pub_name, @city, @state )",  _
   conn )
```

4　AseCommand オブジェクトのパラメータを設定します。

次のコードは、それぞれ dept_id カラムと dept_name カラムのパラメータを定義します。

C# の場合：

```
AseParameter parm = new AseParameter("@pub_id", AseDbType.Char, 4);
insertCmd.Parameters.Add( parm );
parm = new AseParameter("@pub_name", AseDbType.VarChar, 40);
insertCmd.Parameters.Add( parm );
parm = new AseParameter("@city", AseDbType.VarChar, 20);
insertCmd.Parameters.Add( parm );
parm = new AseParameter("@state", AseDbType.Char, 2);
insertCmd.Parameters.Add( parm );
```

Visual Basic .NET の場合：

```
Dim parm As New AseParameter("@pub_id", AseDbType.Char, 4)
insertCmd.Parameters.Add(parm)
parm = New AseParameter("@pub_name", AseDbType.VarChar, 40)
insertCmd.Parameters.Add(parm)
parm = New AseParameter("@city", AseDbType.VarChar, 20)
insertCmd.Parameters.Add(parm)
parm = New AseParameter("@state", AseDbType.Char, 2)
insertCmd.Parameters.Add(parm)
```

5 新しい値を挿入して ExecuteNonQuery メソッドを呼び出し、データベースに変更を適用します。

C# の場合：

```
int recordsAffected = 0;
insertCmd.Parameters[0].Value = "9901";
insertCmd.Parameters[1].Value = "New Publisher";
insertCmd.Parameters[2].Value = "Concord";
insertCmd.Parameters[3].Value = "MA";
recordsAffected = insertCmd.ExecuteNonQuery();
insertCmd.Parameters[0].Value = "9902";
insertCmd.Parameters[1].Value = "My Publisher";
insertCmd.Parameters[2].Value = "Dublin";
insertCmd.Parameters[3].Value = "CA";
recordsAffected = insertCmd.ExecuteNonQuery();
```

Visual Basic .NET の場合：

```
Dim recordsAffected As Integer
insertCmd.Parameters(0).Value = "9901"
insertCmd.Parameters(1).Value = "New Publisher"
insertCmd.Parameters(2).Value = "Concord"
insertCmd.Parameters(3).Value = "MA"
recordsAffected = insertCmd.ExecuteNonQuery()
insertCmd.Parameters(0).Value = "9902"
insertCmd.Parameters(1).Value = "My Publisher"
insertCmd.Parameters(2).Value = "Dublin"
insertCmd.Parameters(3).Value = "CA"
recordsAffected = insertCmd.ExecuteNonQuery()
```

注意 Insert、Update、Delete 文は、ExecuteNonQuery メソッドとともに使用できます。

6 結果を表示して、ウィンドウのグリッドにバインドします。

C# の場合：

```
AseCommand selectCmd = new AseCommand("SELECT * FROM publishers", conn );
AseDataReader dr = selectCmd.ExecuteReader();
dataGrid.DataSource = dr;
```

Visual Basic .NET の場合：

```
Dim selectCmd As New AseCommand("SELECT * FROM publishers", conn)
Dim dr As AseDataReader = selectCmd.ExecuteReader()
DataGrid.DataSource = dr
```

7　AseDataReader オブジェクトと AseConnection オブジェクトをクローズします。

C# の場合：

```
dr.Close();
conn.Close();
```

Visual Basic .NET の場合：

```
dr.Close()
conn.Close()
```

❖ ローを更新するコマンドの発行

1　AseConnection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection(
   c_connStr );
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection(c_connStr)
```

2　接続をオープンします。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

3　Update 文を定義して実行する AseCommand オブジェクトを追加します。

C# の場合：

```
AseCommand updateCmd = new AseCommand(
   "UPDATE publishers " +
   "SET pub_name = 'My Publisher' " +
   "WHERE pub_id='9901'",
   conn );
```

Visual Basic .NET の場合：

```
Dim updateCmd As New AseCommand( _
   "UPDATE publishers " + _
   "SET pub_name = 'My Publisher' " + _
   "WHERE pub_id='9901'", _
   conn )
```

詳細については、「ストアド・プロシージャの使用」(72 ページ) と「AseParameter クラス」(153 ページ) を参照してください。

4 ExecuteNonQuery メソッドを呼び出して、データベースに変更を適用します。

C# の場合：

```
int recordsAffected = updateCmd.ExecuteNonQuery();
```

Visual Basic .NET の場合：

```
Dim recordsAffected As Integer =
   updateCmd.ExecuteNonQuery()
```

5 結果を表示して、ウィンドウのグリッドにバインドします。

C# の場合：

```
AseCommand selectCmd = new AseCommand(
   "SELECT * FROM publishers", conn );
AseDataReader dr = selectCmd.ExecuteReader();
dataGrid.DataSource = dr;
```

Visual Basic .NET の場合：

```
Dim selectCmd As New AseCommand(_
   "SELECT * FROM publishers", conn)
Dim dr As AseDataReader = selectCmd.ExecuteReader()
DataGrid.DataSource = dr
```

6 AseDataReader オブジェクトと AseConnection オブジェクトをクローズします。

C# の場合：

```
dr.Close();
conn.Close();
```

Visual Basic .NET の場合：

```
dr.Close()
conn.Close()
```

❖ ローを削除するコマンドの発行

1 AseConnection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection(c_connStr);
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection(c_connStr)
```

2　接続をオープンします。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

3　Delete 文を定義して実行する AseCommand オブジェクトを作成します。

C# の場合：

```
AseCommand updateCmd = new AseCommand
   "DELETE FROM publishers " +
   " WHERE (pub_id > '9900')",
   conn );
```

Visual Basic .NET の場合：

```
Dim updateCmd As New AseCommand(_
"DELETE FROM publishers " + _
"WHERE (pub_id > '9900')", _
conn )
```

4　ExecuteNonQuery メソッドを呼び出して、データベースに変更を適用します。

C# の場合：

```
int recordsAffected = deleteCmd.ExecuteNonQuery();
```

Visual Basic .NET の場合：

```
Dim recordsAffected As Integer =
   updateCmd.ExecuteNonQuery()
```

5　AseConnection オブジェクトをクローズします。

C# の場合：

```
conn.Close();
```

Visual Basic .NET の場合：

```
dr.Close()
conn.Close()
```

DataReader スキーマ情報の取得

結果セット内のカラムに関するスキーマ情報を取得できます。

AseDataReader を使用している場合、GetSchemaTable メソッドを使用して結果セットの情報を取得できます。GetSchemaTable メソッドは、標準の .NET DataTable オブジェクトを返します。このオブジェクトは、カラム・プロパティも含め、結果セット内のすべてのカラムに関する情報を提供します。

GetSchemaTable メソッドの詳細については、「GetSchemaTable メソッド」(139 ページ) を参照してください。

❖ ***GetSchemaTable* メソッドを使用した結果セット情報の取得**

1 Connection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection(
   c_connStr );
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection( _
   c_connStr )
```

2 接続をオープンします。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

3 目的の Select 文を使用して AseCommand オブジェクトを作成します。このクエリの結果セットのスキーマが返されます。

C# の場合：

```
AseCommand cmd = new AseCommand(
   "SELECT * FROM authors", conn );
```

Visual Basic .NET の場合：

```
Dim cmd As New AseCommand( _
   "SELECT * FROM authors", conn )
```

4 AseDataReader オブジェクトを作成して、作成した Command オブジェクトを実行します。

C# の場合：

```
AseDataReader dr = cmd.ExecuteReader();
```

Visual Basic .NET の場合：

```
Dim dr As AseDataReader = cmd.ExecuteReader()
```

5　データ・ソースからのスキーマを DataTable に書き込みます。

C# の場合：

```
DataTable
schema = dr.GetSchemaTable();
```

Visual Basic .NET の場合：

```
Dim schema As DataTable = _
   dr.GetSchemaTable()
```

6　AseDataReader オブジェクトと AseConnection オブジェクトをクローズします。

C# の場合：

```
dr.Close();
conn.Close();
```

Visual Basic .NET の場合：

```
dr.Close()
conn.Close()
```

7　DataTable をウィンドウのグリッドにバインドします。

C# の場合：

```
dataGrid.DataSource = schema;
```

Visual Basic .NET の場合：

```
dataGrid.DataSource = schema
```

AseDataAdapter を使用したデータへのアクセスと操作

以降の各項では、AseDataAdapter を使用したデータの取得と、ローの挿入、更新、削除の方法について説明します。

AseDataAdapter オブジェクトを使用したデータの取得

AseDataAdapter は、Fill メソッドを使用してクエリの結果を DataSet に書き込み、その DataSet を表示グリッドにバインドすることによって結果セット全体を表示します。

AseDataAdapter を使用すると、単一の結果セットを返す任意の文字列 (SQL 文またはストアド・プロシージャ) を渡すことができます。AseDataAdapter では、デフォルトで、すべてのローが 1 度の操作でフェッチされます。カーソルを使用するには、接続文字列でプロパティを 'use cursor = true' に設定します。この場合、前方向への読み込み専用カーソルが使用され、すべてのローが読み込まれると、カーソルは自動的にクローズします。AseDataAdapter では、DataSet に変更を加えることができます。変更操作が完了したら、データベースに再接続して変更を適用します。

AseDataAdapter オブジェクトを使用すると、ジョインに基づく結果セットを取得できます。

AseDataAdapter の詳細については、「AseDataAdapter クラス」(124 ページ) を参照してください。

AseDataAdapter の例

次の例は、AseDataAdapter を使用して DataSet にデータを書き込む方法を示しています。

❖ **AseDataAdapter オブジェクトを使用したデータの取得**

1 データベースに接続します。

2 新しい DataSet を作成します。ここでは、DataSet の名前を “Results” とします。

C# の場合：

```
DataSet ds =new DataSet ();
```

Visual Basic .NET の場合：

```
Dim ds As New DataSet()
```

3 SQL 文を実行して “Results” という DataSet に結果を書き込む、新しい AseDataAdapter オブジェクトを作成します。

C# の場合：

```
AseDataAdapter da=new
   AseDataAdapter(txtSQLStatement.Text, _conn);
da.Fill(ds, "Results"),
```

Visual Basic .NET の場合：

```
Dim da As New
   AseDataAdapter(txtSQLStatement.Text, conn)
da.Fill(ds, "Results")
```

4 DataSet をウィンドウのグリッドにバインドします。

C# の場合：

```
dgResults.DataSource = ds.Tables["Results"],
```

Visual Basic .NET の場合：

```
dgResults.DataSource = ds.Tables("Results")
```

AseDataAdapter オブジェクトを使用したローの挿入、更新、削除

AseDataAdapter オブジェクトは、DataSet に結果セットを取得します。DataSet はテーブルのコレクションであり、テーブル間の関係と制約を保持しています。DataSet は .NET Framework 内に構築され、データベースへの接続に使用される Adaptive Server ADO.NET Data Provider から独立しています。

AseDataAdapter を使用するときに、まだデータベースに接続していない場合は接続がオープンされ、DataSet への書き込みが行われた後、接続が明示的にオープンされたものでない場合は接続がクローズされます。ただし、DataSet に書き込まれたデータは、データベースとの接続が切断している間も変更できます。

変更をデータベースへただちに適用する必要がない場合は、データやスキーマを含め、DataSet を WriteXml メソッドを使用して XML ファイルに書き込むことができます。この場合、後から ReadXml メソッドを使用して DataSet をロードすれば、変更を適用できます。

詳細については、.Net Framework のドキュメントで WriteXml と ReadXml の説明を参照してください。

Update メソッドを呼び出して DataSet からデータベースへ変更を適用するとき、AseDataAdapter は加えられた変更を分析し、必要に応じて Insert、Update、Delete のいずれかのコマンドを呼び出します。

DataSet を使用して変更 (挿入、更新、削除) できるのは、単一のテーブルのデータのみです。ジョインに基づく結果セットは更新できません。

注意　DataSet に対する変更は、すべて接続が切断している間に行われます。これは、データベース内で、これらのローがアプリケーションによってロックされていないことを意味します。変更の適用対象データを他のユーザがすでに変更している場合もあるため、DataSet に加えた変更をデータベースに適用するときに発生する可能性のある競合すべてを解決するように、アプリケーションを設計する必要があります。

AseDataAdapter を使用する場合の競合の解決

対処する必要のある競合には次のものがあります。

- ユニークなプライマリ・キー － 2 人のユーザが 1 つのテーブルに新しいローを挿入する場合、それぞれのローにユニークなプライマリ・キーを設定する必要があります。自動インクリメント・プライマリ・キーのあるテーブルでは、DataSet の値とデータ・ソースの値とが同期しなくなることがあります。

  自動インクリメント・プライマリ・キーのプライマリ・キー値を取得する方法については、「プライマリ・キー値の取得」(63 ページ) を参照してください。

- 1 つの値に対する複数の更新 － 2 人のユーザが同じ値を変更するときに、どちらの値が正しいかを判定する論理をアプリケーションに組み込んでください。

- スキーマの変更 – DataSet で更新したテーブルのスキーマを他のユーザが変更した場合、その DataSet に加えた変更をデータベースに適用すると、更新は失敗します。
- データの同時実行性 – 同時実行性のある複数のアプリケーションが一貫したデータを参照できる場合、AseDataAdapter はフェッチしたローをロックしないため、DataSet を取得してオフラインで作業しているときに、2 人目のユーザがデータベース内の値を更新できます。

これらの潜在的な問題の多くは、AseCommand、AseDataReader、AseTransaction のオブジェクトを使用してデータベースに変更を適用することで回避できます。トランザクションの独立性レベルを設定し、別のユーザが変更できないようにローをロックするため、Sybase では AseTransaction オブジェクトを使用することをおすすめします。

トランザクションを使用して変更をデータベースに適用する方法の詳細については、「AseCommand オブジェクトを使用したローの挿入、更新、削除」(42 ページ) を参照してください。

競合解決処理を単純化するため、insert 文、update 文、または delete 文をストアド・プロシージャ・コールに設計できます。Insert、Update、Delete 文をストアド・プロシージャに組み込むと、操作が失敗した場合にエラーを検出できます。これらの文に加えて、ストアド・プロシージャにエラー処理論理を追加すると、操作が失敗した場合にエラーをログ・ファイルに記録したり、操作を再試行するなど適切なアクションを実行できます。

❖ AseDataAdapter を使用したテーブルへのローの挿入

1 AseConnection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection(c_connStr);
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection( _
   c_connStr )
```

2 接続をオープンします。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

3　新しい AseDataAdapter オブジェクトを作成します。

C# の場合：

```
AseDataAdapter adapter = new AseDataAdapter();
adapter.MissingMappingAction =
   MissingMappingAction.Passthrough;
adapter.MissingSchemaAction =
   MissingSchemaAction.Add;
```

Visual Basic .NET の場合：

```
Dim adapter As New AseDataAdapter()
adapter.MissingMappingAction = _
   MissingMappingAction.Passthrough
adapter.MissingSchemaAction = _
   MissingSchemaAction.Add
```

4　必要な AseCommand オブジェクトを作成して、必要なパラメータをすべて定義します。

次のコードは、Select コマンドと Insert コマンドを作成して、Insert コマンドのパラメータを定義します。

C# の場合：

```
adapter.SelectCommand = new AseCommand(
   "SELECT * FROM publishers", conn );
adapter.InsertCommand = new AseCommand(
   "INSERT INTO publishers( pub_id, pub_name, city, state) " +
   "VALUES( @pub_id, @pub_name, @city, @state  )", conn);
adapter.InsertCommand.UpdatedRowSource = UpdateRowSource.None;
AseParameter parm = new AseParameter("@pub_id", AseDbType.Char, 4);
parm.SourceColumn = "pub_id";
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parm );
parm = new AseParameter("@pub_name", AseDbType.VarChar, 40);
parm.SourceColumn = "pub_name";
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parm );
parm = new AseParameter("@city", AseDbType.VarChar, 20);
parm.SourceColumn = "city";
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parm );
parm = new AseParameter("@state", AseDbType.Char, 2);
parm.SourceColumn = "state";
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parm );
```

Visual Basic .NET の場合：

```
adapter.SelectCommand = New AseCommand( _
   "SELECT * FROM publishers", conn )
adapter.InsertCommand = New AseCommand( _
   "INSERT INTO publishers( pub_id, pub_name,  city, state) " + _
   " VALUES( @pub_id, @pub_name, @city, @state  )", conn)
adapter.InsertCommand.UpdatedRowSource = _
   UpdateRowSource.None
Dim parm As New AseParameter("@pub_id", AseDbType.Char, 4)
parm.SourceColumn = "pub_id"
parm.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parm )
parm = New AseParameter("@pub_name", AseDbType.VarChar, 40)
parm.SourceColumn = "pub_name"
parm.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parm )
parm = New AseParameter("@city", AseDbType.VarChar, 20)
parm.SourceColumn = "city"
parm.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parm )
parm = New AseParameter("@state", AseDbType.Char, 2)
parm.SourceColumn = "state"
parm.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parm )
```

5 DataTable に Select 文の結果を書き込みます。

C# の場合：

```
DataTable dataTable = new DataTable( "publishers" );
int rowCount = adapter.Fill( dataTable );
```

Visual Basic .NET の場合：

```
Dim dataTable As New DataTable( "publishers" )
Dim rowCount As Integer = adapter.Fill( dataTable )
```

6 DataTable に新しいローを挿入して、変更をデータベースに適用します。

C# の場合：

```
DataRow row1 = dataTable.NewRow();
row1[0] = "9901";
row1[1] = "New Publisher";
row1[2] = "Concord";
row1[3] = "MA";
dataTable.Rows.Add( row1 );
DataRow row2 = dataTable.NewRow();
row2[0] = "9902";
row2[1] = "My Publisher";
row2[2] = "Dublin";
row2[3] = "CA";
dataTable.Rows.Add( row2 );
int recordsAffected = adapter.Update( dataTable );
```

Visual Basic .NET の場合：

```
Dim row1 As DataRow = dataTable.NewRow()
row1(0) = "9901"
row1(1) = "New Publisher"
row1(2) = "Concord"
row1(3) = "MA"
dataTable.Rows.Add( row1 )
Dim row2 As DataRow = dataTable.NewRow()
row2(0) = "9902"
row2(1) = "My Publisher"
row2(2) = "Dublin"
row2(3) = "CA"
dataTable.Rows.Add( row2 )
Dim recordsAffected As Integer =_
   adapter.Update( dataTable )
```

7　更新の結果を表示します。

C# の場合：

```
dataTable.Clear();
rowCount = adapter.Fill( dataTable );
dataGrid.DataSource = dataTable;
```

Visual Basic .NET の場合：

```
dataTable.Clear()
rowCount = adapter.Fill( dataTable )
dataGrid.DataSource = dataTable
```

8　接続をクローズします。

C# の場合：

```
conn.Close();
```

Visual Basic .NET の場合：

```
conn.Close()
```

❖ AseDataAdapter オブジェクトを使用したローの更新

1　AseConnection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection( c_connStr );
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection( _
   c_connStr )
```

2 接続をオープンします。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

3 新しい AseDataAdapter オブジェクトを作成します。

C# の場合：

```
AseDataAdapter adapter = new AseDataAdapter();
adapter.MissingMappingAction =
   MissingMappingAction.Passthrough;
adapter.MissingSchemaAction =
   MissingSchemaAction.Add;
```

Visual Basic .NET の場合：

```
Dim adapter As New AseDataAdapter()
adapter.MissingMappingAction = _
   MissingMappingAction.Passthrough
adapter.MissingSchemaAction = _
   MissingSchemaAction.Add
```

4 AseCommand オブジェクトを作成して、そのパラメータを定義します。

次のコードは、Select コマンドと Update コマンドを作成して、Update コマンドのパラメータを定義します。

C# の場合：

```
adapter.SelectCommand = new AseCommand(
   "SELECT * FROM publishers WHERE pub_id > '9900'",
   conn );
adapter.UpdateCommand = new AseCommand(
   "UPDATE publishers SET pub_name = @pub_name, " +
   "city = @city, state = @state " +
   "WHERE pub_id = @pub_id", conn );
adapter.UpdateCommand.UpdatedRowSource =
   UpdateRowSource.None;
AseParameter parm = new AseParameter("@pub_id",
   AseDbType.Char, 4);
parm.SourceColumn = "pub_id";
parm.SourceVersion = DataRowVersion.Current;
adapter.UpdateCommand.Parameters.Add( parm );
parm = new AseParameter("@pub_name",
   AseDbType.VarChar, 40);
parm.SourceColumn = "pub_name";
parm.SourceVersion = DataRowVersion.Current;
adapter.UpdateCommand.Parameters.Add( parm );
parm = new AseParameter("@city",
```

```
   AseDbType.VarChar, 20);
parm.SourceColumn = "city";
parm.SourceVersion = DataRowVersion.Current;
adapter.UpdateCommand.Parameters.Add( parm );
parm = new AseParameter("@state",
   AseDbType.Char, 2);
parm.SourceColumn = "state";
parm.SourceVersion = DataRowVersion.Current;
adapter.UpdateCommand.Parameters.Add( parm );
```

Visual Basic .NET の場合：

```
adapter.SelectCommand = New AseCommand( _
  "SELECT * FROM publishers WHERE pub_id > '9900'",_
  conn )
adapter.UpdateCommand = New AseCommand( _
  "UPDATE publishers SET pub_name = @pub_name, " + _
  "city = @city, state = @state " + _
  "WHERE pub_id = @pub_id", conn )
adapter.UpdateCommand.UpdatedRowSource = _
  UpdateRowSource.None
Dim parm As New AseParameter("@pub_id", _
  AseDbType.Char, 4)
parm.SourceColumn = "pub_id"
parm.SourceVersion = DataRowVersion.Current
adapter.UpdateCommand.Parameters.Add( parm )
parm = New AseParameter("@pub_name", _
  AseDbType.VarChar, 40)
parm.SourceColumn = "pub_name"
parm.SourceVersion = DataRowVersion.Current
adapter.UpdateCommand.Parameters.Add( parm )
parm = New AseParameter("@city", _
  AseDbType.VarChar, 20)
parm.SourceColumn = "city"
parm.SourceVersion = DataRowVersion.Current
adapter.UpdateCommand.Parameters.Add( parm )
parm = New AseParameter("@state", _
  AseDbType.Char, 2)
parm.SourceColumn = "state"
parm.SourceVersion = DataRowVersion.Current
adapter.UpdateCommand.Parameters.Add( parm )
```

5　DataTable に Select 文の結果を書き込みます。

C# の場合：

```
DataTable dataTable = new DataTable( "publishers" );
int rowCount = adapter.Fill( dataTable );
```

Visual Basic .NET の場合：

```
Dim dataTable As New DataTable( "publishers" )
Dim rowCount As Integer = adapter.Fill( dataTable )
```

6 ローの更新値で DataTable を更新して、変更をデータベースに適用します。

C# の場合：

```
foreach ( DataRow row in dataTable.Rows )
{
   row[1] = ( string ) row[1] + "_Updated";
}
int recordsAffected = adapter.Update( dataTable );
```

Visual Basic .NET の場合：

```
Dim row as DataRow
For Each row in dataTable.Rows
   row(1) = row(1) + "_Updated"
Next
Dim recordsAffected As Integer =_
adapter.Update( dataTable )
```

7 結果をウィンドウのグリッドにバインドします。

C# の場合：

```
dataTable.Clear();
adapter.SelectCommand.CommandText =
   "SELECT * FROM publishers";
rowCount = adapter.Fill( dataTable );
dataGrid.DataSource = dataTable;
```

Visual Basic .NET の場合：

```
dataTable.Clear()
adapter.SelectCommand.CommandText = _
   "SELECT * FROM publishers";
rowCount = adapter.Fill( dataTable )
dataGrid.DataSource = dataTable
```

8 接続をクローズします。

C# の場合：

```
conn.Close();
```

Visual Basic .NET の場合：

```
conn.Close()
```

❖ AseDataAdapter オブジェクトを使用したテーブルからのローの削除

1　AseConnection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection( c_connStr );
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection( _
   c_connStr )
```

2　接続をオープンします。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

3　AseDataAdapter オブジェクトを作成します。

C# の場合：

```
AseDataAdapter adapter = new AseDataAdapter();
adapter.MissingMappingAction =
   MissingMappingAction.Passthrough;
adapter.MissingSchemaAction =
   MissingSchemaAction.AddWithKey;
```

Visual Basic .NET の場合：

```
Dim adapter As New AseDataAdapter()
adapter.MissingMappingAction = _
   MissingMappingAction.Passthrough
adapter.MissingSchemaAction = _
   MissingSchemaAction.AddWithKey
```

4　必要な AseCommand オブジェクトを作成して、必要なパラメータをすべて定義します。

次のコードは、Select コマンドと Delete コマンドを作成して、Delete コマンドのパラメータを定義します。

C# の場合：

```
adapter.SelectCommand = new AseCommand(
   "SELECT * FROM publishers WHERE pub_id > '9900'",
   conn );
adapter.DeleteCommand = new AseCommand(
   "DELETE FROM publishers WHERE pub_id = @pub_id",
   conn );
adapter.DeleteCommand.UpdatedRowSource =
   UpdateRowSource.None;
```

```
AseParameter parm = new AseParameter("@pub_id",
   AseDbType.Char, 4);
parm.SourceColumn = "pub_id";
parm.SourceVersion = DataRowVersion.Original;
adapter.DeleteCommand.Parameters.Add( parm );
```

Visual Basic .NET の場合：

```
adapter.SelectCommand = New AseCommand( _
   "SELECT * FROM publishers WHERE pub_id > '9900'", _
   conn )
adapter.DeleteCommand = New AseCommand( _
   "DELETE FROM publishers WHERE pub_id = @pub_id", conn )
adapter.DeleteCommand.UpdatedRowSource = _
   UpdateRowSource.None
Dim parm As New AseParameter("@pub_id", _
   AseDbType.Char, 4)
parm.SourceColumn = "pub_id"
parm.SourceVersion = DataRowVersion.Original
adapter.DeleteCommand.Parameters.Add( parm )
```

5 DataTable に Select 文の結果を書き込みます。

C# の場合：

```
DataTable dataTable = new DataTable( "publishers" );
int rowCount = adapter.Fill( dataTable );
```

Visual Basic .NET の場合：

```
Dim dataTable As New DataTable( "publishers" )
Dim rowCount As Integer = adapter.Fill( dataTable )
```

6 DataTable を変更して、データベースに変更を適用します。

C# の場合：

```
foreach ( DataRow row in dataTable.Rows )
{
   row.Delete();
}
int recordsAffected = adapter.Update( dataTable );
```

Visual Basic .NET の場合：

```
Dim row as DataRow
For Each row in dataTable.Rows
   row.Delete()
Next
Dim recordsAffected As Integer =_
   adapter.Update( dataTable )
```

7　結果をウィンドウのグリッドにバインドします。

C# の場合：

```
dataTable.Clear();
rowCount = adapter.Fill( dataTable );
dataGrid.DataSource = dataTable;
```

Visual Basic .NET の場合：

```
dataTable.Clear()
rowCount = adapter.Fill( dataTable )dataGrid.
DataSource = dataTable
```

8　接続をクローズします。

C# の場合：

```
conn.Close();
```

Visual Basic .NET の場合：

```
conn.Close()
```

AseDataAdapter スキーマ情報の取得

AseDataAdapter では、FillSchema メソッドを使用して DataSet 内の結果セットに関するスキーマ情報を取得できます。FillSchema メソッドは、標準の .NET DataTable オブジェクトを返します。このオブジェクトは、結果セット内のカラムすべての名前を提供します。

詳細については、「FillSchema メソッド」(127 ページ) を参照してください。

❖ *FillSchema* メソッドを使用した DataSet スキーマ情報の取得

1　AseConnection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection   conn = new AseConnection(
   c_connStr );
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection( _
   c_connStr )
```

2　接続をオープンします。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

3 目的の Select 文を使用して AseDataAdapter を作成します。このクエリの結果セットのスキーマが返されます。

C# の場合：

```
AseDataAdapter  adapter = new AseDataAdapter(
   "SELECT * FROM employee", conn );
```

Visual Basic .NET の場合：

```
Dim adapter As New AseDataAdapter( _
   "SELECT * FROM employee", conn )
```

4 スキーマを書き込む新しい DataTable オブジェクトを、ここでは “Table” という名前で作成します。

C# の場合：

```
DataTable dataTable = new DataTable( "Table" );
```

Visual Basic .NET の場合：

```
Dim dataTable As New DataTable( "Table" )
```

5 データ・ソースからのスキーマを DataTable に書き込みます。

C# の場合：

```
adapter.FillSchema( dataTable, SchemaType.Source );
```

Visual Basic .NET の場合：

```
adapter.FillSchema( dataTable, SchemaType.Source )
```

6 AseConnection オブジェクトをクローズします。

C# の場合：

```
conn.Close();
```

Visual Basic .NET の場合：

```
conn.Close()
```

7 DataSet をウィンドウのグリッドにバインドします。

C# の場合：

```
dataGrid.DataSource = dataTable;
```

Visual Basic .NET の場合：

```
dataGrid.DataSource = dataTable
```

プライマリ・キー値の取得

更新対象テーブルに自動インクリメント・プライマリ・キーがある場合、またはプライマリ・キー・プールからプライマリ・キーを取得する場合は、ストアド・プロシージャを使用して、データ・ソースで生成される値を取得できます。

AseDataAdapter を使用する場合、この手法を利用して、データ・ソースで生成されるプライマリ・キー値を DataSet のカラムに書き込むことができます。AseCommand オブジェクトでこの手法を使用するには、パラメータからキー・カラムを取得するか、DataReader を再オープンします。

例

次の例では、“adodotnet_primarykey” という名前のテーブルを使用します。このテーブルには、“id” と “name” という 2 つのカラムがあります。テーブルのプライマリ・キーは “id” で、自動インクリメント値を含む NUMERIC(8) です。name カラムは CHAR(40) です。

この例では、次のストアド・プロシージャを呼び出して、データベースから自動インクリメント・プライマリ・キー値を取得します。

```
create procedure sp_adodotnet_primarykey
@p_name char(40),
@p_id int output
as
begin
   insert into adodotnet_primarykey(name)
      VALUES(@p_name)
   select @p_id = @@identity
END
```

❖ AseCommand オブジェクトを使用した、自動インクリメント・プライマリ・キーを持つ新しいローの挿入

1　データベースに接続します。

C# の場合：

```
AseConnection conn = new AseConnection( c_connStr );
conn.Open();
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection( _
   c_connStr )
conn.Open()
```

2　DataTable に新しいローを挿入する AseCommand オブジェクトを新規作成します。次のコードでは、行 `int id1 = ( int ) parmId.Value;` によってローのプライマリ・キー値を検証します。

C# の場合：

```
AseCommand      cmd = conn.CreateCommand();
cmd.CommandText = "sp_adodotnet_primarykey";
cmd.CommandType = CommandType.StoredProcedure;
AseParameter parmId = new AseParameter(
```

```
   "@p_id", AseDbType.Integer);
parmId.Direction = ParameterDirection.Output;
cmd.Parameters.Add( parmId );
AseParameter parmName = new AseParameter(
   "@p_name", AseDbType.Char );
parmName.Direction = ParameterDirection.Input;
cmd.Parameters.Add( parmName );
parmName.Value = "R & D --- Command";
cmd.ExecuteNonQuery();
int id1 = ( int ) parmId.Value;
parmName.Value = "Marketing --- Command";
cmd.ExecuteNonQuery();
int id2 = ( int ) parmId.Value;
parmName.Value = "Sales --- Command";
cmd.ExecuteNonQuery();
int id3 = ( int ) parmId.Value;
parmName.Value = "Shipping --- Command";
cmd.ExecuteNonQuery();
int id4 = ( int ) parmId.Value;
```

Visual Basic .NET の場合：

```
Dim cmd As AseCommand = conn.CreateCommand()
cmd.CommandText = "sp_adodotnet_primarykey"
cmd.CommandType = CommandType.StoredProcedure
Dim parmId As New AseParameter("@p_id", _
   AseDbType.Integer)
parmId.Direction = ParameterDirection.Output
cmd.Parameters.Add( parmId )
Dim parmName As New AseParameter("@p_name", _
   AseDbType.Char)
parmName.Direction = ParameterDirection.Input
cmd.Parameters.Add(parmName )

parmName.Value = "R & D --- Command"
cmd.ExecuteNonQuery()
Dim id1 As Integer = parmId.Value
parmName.Value = "Marketing --- Command"
cmd.ExecuteNonQuery()
Dim id2 As Integer = parmId.Value
parmName.Value = "Sales --- Command"
cmd.ExecuteNonQuery()
Dim id3 As Integer = parmId.Value
parmName.Value = "Shipping --- Command"
cmd.ExecuteNonQuery()
dim id4 As Integer = parmId.Value
```

3　結果をウィンドウのグリッドにバインドして、変更をデータベースに適用します。

C# の場合：

```
cmd.CommandText = "select * from " +
   "adodotnet_primarykey";
cmd.CommandType = CommandType.Text;
AseDataReader dr = cmd.ExecuteReader();
dataGrid.DataSource = dr;
```

Visual Basic .NET の場合：

```
cmd.CommandText = "select * from " + _
   "adodotnet_primarykey"
cmd.CommandType = CommandType.Text
Dim dr As AseDataReader = cmd.ExecuteReader()
dataGrid.DataSource = dr
```

4　接続をクローズします。

C# の場合：

```
conn.Close();
```

Visual Basic .NET の場合：

```
conn.Close()
```

❖ AseDataAdapter オブジェクトを使用した、自動インクリメント・プライマリ・キーを持つ新しいローの挿入

1　新しい AseDataAdapter を作成します。

C# の場合：

```
AseConnection conn = new AseConnection(
   c_connStr );
conn.Open();
DataSet dataSet = new DataSet();
AseDataAdapter adapter = new AseDataAdapter();
adapter.MissingMappingAction =
   MissingMappingAction.Passthrough;
adapter.MissingSchemaAction =
   MissingSchemaAction.AddWithKey;
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection( _
   c_connStr )
conn.Open()
Dim dataSet As New DataSet()
Dim adapter As New AseDataAdapter()
adapter.MissingMappingAction = _
   MissingMappingAction.Passthrough
adapter.MissingSchemaAction = _
   MissingSchemaAction.AddWithKey
```

2 データと DataSet のスキーマを書き込みます。次のコードでは、AseDataAdapter.Fill メソッドによって SelectCommand を呼び出して、この操作を実行します。既存のレコードが不要な場合は、Fill メソッドと SelectCommand を使用せずに、DataSet を手動で作成することもできます。

C# の場合：

```
adapter.SelectCommand = new AseCommand(
   "select * from adodotnet_primarykey",
   conn );
```

Visual Basic .NET の場合：

```
adapter.SelectCommand = New AseCommand( _
   "select * from adodotnet_primarykey", conn )
```

3 新しい AseCommand を作成して、データベースからプライマリ・キー値を取得します。

C# の場合：

```
adapter.InsertCommand = new AseCommand(
   "sp_adodotnet_primarykey", conn );
adapter.InsertCommand.CommandType =
   CommandType.StoredProcedure;
adapter.InsertCommand.UpdatedRowSource =
   UpdateRowSource.OutputParameters;
AseParameter parmId = new AseParameter(
   "@p_id", AseDbType.Integer);
parmId.Direction = ParameterDirection.Output;
parmId.SourceColumn = "id";
parmId.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parmId );
AseParameter parmName = new AseParameter(
   "@p_name", AseDbType.Char);
parmName.Direction = ParameterDirection.Input;
parmName.SourceColumn = "name";
parmName.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parmName );
```

Visual Basic .NET の場合：

```
adapter.InsertCommand = new AseCommand( _
   "sp_adodotnet_primarykey", conn )
adapter.InsertCommand.CommandType = _
   CommandType.StoredProcedure
adapter.InsertCommand.UpdatedRowSource = _
   UpdateRowSource.OutputParameters
Dim parmId As New AseParameter( _
   "@p_id", AseDbType.Integer)
parmId.Direction = ParameterDirection.Output
parmId.SourceColumn = "id"
parmId.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parmId )
Dim parmName As New AseParameter( _
```

```
   "@p_name", AseDbType.Char)
parmName.Direction = ParameterDirection.Input
parmName.SourceColumn = "name"
parmName.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parmName )
```

4　DataSet に書き込みます。

C# の場合：

```
adapter.Fill( dataSet );
```

Visual Basic .NET の場合：

```
adapter.Fill( dataSet )
```

5　DataSet に新しいローを挿入します。

C# の場合：

```
DataRow row = dataSet.Tables[0].NewRow();
row[0] = -1;
row[1] = "R & D --- Adapter";
dataSet.Tables[0].Rows.Add( row );
row = dataSet.Tables[0].NewRow();
row[0] = -2;
row[1] = "Marketing --- Adapter";
dataSet.Tables[0].Rows.Add( row );
row = dataSet.Tables[0].NewRow();
row[0] = -3;
row[1] = "Sales --- Adapter";
dataSet.Tables[0].Rows.Add( row );
row = dataSet.Tables[0].NewRow();
row[0] = -4;
row[1] = "Shipping --- Adapter";
dataSet.Tables[0].Rows.Add( row );
```

Visual Basic .NET の場合：

```
Dim row As DataRow = dataSet.Tables(0).NewRow()
row(0) = -1
row(1) = "R & D --- Adapter"
dataSet.Tables(0).Rows.Add( row )
row = dataSet.Tables(0).NewRow()
row(0) = -2
row(1) = "Marketing --- Adapter"
dataSet.Tables(0).Rows.Add( row )
row = dataSet.Tables(0).NewRow()
row(0) = -3
row(1) = "Sales --- Adapter"
dataSet.Tables(0).Rows.Add( row )
row = dataSet.Tables(0).NewRow()
row(0) = -4
row(1) = "Shipping --- Adapter"
dataSet.Tables(0).Rows.Add( row )
```

6 DataSet に加えられた変更をデータベースに適用します。Update() メソッドが呼び出されると、プライマリ・キー値が、データベースから取得された値に変わります。

C# の場合：

```
adapter.Update( dataSet );
dataGrid.DataSource = dataSet.Tables[0];
```

Visual Basic .NET の場合：

```
adapter.Update( dataSet )
dataGrid.DataSource = dataSet.Tables(0)
```

ユーザが DataTable に新しいローを追加して Update メソッドを呼び出すと、AseDataAdapter は InsertCommand を呼び出して、追加された新しいローそれぞれのキー・カラムに出力パラメータをマップします。Update メソッドが呼び出されるのは 1 回だけですが、InsertCommand は、追加される新しいローそれぞれについて、Update メソッドによって必要な回数呼び出されます。

7 データベースへの接続をクローズします。

C# の場合：

```
conn.Close();
```

Visual Basic .NET の場合：

```
conn.Close()
```

BLOB の処理

長い文字列値またはバイナリ・データをフェッチする場合、データを分割してフェッチできるメソッドがあります。バイナリ・データには GetBytes メソッドを、文字列データには GetChars メソッドを使用します。これらを使用しない場合、BLOB データはデータベースからフェッチされるその他のデータと同様の方法で処理されます。

詳細については、「GetBytes メソッド」(134 ページ) と 「GetChars メソッド」(135 ページ) を参照してください。

❖ GetChars メソッドを使用した文字列を返すコマンドの発行

1 Connection オブジェクトを宣言して初期化します。

2 接続をオープンします。

3　SQL 文を定義して実行する Command オブジェクトを追加します。

C# の場合：

```
AseCommand cmd = new AseCommand(
   "select au_id, copy from blurbs", conn );
```

Visual Basic .NET の場合：

```
Dim cmd As New AseCommand( _
   "select au_id, copy from blurbs", conn)
```

4　ExecuteReader メソッドを呼び出して、DataReader オブジェクトを返します。

C# の場合：

```
AseDataReader reader = cmd.ExecuteReader();
```

Visual Basic .NET の場合：

```
Dim reader As AseDataReader = cmd.ExecuteReader()
```

次のコードは、結果セットから 2 つのカラムを読み込みます。最初のカラムは varchar で、2 番目のカラムは Text です。GetChars は、Text カラムから 1 度に 100 文字を読み込みます。

C# の場合：

```
int length = 100;
char[] buf = new char[ length ];
String au_id;
long dataIndex = 0;
long charsRead = 0;
long blobLength = 0;
while( reader.Read() )
{
   au_id = reader.GetString(0);
   do
   {
      charsRead = reader.GetChars(
         1, dataIndex, buf, 0, length);
      dataIndex += length;
      // do something with the chars read
      //.... some code
      //
      // reinitialize char array
      buf = new char[ length ];
   } while ( charsRead == length );
   blobLength = dataIndex + charsRead;
}
```

Visual Basic .NET の場合：

```
Dim length As Integer = 100
Dim buf(length) As Char
Dim au_id As String
Dim dataIndex As Long = 0
Dim charsRead As Long = 0
Dim blobLength As Long = 0
While reader.Read()
   au_id = reader.GetString(0)
   Do
      charsRead = reader.GetChars( _
         1, dataIndex, buf, 0, length)
      dataIndex = dataIndex + length
      ' do something with the data read
      '
      '  use code
      '
      ' reinitialize the char array
      ReDim buf(length)
   Loop While (charsRead = length)
   blobLength = dataIndex + charsRead
End While
```

5 DataReader オブジェクトと Connection オブジェクトをクローズします。

C# の場合：

```
reader.Close();
conn.Close();
```

Visual Basic .NET の場合：

```
reader.Close()
conn.Close()
```

時刻値の取得

.NET Framework には、Time 構造体がありません。Adaptive Server で時刻値をフェッチする場合は、GetDateTime() メソッドを使用してください。このメソッドを使用して、.NET Framework の DateTime オブジェクトとしてデータを返します。

❖ ***GetDateTime* メソッドを使用した時刻値の変換**

1 Connection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection(
   c_connStr );
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection( _
   c_connStr )
```

2 接続をオープンします。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

3 SQL 文を定義して実行する Command オブジェクトを追加します。

C# の場合：

```
AseCommand cmd = new AseCommand(
   "SELECT title_id, title, pubdate FROM titles",
   conn );
```

Visual Basic .NET の場合：

```
Dim cmd As New AseCommand( _
   "SELECT title_id, title, pubdate FROM titles", _
   conn)
```

4 ExecuteReader メソッドを呼び出して、DataReader オブジェクトを返します。

C# の場合：

```
AseDataReader reader = cmd.ExecuteReader();
```

Visual Basic .NET の場合：

```
Dim reader As AseDataReader = cmd.ExecuteReader()
```

次のコードは、GetDateTime メソッドを使用して、DateTime 値を返します。

C# の場合：

```
while ( reader.Read() )
{
   String tid = reader.GetString(0);
   String title = reader.GetString(1);
   DateTime time = reader.GetDateTime(2);
   // do something with the data
}
```

Visual Basic .NET の場合：

```
While reader.Read()
   Dim tid As String = reader.GetString(0)
   Dim title As String = reader.GetString(1)
   Dim time As DateTime = reader.GetDateTime(2)
   ' do something with the data....
End While
```

5 DataReader オブジェクトと Connection オブジェクトをクローズします。

C# の場合：

```
reader.Close();
conn.Close();
```

Visual Basic .NET の場合：

```
reader.Close()
conn.Close()
```

ストアド・プロシージャの使用

Adaptive Server ADO.NET Data Provider ではストアド・プロシージャを使用できます。結果セットを返すストアド・プロシージャを呼び出す場合は、ExecuteReader メソッドを使用します。

注意 ストアド・プロシージャを使用してデータベースからデータを取得する場合、ストアド・プロシージャから出力パラメータ値と結果セットの両方が返されると、結果セットはリセットされ、出力パラメータ値が参照されると同時に結果セットのローは参照できなくなります。Sybase では、このような場合、結果セットのローすべてを参照して使い終わるまで、参照側の出力パラメータ値を最後までそのままにしておくことをおすすめします。

結果セットを返さないストアド・プロシージャを呼び出す場合は、ExecuteNonQuery メソッドを使用します。単一の値のみを返すストアド・プロシージャを呼び出す場合は、ExecuteScalar メソッドを使用します。

ストアド・プロシージャでパラメータが必要な場合は、対応する AseParameter オブジェクトを作成します。CommandType を StoredProcedure に指定した場合は、CommandText をストアド・プロシージャの名前に設定してください。次に例を示します。

```
sp_producttype
```

Parameter オブジェクトの詳細については、「AseParameter クラス」(153 ページ) を参照してください。

❖ ストアド・プロシージャの実行

1　AseConnection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection(
   c_connStr );
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection( _
   c_connStr )
```

2　接続をオープンします。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

3　SQL 文を定義して実行する AseCommand オブジェクトを追加します。次のコードは、CommandType プロパティを使用して、コマンドがストアド・プロシージャであることを識別します。

C# の場合：

```
AseCommand cmd = new AseCommand(
   "titleid_proc", conn );
cmd.CommandType = CommandType.StoredProcedure;
```

Visual Basic .NET の場合：

```
Dim cmd As New AseCommand( _
   "titleid_proc", conn )
cmd.CommandType = CommandType.StoredProcedure
```

4 AseParameter オブジェクトを追加して、ストアド・プロシージャのパラメータを定義します。ストアド・プロシージャに必要なパラメータそれぞれに、新しく AseParameter オブジェクトを作成してください。

C# の場合：

```
AseParameter param = cmd.CreateParameter();
param.ParameterName = "@title_id";
param.AseDbType = AseDbType.VarChar;
param.Direction = ParameterDirection.Input;
param.Value = "BU";
cmd.Parameters.Add( param );
```

Visual Basic .NET の場合：

```
Dim param As AseParameter = cmd.CreateParameter()
param.ParameterName = "@title_id"
param.AseDbType = AseDbType.VarChar
param.Direction = ParameterDirection.Input
param.Value = "BU"
cmd.Parameters.Add( param )
```

Parameter オブジェクトの詳細については、「AseParameter クラス」(153 ページ) を参照してください。

5 ExecuteReader メソッドを呼び出して、DataReader オブジェクトを返します。Get メソッドを使用して、結果を任意のデータ型で返します。

C# の場合：

```
AseDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
   string title = reader.GetString(0);
   string id = reader.GetString(1);
   decimal price = reader.GetDecimal(2);
   // do something with the data....
}
```

Visual Basic .NET の場合：

```
Dim reader As AseDataReader = cmd.ExecuteReader()
While reader.Read()
   Dim title As String = reader.GetString(0)
   Dim id As String = reader.GetString(1)
   Dim price As Decimal = reader.GetDecimal(2)
   ' do something with the data....
End While
```

6　AseDataReader オブジェクトと AseConnection オブジェクトをクローズします。

C# の場合：

```
reader.Close();
conn.Close();
```

Visual Basic .NET の場合：

```
reader.Close()
conn.Close()
```

ストアド・プロシージャを呼び出すもう 1 つの方法

次のように、呼び出し構文を使用してストアド・プロシージャを呼び出すこともできます。この構文は ODBC および JDBC と互換性があります。次に例を示します。

```
AseCommand cmd = new AseCommand("{ call sp_product_info(?) }",
conn);
```

この場合、コマンドの種類を CommandType.StoredProcedure に設定しないでください。この構文は、名前付きパラメータを使用しないで、AseCommand.NamedParameters プロパティを “false” に設定している場合に使用できます。

結果セットまたは単一の値を返すストアド・プロシージャの呼び出し方法の詳細については、「AseCommand オブジェクトを使用したデータの取得」(37 ページ) を参照してください。

結果セットを返さないストアド・プロシージャの呼び出し方法の詳細については、「AseCommand オブジェクトを使用したローの挿入、更新、削除」(42 ページ) を参照してください。

トランザクション処理

Adaptive Server ADO.NET Data Provider では、AseTransaction オブジェクトを使用して複数の文をグループ化できます。それぞれのトランザクションは、変更を永続的にデータベースに適用する COMMIT、またはトランザクションの操作すべてを取り消す ROLLBACK で終了します。トランザクションが完了した後、さらに変更を行う場合は、新しく AseTransaction オブジェクトを作成してください。ODBC や Embedded SQL の場合は動作が異なり、トランザクションがクローズされるまで、COMMIT や ROLLBACK が実行された後もトランザクションが保持されます。

トランザクションを作成しない場合、Adaptive Server ADO.NET Data Provider はデフォルトで autocommit モードで動作します。insert、update、または delete が実行されるたびに Commit が暗黙的に実行され、操作が完了すると変更がデータベースに反映されます。この場合、変更はロールバックできません。

AseTransaction オブジェクトの詳細については、「AseTransaction クラス」(165 ページ) を参照してください。

トランザクションの独立性レベルの設定

トランザクションの開始時に独立性レベルを指定するように選択できます。この独立性レベルは、そのトランザクション内で実行されるすべてのコマンドに適用されます。

独立性レベルの詳細については、Adaptive Server Enterprise の『パフォーマンス＆チューニング・ガイド』を参照してください。

Select 文を入力したときに Adaptive Server で使用されるロックは、トランザクションの独立性レベルに応じて異なります。

次の例は、AseTransaction オブジェクトを使用して SQL 文を発行してからロールバックを実行します。このトランザクションは独立性レベル 2 (RepeatableRead) を使用し、変更中のローに Write ロックを適用して、他のデータベース・ユーザがローを更新できないようにします。

❖ **AseTransaction オブジェクトを使用したコマンドの発行**

1 AseConnection オブジェクトを宣言して初期化します。

C# の場合：

```
AseConnection conn = new AseConnection(
   c_connStr );
```

Visual Basic .NET の場合：

```
Dim conn As New AseConnection( _
   c_connStr )
```

2 接続をオープンします。

C# の場合：

```
conn.Open();
```

Visual Basic .NET の場合：

```
conn.Open()
```

3 「T シャツ (Tee shirts)」の価格を変更する SQL 文を発行します。

C# の場合：

```
string stmt = "update product " +
   " set unit_price = 2000.00 " +
   " where name = 'Tee shirt'";
```

Visual Basic .NET の場合：

```
Dim stmt As String = "update product " + _
   " set unit_price = 2000.00 " + _
   " where name = 'Tee shirt'"
```

4　Command オブジェクトを使用して SQL 文を発行する AseTransaction オブジェクトを作成します。

トランザクションを使用することで、独立性レベルを指定できるようになります。この例では独立性レベル 2 (RepeatableRead) を使用して、他のデータベース・ユーザがローを更新できないようにします。

C# の場合：

```
AseTransaction trans = conn.BeginTransaction(
   IsolationLevel.RepeatableRead );
AseCommand cmd = new AseCommand( stmt, conn, trans );
int rows = cmd.ExecuteNonQuery();
```

Visual Basic .NET の場合：

```
Dim trans As AseTransaction = _
   conn.BeginTransaction( _
   IsolationLevel.RepeatableRead )
Dim cmd As New AseCommand( _
   stmt, conn, trans )
Dim rows As Integer = cmd.ExecuteNonQuery()
```

5　変更をロールバックします。

C# の場合：

```
trans.Rollback();
```

Visual Basic .NET の場合：

```
trans.Rollback()
```

AseTransaction オブジェクトを使用すると、データベースに対する変更をコミットしたりロールバックしたりできます。トランザクションを使用しない場合、Adaptive Server ADO.NET Data Provider は autocommit モードで動作するため、データベースに加えた変更をロールバックできなくなります。変更を永続的に残す場合は、次のように指定します。

C# の場合：

```
trans.Commit();
```

Visual Basic .NET の場合：

```
trans.Commit()
```

6　AseConnection オブジェクトをクローズします。

C# の場合：

```
conn.Close();
```

Visual Basic .NET の場合：

```
conn.Close()
```

エラー処理

アプリケーションは、ADO.NET エラーも含め、発生するすべてのエラーを処理できるように設計する必要があります。コード内の ADO.NET エラーの処理方法は、アプリケーションの他のエラーを処理する場合と同じです。

実行時にエラーが発生すると、Adaptive Server ADO.NET Data Provider は AseException オブジェクトを返します。各 AseException オブジェクトは AseError オブジェクトのリストで構成され、これらのエラー・オブジェクトにはエラー・メッセージとコードが組み込まれています。この他に、IndexOutOfRangeException や NotSupportedException などの例外も利用できます。

エラーは、変更をデータベースに適用するときに発生する競合とは異なります。アプリケーションには、競合が発生した場合に正しい値を計算したりログに記録したりする処理が必要です。

Adaptive Server ADO.NET Data Provider のエラー処理の例

Simple サンプル・プロジェクトの例を次に示します。実行時に発生したエラーや Adaptive Server ADO.NET Data Provider オブジェクトのエラーはすべて、メッセージ・ボックスに表示されます。次のコードは、エラーを検出してメッセージを表示します。

C# の場合：

```
catch( AseException ex )
{
   MessageBox.Show( ex.Message );
}
```

Visual Basic .NET の場合：

```
Catch ex As AseException
   MessageBox.Show(ex.Message)
End Try
```

接続エラー処理の例

Table Viewer サンプル・プロジェクトの例を次に示します。アプリケーションがデータベースへの接続を試行しているときにエラーが発生した場合、次のコードは try-catch ブロックを使用してエラーを検出し、メッセージを表示します。

C# の場合：

```
try
{
   _conn = new AseConnection(
      txtConnectString.Text );
   _conn.Open();
}
catch( AseException ex )
{
   MessageBox.Show(ex.Message, "Failed to connect");
}
```

Visual Basic .NET の場合：

```
Try
   Dim  _conn As New AseConnection( _
      txtConnectString.Text )
   conn.Open()
Catch ex As AseException
   MessageBox.Show(ex.Message, "Failed to connect")
End Try
```

その他のエラー処理の例については、「Simple サンプル・プロジェクトの理解」(13 ページ) と 「Table Viewer サンプル・プロジェクトの理解」(18 ページ) を参照してください。

エラー処理の詳細については、「AseException クラス」(150 ページ) と 「AseError クラス」(147 ページ) を参照してください。

パフォーマンスの考慮事項

この項では、Adaptive ServerADO.NET Data Provider を使用してアプリケーションの開発と展開を行う上で役立つヒントを紹介します。

DbType.String と DbType.AnsiString

DbType.String と DbType.AnsiString では、ともに文字データが処理されます。ただし、この両データ型の処理方法はそれぞれに異なるので、不適切なデータ型を使用すると、アプリケーションのパフォーマンスに悪影響を及ぼす可能性があります。DbType.String では、パラメータは 2 バイトの Unicode 値として識別されて、サーバに送信されます。DbType.AnsiString では、パラメータはマルチバイトの文字列として送信されます。不要な文字列変換を避けるため、次のように使用します。

- char または varchar のカラムおよびパラメータに対しては、DbType.AnsiString を使用します。
- unichar または univarchar のカラムおよびパラメータに対しては、DbType.String を使用します。

第 4 章

Adaptive Server の高度な機能

この章では、ADO.NET Data Provider で使用できる Adaptive Server の高度な機能について説明します。

サポートされている Adaptive Server クラスタ・エディションの機能

この項では、クラスタ・エディション環境をサポートする ASE ADO.NET Driver の機能について説明します。クラスタ・エディション環境では、複数の Adaptive Server が共有ディスクのセットと高速プライベート相互接続に接続します。この場合、複数の物理ホストと論理ホストを使用して、Adaptive Server を拡張できます。

クラスタ・エディションの詳細については、『Cluster ユーザーズ・ガイド』を参照してください。

ログインのリダイレクト

クラスタ・エディション環境では一般に、常にサーバ間で処理負荷の不均衡が発生しています。ビジー状態のサーバに対してクライアント・アプリケーションが接続した場合、ログインのリダイレクト機能によって、サーバの負荷バランスが調整されます。具体的には、クラスタ内の負荷が少ない別サーバに対して、クライアント接続がリダイレクトされます。ログインのリダイレクトが発生するのはログイン・シーケンス中であり、リダイレクトが発生したことは、クライアント・アプリケーションには通知されません。ログインのリダイレクト機能をサポートしているサーバに対してクライアント・アプリケーションが接続した時点で、この機能は自動的に有効になります。

注意 クライアントをリダイレクトするように設定されているサーバに対してクライアント・アプリケーションが接続すると、ログインに時間がかかる場合があります。これは、クライアント接続が別サーバにリダイレクトされるたびに、ログイン・プロセスが再開されるからです。

接続マイグレーション

接続マイグレーション機能を使用すると、クラスタ・エディション環境内のサーバは動的に負荷を分散できます。さらに、既存のクライアント接続とそのコンテキストをクラスタ内の別サーバにシームレスにマイグレートできます。この機能によって、クラスタ・エディション環境では、最適なリソース配分と処理時間の短縮が実現します。サーバ間のマイグレーションはシームレスに行われるので、接続マイグレーション機能は、可用性の高い「ダウン時間ゼロ」の環境を構築する場合にも役立ちます。接続マイグレーション機能をサポートしているサーバに対してクライアント・アプリケーションが接続した時点で、この機能は自動的に有効になります。

注意 接続マイグレーション中には、コマンドの実行に時間がかかる場合があります。状況に応じて、コマンドのタイムアウト値を増やすことをおすすめします。

接続フェールオーバ

接続フェールオーバ機能を使用すると、停電やソケットの障害など、予想外の原因でプライマリ・サーバが使用不可になった場合に、クライアント・アプリケーションは接続先を別の Adaptive Server に切り替えることができます。クラスタ環境では、クライアント・アプリケーションは動的なフェールオーバ・アドレスを使用して、複数のサーバに対して何度もフェールオーバできます。

高可用性に対応したシステムでは、フェールオーバ・ターゲットの候補をクライアント・アプリケーションにあらかじめ設定しておく必要はありません。Adaptive Server は、クラスタ・メンバシップ、論理クラスタの使用状況、負荷分散などに基づいて、最適なフェールオーバ・リストを常にクライアントに提供します。クライアントは、フェールオーバ時にフェールオーバ・リストの順序付けを参照して、再接続します。ドライバがサーバに正常に接続した場合は、返されたリストに基づいて、ホスト値のリストが内部的に更新されます。それ以外の場合は、接続失敗例外が発生します。

クラスタ・エディションの接続フェールオーバの有効化

クラスタ・エディションの接続フェールオーバを有効にするには、HASession 接続文字列プロパティを 1 に設定します。次に例を示します。

```
Data Source=server1;Port=port1;User ID=sa;Password=;
Initial Catalog=sdc;HASession=1;
AlternateServers=server2:port2,...,serverN:portN;
```

この例では、Data Source はプライマリ・サーバとポートを定義します。Sybase が提供する ADO.NET Provider は、最初にプライマリ・サーバへ接続します。接続に失敗した場合は、AlternateServers に列挙されているサーバへの接続を順番に実行します。接続に成功するか、リストの末尾に達するまで、この処理が繰り返されます。

注意　接続文字列で指定された代替サーバのリストは、初期接続時にのみ使用されます。使用可能なインスタンスとの接続の確立後、高可用性をサポートしているクライアントは、最適なフェールオーバ・ターゲットを含む最新のリストをサーバから受信します。この新しいリストは、指定されたリストを上書きします。

分散トランザクションの使用

Adaptive Server ADO.NET Data Provider を使用し、それを 2 フェーズ・コミット・トランザクションに含めることができます。この機能では、分散トランザクションを管理する .NET Enterprise Services を使用する必要があります。

Enterprise Services を使用するプログラミング

アンマネージ・コード内のサービスは、COM+ サービスと呼ばれます。COM+ サービス・インフラストラクチャは、マネージ・コードとアンマネージ・コードからアクセスできます。.NET では、これらのサービスは Enterprise Services と呼ばれます。ADO.NET を使用する Enterprise Services 内でトランザクションを扱うのは簡単です。

❖ **Enterprise Services を使用するプログラミング**

1 *System.EntrepriseService.ServicedComponent* からコンポーネントを抽出します。

2 必要なサービスとそのオプションを指定するための独自の属性 (Transaction、AutoComplete など) を指定します。属性の全リストについては、Enterprise Services のマニュアルを参照してください。

注意 .NET トランザクション属性の Timeout Option には、明示的に *-1* または非常に大きな数を設定する必要があります。.NET のマニュアルでは、ADO.NET トランザクションのデフォルトのタイムアウト値が *0* であることが記載されていますが、これはタイム・アウトしないことを意味します。ただし、実際には即時にトランザクションがタイムアウトし、トランザクション全体がロール・バックされます。

---

3 アセンブリを署名および構築します。

4 アセンブリを登録します。

分散トランザクションでの接続プロパティのサポート

次に、分散トランザクションのサポート時に使用する接続プロパティを示します。

- 分散トランザクション・プロトコル (DistributedTransactionProtocol) – 分散トランザクション、XA インタフェース標準、MS DTC OLE ネイティブ・プロトコルをサポートするために使用するプロトコルを指定するには、接続文字列でプロパティ DistributedTransactionProtocol=*OLE* ネイティブ・プロトコルを設定します。デフォルトのプロトコルは *XA* です。
- 密結合トランザクション (TightlyCoupledTransaction) － 2 つのリソース・マネージャを使用する分散トランザクションで同一の Adaptive Server サーバを指定すると、「密結合トランザクション」と呼ばれる状態になります。この場合、このプロパティを *1* に設定していないと分散トランザクションが失敗することがあります。

  つまり、同一の Adaptive Server サーバに対して 2 つのデータベース接続をオープンしてから、オープンした接続を同一の分散トランザクションに登録する場合は、TightlyCoupledTransaction=*1* を設定する必要があります。
- 登録 － AseConnection オブジェクトは、トランザクションがアクティブであることを特定した場合に、既存の分散トランザクションに自動的に登録します。接続を開くか、接続プールから取得したときに、自動トランザクション登録が行われます。AseConnection の接続文字列パラメータとして Enlist=0 を指定することにより、この自動登録機能を無効にできます。

自動登録が無効である場合、既存トランザクションへの参照である ITransaction パラメータとともに AseConnection の EnlistDistributedTransaction メソッドを呼び出すことにより、既存の分散トランザクションに登録できます。EnlistDistributedTransaction を呼び出した後、AseConnection のこのインスタンスを使用したすべての更新は、このグローバル・トランザクションの一部として適用されます。したがって、グローバル・トランザクションがコミットまたはロール・バックされると、それに伴ってこのトランザクションもコミットまたはロール・バックされます。

注意 AseConnection オブジェクトは、EnlistDistributedTransaction を呼び出す前に開いていることが必要です。

ビジネス・オブジェクトをプールする場合、EnlistDistributedTransaction を使用できます。ビジネスオブジェクトが開いている接続とともにプールされている場合、その接続が開いているか、または接続プールから取得される場合に限り、自動トランザクション登録が行われます。プールされたビジネス・オブジェクトにより複数のトランザクションが実行される場合、そのオブジェクトの開いている接続は新しく開始したトランザクションで自動的に登録されません。この場合は、AseConnection の自動トランザクション登録を無効にした後、EnlistDistributedTransaction を使用して AseConnection をトランザクションに登録できます。

警告！ AseConnection が BeginTransaction を使用するか、または AseCommand とともに BEGIN TRANSACTION 文を明示的に実行することによりすでにトランザクションを開始している場合、EnlistDistributedTransaction は例外を返します。

ディレクトリ・サービス

ディレクトリ・サービスを使用すると、Adaptive Server ADO.NET Data Provider は中央にある LDAP サーバから接続やその他の情報を取得して Adaptive Server サーバに接続できます。ここでは、DSURL (Directory Service URL) を使用して、データを取得する LDAP サーバを示します。

ディレクトリ・サービスとしての LDAP

LDAP (Lightweight Directory Access Protocol) は、ディレクトリ・サービスへの業界標準のアクセス方法です。ディレクトリ・サービスを使用すると、コンポーネントは LDAP サーバから情報を DN (識別名) で検索できます。LDAP サーバは、企業またはネットワーク上で使用されるサーバ、ユーザ、ソフトウェアの情報を格納したり管理したりします。

LDAP サーバは、Adaptive Server やクライアントが動作しているプラットフォームとは別のプラットフォームに配置できます。LDAP では通信プロトコル、およびクライアント／サーバ間で交換されるメッセージのコンテンツを定義します。LDAP サーバに格納され、取得が可能な情報は、次のとおりです。

- Adaptive Server に関する情報 (IP アドレス、ポート番号、ネットワーク・プロトコルなど)
- セキュリティ・メカニズムとフィルタ
- 高可用性コンパニオン・サーバ名

詳細については、Adaptive Server Enterprise の『システム管理ガイド』を参照してください。

LDAP サーバの設定時に、次のアクセス制限を指定できます。

- 匿名認証 － すべてのユーザがあらゆる情報にアクセスできます。
- ユーザ名とパスワードによる認証 － Data Provider は、DSURL または ConnectString の *DSPrincipal* と *DSPassword* プロパティに指定されたユーザ名とパスワードを使用します。

ディレクトリ・サービスの使用

ディレクトリ・サービスを使用するには、ConnectString に次のプロパティを追加します。

```
DSURL= ldap://SYBLDAP:389/dc=sybase,dc=com??one?sybase
Servername=MANGO
```

URL は LDAP URL で、LDAP ライブラリを使用して URL を解決します。

LDAP サーバの高可用性をサポートするため、DSURL は複数の URL を受け入れます。各 URL は次のようにセミコロンで区切ります。たとえば、次のようにします。

```
DSURL={ldap://SYBLDAP:389/dc=sybase,dc=com??one?sybaseServern
ame=MANGO;
ldap://SYBLDAP1:389/dc=sybase,dc=com??one?sybaseServername=MA
NGO}
```

DSURL は次のように指定します。

```
ldap://hostport/dn[?attrs[?scope[?filter[?userdn?userpass]]]]
```

パラメータの意味は次のとおりです。

- *hostport* は、次のような、ホスト名とオプションの `portnumber` です。`SYBLDAP1:389`
- *dn* は、`dc=sybase,dc-com` などの検索ベースです。
- *attrs* は、LDAP に要求される属性のカンマ区切りリストです。これはブランクにします。Data Provider はすべての属性を必要とします。
- *scope* は、次の 3 つの文字列のいずれかになります。
  - `base` (デフォルト) − ベースを検索します。
  - `one` − 直下のディレクトリを検索します。
  - `sub` − サブ・ツリーを検索します。
- *filter* は検索フィルタです。通常は、sybaseServername です。ここをブランクにする場合は、Data Source または ConnectionString の Server Name プロパティを設定します。
- *userdn* は、ユーザの識別名 (DN: Distinguished Name) です。LDAP サーバが匿名ログインをサポートしていない場合は、ここでユーザの DN を設定するか、ConnectionString の DSPrincipal プロパティを設定します。
- *userpass* はパスワードです。LDAP サーバが匿名ログインをサポートしていない場合は、ここでパスワードを設定するか、ConnectionString の DSPassword プロパティを設定します。

パスワードの暗号化

Adaptive Server ADO.NET Data Provider はデフォルトで、ネットワークを介してプレーン・テキストのパスワードを Adaptive Server に送信して認証を求めます。ただし、Adaptive Server ADO.NET Data Provider は、パスワードの対称／非対称暗号化もサポートしています。この機能を使用すると、デフォルトの動作を変更し、パスワードを暗号化してからネットワークに送信できます。

対称暗号化メカニズムでは、パスワードの暗号化と復号化に同じキーが使用されます。これに対して、非対称暗号化メカニズムでは、暗号化にはパブリック・キー、復号化には別のプライベート・キーが使用されます。プライベート・キーはネットワークを介して共有されないため、非対称暗号化の方が対称暗号化よりも安全であると考えられます。パスワードの暗号化が有効になっていて、サーバが非対称暗号化をサポートしている場合、非対称暗号化が対称暗号化の代わりに使用されます。

注意 非対称暗号化を使用すると、ログインに若干の遅延が発生する可能性があります。これは、非対称暗号化に追加の処理時間が必要なためです。

パスワードの暗号化の有効化

パスワードの暗号化を有効にするには、EncryptPassword 接続プロパティを設定する必要があります。この接続プロパティでは、パスワードが暗号化フォーマットで転送されるかどうかを指定します。パスワードの暗号化が有効になっていると、ログインがネゴシエートされた場合にのみ、パスワードはネットワークに送信されます。パスワードは最初に暗号化されてから送信されます。EncryptPassword の値は次のとおりです。

- 0 − プレーン・テキスト形式のパスワードを使用します。これはデフォルトの値です。
- 1 − 暗号化されたパスワードを使用します。サポートされていない場合、エラー・メッセージを返します。
- 2 − 暗号化されたパスワードを使用します。サポートされていない場合、プレーン・テキスト形式のパスワードを使用します。

注意 非対称暗号化を使用するには、非対称暗号化をサポートするサーバ (Adaptive Server 15.0.2 など) が必要です。

例

この例では、ログインがネゴシエートされ、パスワードの暗号化と送信が行われるまで、ネットワークに sapass は送信されません。

```
AseConnection.ConnectionString=
”Data Source=MANGO;" +
   "Port = 5000;" +
   "Database=pubs2;" +
   "UID=sa;" +
   "PWD=sapass;" +
   "EncryptPassword=1;”;
```

SSL の使用

SSL (Secure Sockets Layer) は、クライアントとサーバ間、およびサーバ同士の接続において、ワイヤ・レベルまたはソケット・レベルで暗号化されたデータを送信する業界標準です。

SSL ハンドシェイク

サーバとクライアントが、安全な暗号化セッションをネゴシエートして合意してから、SSL 接続が確立されます。これは、「SSL ハンドシェイク」と呼ばれています。クライアント・アプリケーションが接続を要求すると、SSL 対応サーバが証明書を提示し、ID を証明してから、データを送信します。基本的に、SSL ハンドシェイクは次の手順によって構成されています。

1　クライアントがサーバに接続要求を送信します。要求には、クライアントがサポートしている SSL (または TLS: Transport Layer Security) オプションが含まれています。

2　サーバは、証明書とサポートされている CipherSuite のリストを返します。これには、SSL/TLS サポート・オプション、キー交換で使用されるアルゴリズム、デジタル署名が含まれます。

3　クライアントとサーバの両者が 1 つの CipherSuite について合意すると、安全で暗号化されたセッションが確立されます。

SSL ハンドシェイクと SSL/TLS プロトコルの詳細については、Internet Engineering Task Force Web site (http://www.ietf.org) を参照してください。

パフォーマンス

安全なセッションを確立する場合、追加のオーバヘッドが発生します。これは、暗号化によってデータ・サイズが増加することに加えて、情報の暗号化と復号化に伴う計算が必要になるからです。一般に、SSL ハンドシェイク中に生じる I/O の増加によって、ユーザ・ログインにかかる時間が 10 ～ 20 倍になることがあります。

CipherSuite

SSL ハンドシェイク中に、クライアントとサーバは、CipherSuite を介して共通のセキュリティ・プロトコルをネゴシエートします。CipherSuite は、SSL プロトコルで使用されるキー交換アルゴリズム、ハッシュ方式、暗号化方式の優先順位リストです。CipherSuite の詳細については、IETF の Web サイト IETF organization Web site (http://www.ietf.org) を参照してください。

デフォルトでは、クライアントとサーバの両方がサポートしている最も強力な CipherSuite は、SSL ベースのセッションに使用される CipherSuite です。サーバ接続属性は、接続文字列か、LDAP などのディレクトリ・サービスによって指定されます。

Adaptive Server ADO.NET Data Provider と Adaptive Server は、SSL Plus ライブラリ API と暗号エンジンである Security Builder (両方とも Certicom Corp 製) で使用可能な CipherSuite をサポートしています。

注意　次に示す CipherSuite のリストは TLS 仕様に準拠しています。TLS は、SSL 3.0 を拡張したものであり、SSL バージョン 3.0 CipherSuite の別名です。

Adaptive Server ADO.NET Data Provider でサポートしている CipherSuite は、次のとおりです。

- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_RC4_128_SHA
- TLS_RSA_WITH_RC4_128_MD5
- TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- TLS_DHE_DSS_WITH_RC4_128_SHA
- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_DES_CBC_SHA
- TLS_DHE_DSS_WITH_DES_CBC_SHA
- TLS_DHE_RSA_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
- TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA
- TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT_WITH_RC4_40_MD5
- TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
- TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA

Adaptive Server ADO.NET Data Provider の SSL

SSL には、次のセキュリティ・レベルがあります。

- SSL セッションが確立されると、ユーザ名とパスワードが暗号化された安全で接続によって送信されます。
- SSL 対応サーバへの接続を確立すると、サーバは接続対象のサーバであることを自己認証し、暗号化された SSL セッションが開始され、データが送信されます。
- サーバ証明書のデジタル署名を比較して、サーバから受信した情報が転送中に変更されたかどうかを判断します。

証明書によるサーバの検証

Adaptive Server ADO.NET Data Provider が SSL 対応サーバにクライアント接続する場合、サーバは証明書ファイルが必要です。これは、サーバの証明書と暗号化されたプライベート・キーで構成されます。また、証明書は署名/認証局 (CA: Certification Authority) によってデジタル署名されている必要もあります。Adaptive Server ADO.NET Data Provider のクライアント・アプリケーションが Adaptive Server へのソケット接続を確立する方法は、既存のクライアント接続の確立方法と同じです。ネットワーク・トランスポート・レベルの接続コールがクライアント側で完了し、承認コールがサーバ側で完了すると、ソケット上で SSL ハンドシェイクが行われ、その後でユーザ・データが送信されます。

SSL 対応サーバに正しく接続するには、次のことが必要です。

1　クライアント・アプリケーションが接続要求を行った場合、SSL 対応サーバは証明書を提示しなければなりません。

2　クライアント・アプリケーションは、証明書に署名した CA を認識しなければなりません。
「信頼された」CA すべてを含んだリストは、次に示す信頼されたルート・ファイルにあります。

詳細については、『Open Client Client-Library/C リファレンス・マニュアル』を参照してください。

信頼されたルート・ファイル

既知で信頼された CA のリストは、信頼されたルート・ファイルに保管されています。エンティティ (クライアント・アプリケーション、サーバ、ネットワーク・リソースなど) に既知の CA の証明書がある以外は、信頼されたルート・ファイルは証明書ファイルのフォーマットと同じです。システム・セキュリティ担当者が、標準 ASCII テキスト・エディタを使って、信頼された CA を追加したり、削除したりします。

アプリケーション・プログラムでは、ConnectString の TrustedFile=*trusted file path* プロパティを使用して、信頼されたルート・ファイルの位置を指定します。最も一般的に使用される CA (Thawte、Entrust、Baltimore、VeriSign、RSA) が記載された信頼されるルート・ファイルは *%SYBASE%¥ini¥trusted.txt* にインストールされています。

SSL 接続の有効化

Data Provider で SSL を有効化するには、ConnectionString プロパティに Encryption=ssl; TrustedFile=<信頼されたファイル> を追加します。これで、AseConnection が Adaptive Server サーバと SSL 接続をネゴシエートするようになります。次に例を示します。

```
AseConnection.ConnectionString=
     "Data Source=MANGO;" +
     "Port = 5000;" +
     "Database=pubs2;" +
     "UID=sa;" +
     "PWD=sapass;" +
     "Encryption=ssl;" +
     "TrustedFile='c:¥sybase¥ini¥trusted.txt';";
```

注意 SSL を使用するように、Adaptive Server を設定してください。SSL の詳細については、『Adaptive Server Enterprise システム管理ガイド』を参照してください。

高可用性システムでのフェールオーバの使用

高可用性クラスタには、2 つ以上のマシンが含まれます。これらのマシンは、1 つのマシン (またはアプリケーション) がダウンした場合にもう 1 つのマシンが両方のマシンの負荷を処理するように設定されています。このようなマシンのそれぞれを、高可用性クラスタのノードといいます。一般的に、高可用性クラスタはシステムが常に稼働していなければならないような環境で使用します。たとえば、クライアントが 1 年 365 日絶えず接続する銀行のシステムなどです。

フェールオーバによって、Adaptive Server をアクティブ／アクティブ設定またはアクティブ／パッシブ設定の高可用性クラスタで運用できます。

フェールオーバが発生すると、プライマリ・コンパニオンに接続していたクライアントは、フェールオーバ・プロパティを使用して、自動的にセカンダリ・コンパニオンへのネットワーク接続を再確立します。フェールオーバを有効にするには、接続プロパティ HASession を “1” (デフォルト値は “0”) に設定します。このプロパティを設定しないと、サーバでフェールオーバが設定されていても、セッションではフェールオーバが行われません。SecondaryServer プロパティと SecondaryPort プロパティも設定します。

使用するシステムの高可用性設定の詳細については、Adaptive Server のマニュアル『高可用性システムにおける Sybase フェールオーバの使用』を参照してください。

トランザクションでフェールオーバが発生した場合、フェールオーバ前にデータベースにコミットされた変更のみが保持されます。フェールオーバが発生すると、プロバイダは、セカンダリ・サーバへの再接続を試行します。セカンダリ・サーバへの接続が確立されると、ADO.NET Data Provider は、フェールオーバの発生を示すメッセージとともに AseFailoverException を返します。クライアントは、新しい接続を使用して、失敗したトランザクションを再適用しなければなりません。セカンダリ・サーバへの接続が確立できない場合は、接続が失われたことを示すメッセージとともに、ADO.NET Data Provider で通常の AseException が発生します。次に例を示します。

```
AseConnection.ConnectionString =
   ”Data Source='tpsun1';" +
   "Port = 5000;" +
   "Database=pubs2;" +
   "User ID=sa;" +
   "Password=sapass;" +
   "HASession=1;" +
   "Secondary Data Source='tpsun2';" +
   "Secondary Server Port=5000";
```

次のコードは、AseFailoverException の検出方法を示しています。

```
....
Open connection
...more code

try
{
   using (AseDataReader rdr =
   selectCmd.ExecuteReader())
   {
   ....
   }
}
catch (AseFailoverException)
{
   //Make sure that you catch AseFailoverException
   //before AseException as AseFailoverException is
   //derived from AseException

   //HA has occured.The application has successfully
   //connected to the secondary server.All uncommitted
   //transactions have been rolled back.

   //You could retry your transactions or prompt user
   //for a retry operation
}
catch (AseException)
{
//Either some other problem or the Failover did not
```

```
//successfully connect to the secondary server.Apps.
//should react accordingly
}
```

Kerberos 認証の使用

Kerberos は、簡単なログイン認証と相互のログイン認証を提供する業界標準のネットワーク認証システムです。Kerberos を使用することで、さまざまなアプリケーションにわたるシングル・サインオンをきわめて安全な環境内で行えます。ネットワークでパスワードを渡す代わりに、Kerberos サーバがユーザのパスワードと使用可能なサービスのパスワードの暗号化されたバージョンを保持します。

さらに Kerberos では、機密性とデータの整合性を維持するために暗号化を使用します。

Adaptive Server と Adaptive Server ADO.NET Data Provider は、Kerberos 接続をサポートします。Adaptive Server ADO.NET Data Provider は特に、MIT、CyberSafe、Active Directory の KDC (Key Distribution Center) をサポートします。

プロセスの概要

Kerberos 認証プロセスは次のように機能します。

1 クライアント・アプリケーションは、特定のサービスにアクセスするための「チケット」を Kerberos サーバに要求します。

2 Kerberos サーバは、2 つのパケットを含むチケットをクライアントに返します。第 1 のパケットはユーザ・パスワードにより暗号化されます。第 2 のパケットはサービス・パスワードにより暗号化されます。これらの各パケット内に「セッション・キー」が含まれます。

3 クライアントは、セッション・キーを取得するためにユーザ・パケットを復号化します。

4 クライアントは新しい認証パケットを作成し、それをセッション・キーにより暗号化します。

5 クライアントは、認証パケットとサービス・パケットをサービスに送信します。

6 サービスは、セッション・キーを取得するためにサービス・パケットを復号化し、ユーザ情報を取得するために認証パケットを復号化します。

7 サービスは、認証パケットからのユーザ情報と、サービス・パケットにも含まれているユーザ情報を比較します。両者が一致する場合、ユーザは認証済みです。

8　サービスは、認証パケットに含まれる検証データに加えてサービス固有の情報を含む確認パケットを作成します。

9　サービスは、このデータをセッション・キーとともに暗号化し、それをクライアントに返します。

10　クライアントは、パケットを復号化するために Kerberos から受信したユーザ・パケット内のセッション・キーを使用し、サービスがそれ自身の主張に一致しているかどうかを検証します。

こうした方法で、ユーザとサービスは相互に認証されます。以後、クライアントとサービス (この場合は Adaptive Server データベース・サーバ) の間の通信はすべて、セッション・キーにより暗号化されます。これにより、サービスとクライアント間で送信されるすべてのデータが望ましくない閲覧者から正しく保護されます。

稼働条件

認証システムとして Kerberos を使用するには、Kerberos に認証を委任するように Adaptive Server Enterprise を設定します。詳細については、Adaptive Server Enterprise の『システム管理ガイド』を参照してください。Windows では、クライアント・ライブラリとともに Kerberos クライアント・ライブラリがインストールされます。

Adaptive Server ADO.NET Data Provider で Kerberos を使用するには、Kerberos に対して MIT/CyberSafe Client ライブラリを設定し、Adaptive Server を有効にします。

Kerberos 認証の有効化

Adaptive Server ADO.NET Data Provider に対して Kerberos を有効にするには、プログラムに次の行を追加します。

```
AuthenticationClient=<one of 'mitkerberos' or
'cybersafekerberos' or 'activedirectory' and
ServerPrincipal=<ASE server name>
```

ここで *<ASE server name>* は、KDC (Key Distribution Center) 内で設定された論理名またはプリンシパルです。

Adaptive Server ADO.NET Data Provider はこの情報を使用して、設定された KDC および Adaptive Server サーバと Kerberos 認証をネゴシエートします。Windows では、activedirectory を選択することで追加設定を不要にできます。

Kerberos クライアント・ライブラリは、さまざまな KDC 間で互換性を持ちます。たとえば、KDC が Microsoft Active Directory の場合でも、mitkerberos と同じ AuthenticationClient を設定できます。

Kerberos クライアントで別のキャッシュ内の TGT を検索する場合は、userprincipal 接続プロパティを指定できます。

SQL_DRIVER_NOPROMPT とともに SQLDriverConnect を使用する場合、ConnectString は次のようになります。

```
"Driver=Adaptive Server Enterprise;UID=sa;
PWD='';Server=sampleserver;
Port=4100;Database=pubs2;
AuthenticationClient=mitkerberos;
ServerPrincipal=MANGO;”
```

Windows での Kerberos の有効化

AseConnection.ConnectionString: に次のプロパティを追加します。

```
AuthenticationClient=<one of activedirectory or mitkerberos or
cybersafekerberos>
ServerPrincipal=<MANGO>
```

<*Mango*> は、サインオンの認証のために使用されるプリンシパル・サーバの名前です。

Key Distribution Center からの初期チケットの取得

Kerberos 認証を使用するには、Key Distribution Center から TGT (Ticket Granted Ticket) と呼ばれる初期チケットを生成します。このチケットを取得する手順は、使用する Kerberos ライブラリに応じて異なります。詳細については、ベンダのマニュアルを参照してください。

❖ MIT Kerberos クライアント・ライブラリ用の TGT の生成

1 コマンド・ラインに次のように入力して kinit ユーティリティを開始します。

```
% kinit
```

2 your_name@YOUR.REALM などの kinit ユーザ名を入力します。

3 my_password など、your_name@YOUR.REALM のパスワードを入力します。パスワードを入力すると、kinit ユーティリティにより TGT (Ticket Granting Ticket) に対する要求が認証サーバに送信されます。

このパスワードは、キーの計算のために使用されます。そのキーは、応答の一部を復号化するために使用されます。この応答には、セッション・キーに加えて要求の確認が含まれます。パスワードを正しく入力していれば、この段階で TGT が取得されています。

4　コマンド・ラインに次のように入力して、TGT が取得されていることを確認します。

```
% klist
```

klist コマンドの結果は次のようになるはずです。

```
Ticket cache：/var/tmp/krb5cc_1234
Default principal：your_name@YOUR.REALM
Valid starting      Expires            Service principal
24-Jul-95 12:58:02  24-Jul-95 20:58:15  krbtgt/YOUR.REALM@YOUR.REALM
```

結果の説明

Ticket cache：チケット・キャッシュ・フィールドにより、どのファイルにクレデンシャル・キャッシュが含まれているかがわかります。

Default principal：デフォルトのプリンシパルは、TGT を所有するユーザ (この場合はユーザ自身) のログインです。

Valid starting/Expires/Service principal：以降の出力は、既存のチケットのリストです。これは要求した最初のチケットであるため、1 つのチケットのみがリストに含まれています。サービス・プリンシパル (krbtgt/YOUR.REALM@YOUR.REALM) は、このチケットが TGT であることを示しています。
このチケットは、約 8 時間有効であることに注意してください。

第 5 章

Adaptive Server ADO.NET Data Provider API リファレンス

この章では、Adaptive Server ADO.NET Data Provider の API について説明します。最新の API のマニュアルと、Microsoft ADO.NET インタフェースの実装に該当するプロパティとメソッドの詳細については、Adaptive Server ADO.NET のオンライン・ヘルプを参照してください。オンライン・ヘルプにアクセスするには、Microsoft Document Explorer を開いて、[Sybase ASE ADO.NET Data Provider] を選択します。詳細とその使用例については、Microsoft .NET Framework のドキュメントも参照してください。

AseClientPermission クラス

説明 Adaptive Server ADO.NET Data Provider で、ユーザが Adaptive Server Enterprise データ・ソースに適切なセキュリティ・レベルでアクセスできるようにします。

基本クラス DBDataPermission

AseClientPermission コンストラクタ

説明 AseClientPermission クラスの新しいインスタンスを初期化します。

構文 1 void **AseClientPermission( )**

構文 2 void **AseClientPermission(** PermissionState *state* **)**

構文 3 void **AseClientPermission(** PermissionState *state*,
bool *allowBlankPassword* **)**

パラメータ **state**PermissionState 値の 1 つ。

allowBlankPasswordブランクのパスワードが許可されるかどうかを示す。

AseClientPermissionAttribute クラス

説明 カスタム・セキュリティ属性にセキュリティの操作を関連付けます。

基本クラス DBDataPermissionAttribute

AseClientPermissionAttribute コンストラクタ

説明 AseClientPermissionAttribute クラスの新しいインスタンスを初期化します。

構文 void **AseClientPermissionAttribute(** SecurityAction *action* **)**

パラメータ **action の値**宣言されているセキュリティを使用して実行可能な操作を表す SecurityAction 値の 1 つ。

戻り値 AsePermissionAttribute オブジェクト。

CreatePermission メソッド

説明 属性のプロパティに応じて設定される AsePermission オブジェクトを返します。

構文 IPermission **CreatePermission( )**

AseCommand クラス

説明

Adaptive Server に対して実行するコマンドを表し、動的 SQL 文またはストアド・プロシージャをカプセル化します。Adaptive Server データベースでコマンドを実行する次のメソッドがあります。

- ExecuteNonQuery － 結果セットを返さない Execute コマンド。
- ExecuteScalar － 結果セットを返さない Execute コマンド。
- ExecuteReader － 単一の値を返す Execute コマンド。
- ExecuteXmlReader － XML を返す Execute コマンド。

基本クラス

Component

実装

IDbCommand、IDisposable

参照

「AseCommand を使用したデータの取得と操作」(37 ページ)、「データに対するアクセスと操作」(36 ページ)

注意 パラメータを取るストアド・プロシージャを呼び出す場合は、ストアド・プロシージャに対してパラメータを指定してください。詳細については、「ストアド・プロシージャの使用」(72 ページ) と「AseParameter クラス」(153 ページ) を参照してください。

AseCommand コンストラクタ

説明

AseCommand オブジェクトを初期化します。

構文 1

void **AseCommand( )**

構文 2

void **AseCommand(** string *cmdText* **)**

構文 3

void **AseCommand(** string *cmdText*, AseConnection *connection* **)**

構文 4

void **AseCommand(** string *cmdText*, AseConnection *connection*, AseTransaction *transaction* **)**

パラメータ

cmdText：SQL 文またはストアド・プロシージャのテキスト。

connection：現在の接続。

transaction：AseConnection を実行する AseTransaction。

Cancel メソッド

説明 AseCommand オブジェクトの実行をキャンセルします。

構文 void **Cancel( )**

使用法 キャンセルする対象がない場合は、何も実行されません。処理中のコマンドがあり、キャンセル操作に失敗した場合、例外は生成されません。

実装 IDbCommand.Cancel

CommandText プロパティ

説明 SQL 文またはストアド・プロシージャのテキストを表します。

構文 string **CommandText**

アクセス 読み込みと書き込み

プロパティ値 実行する SQL 文またはストアド・プロシージャの名前。デフォルトは空の文字列です。

実装 IDbCommand.CommandText

参照 「AseCommand コンストラクタ」(101 ページ)

CommandTimeout プロパティ

説明 コマンド実行の試行を終了してエラーを生成するまでの待機時間を秒単位で表します。

構文 int **CommandTimeout**

アクセス 読み込みと書き込み

実装 IDbCommand.CommandTimeout

デフォルト 30 秒

使用法 0 を指定すると、待機時間は制限されません。コマンド実行の試行が永久的に待機されるため、0 は指定しないでください。

CommandType プロパティ

説明 AseCommand によって示されるコマンドの種類を表します。

構文 CommandType **CommandType**

アクセス 読み込みと書き込み

実装 IDbCommand.CommandType

使用法

サポートされるコマンドの種類は次のとおりです。

- **CommandType.StoredProcedure**：この CommandType を指定したときは、コマンド・テキストをストアド・プロシージャの名前にして、必要な引数を AseParameter オブジェクトとして指定します。
- **CommandType.Text**：これはデフォルトの値です。

CommandType プロパティを StoredProcedure に設定したときは、CommandText プロパティにストアド・プロシージャの名前を設定してください。Execute メソッドの 1 つを呼び出すと、このストアド・プロシージャが実行されます。

Connection プロパティ

説明

AseCommand オブジェクトで使用する接続オブジェクトを表します。

構文

AseConnection **Connection**

アクセス

読み込みと書き込み

デフォルト

デフォルト値は null 参照です。Visual Basic の場合は “Nothing” です。

CreateParameter メソッド

説明

AseCommand オブジェクトにパラメータを渡す AseParameter オブジェクトを作成します。

構文

AseParameter **CreateParameter( )**

戻り値

AseParameter オブジェクトとしての新しいパラメータ。

使用法

- ストアド・プロシージャや一部の SQL 文では、*@name* パラメータによって文のテキストに示されたパラメータを取ることができます。
- CreateParameter メソッドは AseParameter オブジェクトを作成します。AseParameter のプロパティを設定することで、パラメータの値やデータ型などを指定できます。

参照

「AseParameter クラス」(153 ページ)

ExecuteNonQuery メソッド

説明

Insert、Update、Delete、データ定義文など、結果セットを返さない文を実行します。

構文

int **ExecuteNonQuery( )**

戻り値

影響を受けたローの数。

実装

IDbCommand.ExecuteNonQuery

使用法

- ExecuteNonQuery では、DataSet を使わずにデータベース内のデータを変更できます。データを変更するには、Update、Insert、または Delete 文を実行します。
- ExecuteNonQuery はローを返しませんが、出力パラメータ、またはパラメータにマッピングされた戻り値にはデータが格納されます。
- Update、Insert、Delete 文では、戻り値はコマンドによって影響を受けたローの数です。他の種類の文では、戻り値は -1 になります。

ExecuteReader メソッド

説明 結果セットを返す SQL 文を実行します。

構文 1 AseDataReader **ExecuteReader( )**

構文 2 AseDataReader **ExecuteReader(CommandBehavior** *behavior* **)**

パラメータ **behavior**：CloseConnection、Default、KeyInfo、SchemaOnly、SequentialAccess、SingleResult、または SingleRow のいずれか。

デフォルト値は CommandBehavior.Default です。このパラメータの詳細については、.NET Framework のドキュメントで CommandBehavior Enumeration (CommandBehavior 列挙体) に関する説明を参照してください。

戻り値 AseDataReader オブジェクトとしての結果セット。

使用法 必要に応じて CommandText や Parameters を設定した、現在の AseCommand オブジェクトを実行します。AseDataReader オブジェクトは、結果セットの前方向への読み込みのみ可能です。変更可能な結果セットが必要な場合は、AseDataAdapter クラスを使用してください。

参照 「AseDataAdapter クラス」(124 ページ)、「AseDataReader クラス」(131 ページ)

ExecuteScalar メソッド

説明 単一の値を返す文を実行します。複数のローとカラムを返すクエリに対してこのメソッドを呼び出すと、最初のローの最初のカラムだけが返されます。

構文 object **ExecuteScalar( )**

戻り値 結果セットの最初のローの最初のカラム。結果セットが空の場合は、null 参照が返されます。

実装 IDbCommand.ExecuteScalar

ExecuteXmlReader メソッド

説明　有効な FOR XML 句を持つ SQL 文を実行し、結果セットを返します。XML が格納された 1 つのテキスト・カラムを返す文を使用して、ExecuteXmlReader を呼び出すこともできます。

構文 1

XmlReader **ExecuteXmlReader( )**

パラメータ　なし

戻り値　XmlReader オブジェクトとしての結果セット。

使用法　必要に応じて CommandText や Parameters を設定した、現在の AseCommand オブジェクトを実行します。

参照　Microsoft .NET ドキュメントの XmlReader。

NamedParameters

説明　このプロパティは、この接続に関連付けられている AseCommand オブジェクトのデフォルトの動作を決定します。

構文

bool **NamedParameters**

プロパティ値　デフォルト値は、関連付けられている AseConnection オブジェクトで設定されている値と同じ。このプロパティが “true” (AseConnection のデフォルト) の場合、Provider は、パラメータ・マーカ ('?') ではなくパラメータ名が使われていると想定します。次に例を示します。

```
select total_sales from titles where title_id = @title_id
```

パラメータ・マーカを使用する場合は、このプロパティを false に設定します。これは、ODBC および JDBC と互換性があります。

次に例を示します。

```
select total_sales from titles where title_id = ?
```

アクセス　読み込みと書き込み

Parameters プロパティ

説明　現在の文のパラメータ・コレクションを表します。パラメータを記述するには、CommandText に *@name* パラメータまたは疑問符を使用します。

構文

AseParameterCollection **Parameters**

アクセス　読み込み専用

プロパティ値　SQL 文またはストアド・プロシージャのパラメータ。デフォルト値は空のコレクションです。

使用法

- CommandType が Text に設定されている場合は、*@name* パラメータを使用します。次に例を示します。

  ```
  SELECT * FROM Customers WHERE CustomerID = @name
  ```

  NamedParameters が “false” に設定されている場合は、? パラメータ・マーカを使用します。次に例を示します。

  ```
  SELECT * FROM Customers WHERE CustomerID = ?
  ```

- 疑問符のプレースホルダを使用する場合、AseParameter オブジェクトが AseParameterCollection に追加される順序と、コマンド・テキスト内のパラメータの疑問符プレースホルダの位置を完全に一致させてください。
- コレクション内のパラメータと実行されるクエリの要件が一致しない場合は、エラーが発生するか、例外が返されます。
- 出力パラメータには、AseDbType を指定してください (準備のありなしに関係なく)。

参照

「AseParameterCollection クラス」(158 ページ)

Prepare メソッド

説明

データ・ソースに対する AseCommand を準備またはコンパイルします。

構文

void **Prepare( )**

実装

IDbCommand.Prepare

使用法

- Prepare を呼び出す前に、準備対象となる文の各パラメータのデータ型を指定します。
- Prepare の呼び出し後に Execute メソッドを呼び出すと、Size プロパティに指定した値よりも大きいパラメータ値は、最初に指定したパラメータのサイズに自動的にトランケートされます。このとき、トランケート・エラーは返されません。

Transaction プロパティ

説明

現在のコマンドをトランザクションに関連付けます。

構文

AseTransaction **Transaction**

アクセス

読み込みと書き込み

使用法

デフォルト値は null 参照です。Visual Basic の場合は Nothing です。

Transaction プロパティに特定の値がすでに設定されていて、コマンドが実行中の場合、このプロパティは設定できません。AseCommand オブジェクトと同じ AseConnection に接続されていない AseTransaction オブジェクトをトランザクション・プロパティに設定すると、次回、文を実行する際に例外が返されます。

UpdatedRowSource プロパティ

説明

AseDataAdapter の Update メソッドで使用されるときに DataRow に適用されるコマンドの結果を表します。

構文

UpdateRowSource **UpdatedRowSource**

アクセス

読み込みと書き込み

実装

IDbCommand.UpdatedRowSource

プロパティ値

UpdatedRowSource 値の 1 つ。コマンドが自動的に生成される場合、デフォルトは None です。コマンドが自動的に生成されない場合、デフォルトは “Both” になります。

AseCommandBuilder クラス

説明

SQL SELECT 文に基づいてシングルテーブルの SQL INSERT、UPDATE、および DELETE 文を自動的に作成します。

基本クラス

Component

実装

IDisposable

参照

「AseDataAdapter クラス」(124 ページ)

AseCommandBuilder コンストラクタ

説明

AseCommandBuilder オブジェクトを初期化します。

構文 1

void **AseCommandBuilder( )**

構文 2

void **AseCommandBuilder(** AseDataAdapter *adapter* **)**

パラメータ

adapter調整のための文を生成する対象となる AseDataAdapter オブジェクトです。

DataAdapter プロパティ

説明 文を生成する対象となる AseDataAdapter です。

構文 AseDataAdapter **DataAdapter**

アクセス 読み込みと書き込み

プロパティ値 AseDataAdapter オブジェクトです。

使用法 AseCommandBuilder の新しいインスタンスを作成すると、この AseDataAdapter に関連するすべての既存 AseCommandBuilder が解放されます。

DeleteCommand プロパティ

説明 Update() が呼び出されたとき、DataSet 内の削除されたローに対応するデータベース内のローを削除するためにデータベースに対して実行される AseCommand オブジェクトを表します。

構文 AseCommand **DeleteCommand**

アクセス 読み込みと書き込み

使用法 既存の AseCommand オブジェクトに DeleteCommand を割り当てた場合、AseCommand オブジェクトのクローンは作成されません。DeleteCommand は、既存の AseCommand に対する参照を維持します。

DeriveParameters メソッド

説明 指定した AseCommand オブジェクトの Parameters コレクションを作成します。これは、AseCommand で指定したストアド・プロシージャで使用されます。

構文 void **DeriveParameters(** AseCommand *command* **)**

パラメータ **コマンド**パラメータを抽出する対象となる AseCommand オブジェクトです。

使用法

- DeriveParameters は、AseCommand のすべての既存パラメータ情報を上書きします。
- DeriveParameters では、データベース・サーバへの追加呼び出しが必要になります。パラメータ情報が事前にわかっている場合は、情報を明示的に設定することによって Parameters コレクションを作成する方が効率的です。

Dispose メソッド

説明 オブジェクトに関連付けられたリソースを解放します。

構文 void **Dispose( )**

GetDeleteCommand メソッド

説明　生成された AseCommand オブジェクトは、AseDataAdapter.Update() が呼び出されたときに、データベースに対して Delete オペレーションを実行します。

構文

```
AseCommand GetDeleteCommand( )
```

戻り値　削除を実行するために必要な自動生成された AseCommand オブジェクト。

使用法

- GetDeleteCommand メソッドは、実行対象の AseCommand オブジェクトを返します。これは、情報入手またはトラブルシューティングの目的で役立つ場合があります。
- 修正するコマンドのベースとして GetDeleteCommand を使用することもできます。たとえば、GetDeleteCommand を呼び出し、CommandTimeout 値を変更したうえで、その値を明示的に AseDataAdapter に設定できます。
- アプリケーションが Update または GetDeleteCommand を呼び出すと、最初の SQL 文が生成されます。最初の SQL 文が生成された後は、アプリケーションは、何らかの方法で SQL 文を変更する場合に RefreshSchema を呼び出す必要があります。呼び出しを行わない場合、GetDeleteCommand は以前の文の情報を引き続き使用します。

GetInsertCommand メソッド

説明　生成された AseCommand オブジェクトは、AseDataAdapter.Update() が呼び出されたときに、データベースに対して Insert オペレーションを実行します。

構文

```
AseCommand GetInsertCommand( )
```

戻り値　挿入を実行するために必要な自動生成された AseCommand オブジェクト。

使用法

- GetInsertCommand メソッドは、実行対象の AseCommand オブジェクトを返します。これは、情報入手またはトラブルシューティングの目的で役立つ場合があります。
- 修正するコマンドのベースとして GetInsertCommand を使用することもできます。たとえば、GetInsertCommand を呼び出し、CommandTimeout 値を変更したうえで、その値を明示的に AseDataAdapter に設定できます。
- アプリケーションが Update または GetInsertCommand のいずれかを呼び出すと、SQL 文が生成されます。最初の SQL 文が生成された後は、アプリケーションは、何らかの方法で SQL 文を変更する場合に RefreshSchema を呼び出す必要があります。呼び出しを行わない場合、GetInsertCommand は引き続き、以前の文の情報 (正しくないことがあります) を使用します。

GetUpdateCommand メソッド

説明

生成された AseCommand オブジェクトは、AseDataAdapter.Update() が呼び出されたときに、データベースに対して Update 処理を実行します。

構文

AseCommand **GetUpdateCommand( )**

戻り値

更新を実行するために必要な自動生成された AseCommand オブジェクト。

使用法

- GetUpdateCommand メソッドは、実行対象の AseCommand オブジェクトを返します。これは、情報入手またはトラブルシューティングの目的で役立つ場合があります。
- 修正するコマンドのベースとして GetUpdateCommand を使用することもできます。たとえば、GetUpdateCommand を呼び出し、CommandTimeout 値を変更したうえで、その値を明示的に AseDataAdapter に設定できます。
- アプリケーションが Update または GetUpdateCommand を呼び出すと、最初の SQL 文が生成されます。最初の SQL 文が生成された後は、アプリケーションは、何らかの方法で SQL 文を変更する場合に RefreshSchema を呼び出す必要があります。呼び出しを行わない場合、GetUpdateCommand は、以前の文の情報 (正しくないことがあります) を引き続き使用します。

InsertCommand プロパティ

説明

Update() が呼び出されたとき、DataSet 内の挿入されたローに対応するデータベース内のローを追加するためにデータベースに対して実行される AseCommand を表します。

構文

AseCommand **InsertCommand**

アクセス

読み込みと書き込み

使用法

既存の AseCommand オブジェクトに InsertCommand を割り当てた場合、AseCommand のクローンは作成されません。InsertCommand は、既存の AseCommand に対する参照を維持します。

このコマンドによって複数のローが返された場合、AseCommand オブジェクトの UpdatedRowSource プロパティの設定に応じて、それらのローが DataSet に追加されます。

参照

「Update メソッド」(130 ページ)

PessimisticUpdate プロパティ

説明　ペシミスティック更新またはオプティミスティック更新のどちらを実装するかを示します。

構文

```
public bool PessimisticUpdate
```

アクセス　読み込みと書き込み

プロパティ値　ペシミスティック更新の場合は true。オプティミスティック更新の場合は false。

使用法

- ペシミスティック更新では、レコードがロックされます。ロックされたレコードは、すべてのユーザが表示できますが、編集できるユーザは 1 人だけに制限されます。
- オプティミスティック更新では、複数のユーザが同じレコードを編集できます。

QuotePrefix プロパティ

説明　スペースなどの文字を含むデータベース・オブジェクト名を指定する場合に使用する先頭文字 (複数可) です。

構文

```
string QuotePrefix
```

アクセス　読み込みと書き込み

プロパティ値　使用する先頭文字 (複数可)。角かっこを指定できます。また、ASE QUOTED_IDENTIFIER オプションが “off” に設定されている場合は、二重引用符も使用できます。デフォルトは空の文字列です。

使用法

- ASE オブジェクトには、スペース、カンマ、セミコロンなどの文字を含めることができます。QuotePrefix プロパティと QuoteSuffix プロパティには、オブジェクト名をカプセル化するためのデリミタを指定します。
- Insert、Update、または Delete オペレーションの後に QuotePrefix プロパティまたは QuoteSuffix プロパティを変更することはできませんが、DataAdapter の Update メソッドを呼び出した後でその設定を変更できます。

参照

- 識別子の詳細については、『ASE リファレンス・マニュアル』を参照してください。
- QUOTED IDENTIFIER オプションの詳細については、『ASE リファレンス・マニュアル』を参照してください。

QuoteSuffix プロパティ

説明 スペースなどの文字を含むデータベース・オブジェクト名を指定する場合に使用する末尾文字 (複数可) です。

構文 string **QuoteSuffix**

アクセス 読み込みと書き込み

プロパティ値 使用する末尾文字 (複数可)。角かっこを指定できます。また、ASE QUOTED_IDENTIFIER オプションが off に設定されている場合は、二重引用符も使用できます。デフォルトは空の文字列です。

使用法

- ASE オブジェクトには、スペース、カンマ、セミコロンなどの文字を含めることができます。QuotePrefix プロパティと QuoteSuffix プロパティには、オブジェクト名をカプセル化するためのデリミタを指定します。
- Insert、Update、または Delete オペレーションの後に QuotePrefix プロパティまたは QuoteSuffix プロパティを変更することはできませんが、DataAdapter の Update メソッドを呼び出した後でその設定を変更できます。

参照

- 識別子の詳細については、『ASE リファレンス・マニュアル』を参照してください。
- QUOTED IDENTIFIER オプションの詳細については、『ASE リファレンス・マニュアル』を参照してください。

RefreshSchema メソッド

説明 Insert 文、Update 文、または Delete 文の生成で使用されるデータベース・スキーマ情報を更新します。

構文 void **RefreshSchema( )**

使用法 関連する AseDataAdapter の SelectCommand の値が変更された場合は、必ず RefreshSchema を呼び出してください。

SelectCommand プロパティ

説明 Fill または FillSchema で、DataSet にコピーする結果セットをデータベースから取得するために使用される AseCommand を表します。

構文 AseCommand **SelectCommand**

アクセス 読み込みと書き込み

使用法

- 事前に作成された AseCommand に SelectCommand を割り当てた場合、AseCommand のクローンは作成されません。SelectCommand は、その AseCommand オブジェクトに対する参照を維持します。
- SelectCommand がローを返さない場合は、DataSet にテーブルが追加されず、例外も発生しません。
- Select 文は、AseDataAdapter コンストラクタ内でも指定できます。

UpdateCommand プロパティ

説明

Update() が呼び出されたとき、DataSet 内の更新されたローに対応するデータベース内のローを更新するためにデータベースに対して実行される AseCommand を表します。

構文

AseCommand **UpdateCommand**

アクセス

読み込みと書き込み

使用法

- 事前に作成された AseCommand に UpdateCommand を割り当てた場合、AseCommand のクローンは作成されません。UpdateCommand は、その AseCommand オブジェクトに対する参照を維持します。
- このコマンドによって複数のローが返された場合、AseCommand オブジェクトの UpdatedRowSource プロパティの設定に応じて、それらのローが DataSet にマージされます。

参照

「Update メソッド」(130 ページ)

AseConnection クラス

説明

Adaptive Server データベースへの接続を表します。

基本クラス

Component

実装

IDbConnection、IDisposable

参照

「データベースへの接続」(31 ページ)

AseConnection コンストラクタ

説明 AseConnection オブジェクトを初期化します。データベースに対して操作を実行するには、最初に接続をオープンします。

構文 1 **AseConnection( )**

構文 2 **AseConnection(** string *connectionString* **)**

パラメータ **connectionString**：Adaptive Server 接続文字列。接続文字列は、一連のキーワードと値のペアをセミコロンで区切ったものです。

表 *5-1:* 接続文字列パラメータ

プロパティ名	説明	必須	デフォルト値
UID、UserID、User ID、User	Adaptive Server サーバへの接続に必要なユーザ ID。大文字と小文字を区別する。	はい	空
PWD、Password	Adaptive Server サーバへの接続に使用するパスワード。大文字と小文字を区別する。	いいえ (ユーザ名にパスワードが不要な場合)	空
Server、Data Source、DataSource、Address、Addr、Network Address、Server Name	Adaptive Server サーバの名前または IP アドレス。	はい	空
Port、Server Port	Adaptive Server サーバのポート番号。	はい (ポート番号が Datasource に指定されていない場合)	空
AnsiNull	ODBC に厳格に準拠し、"= NULL" を使用不可とする。代わりに "IsNull" を使用する。デフォルトの動作を変更する場合は 1 に設定する。	いいえ	0
ApplicationName、Application Name	クライアント・アプリケーションを識別するために Adaptive Server が使用する名前。	いいえ	空
BufferCacheSize	入力／出力バッファをプール内に保持する。結果セットが非常に大きい場合、この値を大きくするとパフォーマンスが向上する。	いいえ	20
CharSet	指定されている文字セット。Adaptive Server ADO.NET Data Provider はデフォルトで Adaptive Server と同じデフォルトの文字セットをネゴシエートする。デフォルトの文字セットは ServerDefault。	いいえ	ServerDefault の値
ClientHostName	サーバへのログイン・レコードで渡されるクライアント・ホストの名前。たとえば、`ClientHostName='MANGO'`。	いいえ	空
ClientHostProc	サーバへのログイン・レコードで渡される、このホスト・マシン上のクライアント・プロセスを識別する語。たとえば、`ClientHostProc='MANGO-PROC'`。	いいえ	空

プロパティ名	説明	必須	デフォルト値
CodePageType	使用する文字コードの種類を指定する。有効な値は、ANSI と OEM。	いいえ	ANSI
ConnectionIdleTimeout、Connection IdleTimeout、Connection Idle Timeout	ドライバが接続をクローズする前に、接続が接続プール内でアイドル状態を維持できる時間 (秒数)。ゼロを指定すると、接続は無期限にアイドル状態を維持できる。	いいえ	0
Connection Lifetime	接続がオープン状態を維持できる時間 (秒数)。デフォルトの Connection Lifetime に到達 (超過) した接続をクライアントがクローズしたときに、ドライバは接続を接続プールに返すのではなくクローズする。アイドル状態の接続は、デフォルトの Connection Lifetime に到達した時点でクローズされ接続プールから削除される。 Connection Lifetime のデフォルト値は 0 であり、接続は無期限にオープン状態のままであることを示す。	いいえ	0
CumulativeRecordCount、Cumulative Record Count、 CRC	ストアド・プロシージャで複数の update 文が実行されたとき、ドライバ (ADO.NET のプロバイダを使用する場合) はデフォルトで、更新されたレコード総数を返す。このカウントには、update や insert に設定されたトリガで実行されるすべての更新操作が含まれる。ドライバが最後の更新カウントだけを返すようにする場合は、このプロパティを 0 に設定する。	いいえ	1
Database、 Initial Catalog	接続するデータベース。	いいえ	空
DSPassword、 Directory Service Password	LDAP サーバで匿名アクセスが許可されない場合に、LDAP サーバでの認証に使用するパスワード。パスワードは DSURL でも指定可能。	いいえ	空
DSPrincipal、 Directory Service Principal	LDAP サーバで匿名アクセスが許可されない場合に、LDAP サーバでの認証に使用するユーザ名。プリンシパルは DSURL でも指定可能。	いいえ	空
DSURL、 Directory Service URL	LDAP サーバへの URL。	いいえ	空
DTCProtocol (Windows のみ)	分散トランザクションを使用する場合に、ドライバで XA プロトコルまたは OleNative プロトコルのどちらかを使用することを許可する。詳細については、「第 4 章 Adaptive Server の高度な機能」の「分散トランザクションの使用」(83 ページ) を参照。	いいえ	XA

プロパティ名	説明	必須	デフォルト値
EnableServerPacketSize	最適なパケット・サイズを選択するために、Adaptive Server サーバ・バージョン 15.0 以降を許可する。	いいえ	1
EncryptPassword、EncryptPwd、Encrypt Password	パスワードの暗号化を有効にするかどうかを指定する。0 はパスワードの暗号化を無効にし、1 はパスワードの暗号化を有効にする。	いいえ	0
Encryption	指定されている暗号化方式。指定できる値：ssl、none。	いいえ	空
FetchArraySize	サーバから結果をフェッチする場合にドライバが取得するロー数を指定する。	いいえ	25
HASession	高可用性を有効にするかどうかを指定する。0 は高可用性を無効にし、1 は高可用性を有効にする。	いいえ	0
IgnoreErrorsIfRS Pending	エラー・メッセージが表示された場合に、ドライバの処理を続行するかどうかを指定する。1 に設定した場合、ドライバでエラーが無視され、サーバからさらに結果が取得可能な場合は結果の処理が続行される。0 に設定した場合、ドライバではエラーが発生した場合に保留中の結果があっても結果の処理を停止する。	いいえ	0
Language	Adaptive Server が返すエラー・メッセージの言語。	いいえ	空 ― デフォルトでは英語を使用。
LoginTimeOut、Connect Timeout、Connection Timeout	アプリケーションへ戻る前に、ログインの試行を待機する秒数。0 に設定するとタイムアウトが無効になり、接続の試行を永久的に待機する。	いいえ	15
min pool size	Adaptive Server への接続を強制的にクローズし、オープンしている接続総数が最小プール・サイズに達しないようにできる。プール内の接続数が最小プール・サイズと等しくなると、AseConnection.Close() で接続がクローズされる。	いいえ	20
max pool size	指定した最大プール・サイズよりも接続プールが大きくならないように制限できる。この制限に達すると、AseConnection.Open() で AseException が返される。	いいえ	100

プロパティ名	説明	必須	デフォルト値
NamedParameters	名前付きパラメータではなくパラメータ・マーカを使用する場合は、false に設定する。名前付きパラメータを使用する SQL は、`SELECT * from titles where title_id = @title_id` のようになる。 同じ SQL にパラメータ・マーカを使用すると、`SELECT * from titles where title_id = ?` のようになる。	いいえ	true
PacketSize、 Packet Size	Adaptive Server とクライアント間で送受信されるネットワーク・パケット 1 つあたりのバイト数。	いいえ	512
Ping Server	接続プールにある接続を使用する前に、その接続が有効であることを確認する必要がない場合は、false に設定する。	いいえ	true
pooling	接続プールを無効にする場合は、false に設定する。	いいえ	true
QuotedIdentifier	Adaptive Server で二重引用符で囲まれた文字列を識別子として処理するかどうかを指定する。0 は引用符付き識別子を無効にし、1 は引用符付き識別子を有効にする。	いいえ	0
RestrictMaximum PacketSize	EnableServerPacketSize に 1 を設定した場合にメモリに制約があるときは、このプロパティに 512 の倍数 (最大 65536) の int 値を設定する。	いいえ	0
SecondaryPort、 Secondary Port、 Secondary Server Port	アクティブ／アクティブ設定またはアクティブ／パッシブ設定でフェールオーバ・サーバとして機能する Adaptive Server サーバのポート番号。	はい (HASession が 1 に設定されていて、ポート番号が Secondary DataSource に指定されていない場合)	空
SecondaryServer、 Secondary Data Source、 Secondary DataSource、 Secondary Server、 Secondary Address、 Secondary Addr、 Secondary Network Address	アクティブ／アクティブ設定またはアクティブ／パッシブ設定でフェールオーバ・サーバとして機能する Adaptive Server サーバの名前または IP アドレス。 「セカンダリ・データ・ソース」は、SecondaryServer:SecondaryPort または SecondaryServer, SecondaryPort で指定可能。	はい (HASession が 1 に設定されている場合)	空
SQL Initialization String、 SQLInitializationString、 SQL Init String、 SQLInitString、 Initialization String、 InitializationString	データベース・サーバに渡されて、接続の直後に実行されるコマンドをスペースで区切って定義する。指定するコマンドは、結果の問い合わせには使用されない SQL コマンドであること。	いいえ	空

プロパティ名	説明	必須	デフォルト値
TextSize	Adaptive Server との間で送受信されるバイナリまたはテキスト・データのバイト単位での最大サイズ。たとえば、`TextSize=64000` では、最大サイズが 64 KB に制限される。	いいえ	空。Adaptive Server のデフォルトは 32 KB。
TightlyCoupled Transaction (Windows のみ)	分散トランザクションを使用するときに、同一の Adaptive Server サーバに接続している 2 つの DSN を使用している場合は、このプロパティを 1 に設定する。「第 4 章 Adaptive Server の高度な機能」の「分散トランザクションの使用」(83 ページ) を参照。	いいえ	0
TrustedFile	Encryption を ssl に設定した場合は、このプロパティに信頼されたファイルへのパスを設定する。	いいえ	空
UseCursor、Use Cursor	ドライバでカーソルを使用するかどうかを指定する。0 はカーソルを使用しない。1 はカーソルを使用する。	いいえ	0

注意 Data Source、DataSource、Secondary Data Source、Secondary DataSource は特殊なキーワードです。これらは、サーバ名を指定するだけでなく、次の形式でも使用できます。

```
DataSource=servername,port
```

または

```
DataSource=servername:port
```

たとえば、`DataSource=gamg:4100` では、サーバ名が “gamg” に、ポートが “4100” に設定されます。この場合、Port キーワードは接続文字列で不要になります。

例

次の文は、“HR-001” という名前の Adaptive Server データベース・サーバで実行されている “policies” という名前のデータベースへの接続に使用する AseConnection オブジェクトを初期化します。この接続では、ユーザ ID に “admin”、パスワードに “money” を使用します。

```
"Data Source='HR-001';
Port=5000; UID='admin';
PWD='money';
Database='policies';"
```

BeginTransaction メソッド

説明　トランザクション・オブジェクトを返します。1 つのトランザクション・オブジェクトに関連付けられているすべてのコマンドは、単一のトランザクションとして実行されます。トランザクションは、Commit() または Rollback() によって終了します。

構文 1　AseTransaction **BeginTransaction( )**

構文 2　AseTransaction **BeginTransaction(** IsolationLevel *isolationLevel* **)**

パラメータ　**isolationLevel：**IsolationLevel 列挙体のメンバ。デフォルト値は ReadCommitted です。

戻り値　新しいトランザクションを表すオブジェクト。

使用法　トランザクション・オブジェクトにコマンドを関連付けるには、AseCommand.Transaction プロパティを使用します。

例

```
AseTransaction tx = conn.BeginTransaction(IsolationLevel.ReadUncommitted );
```

参照　「Commit メソッド」(166 ページ)、「Rollback メソッド」(166 ページ)、「トランザクション処理」(75 ページ)。一貫性の詳細については、Adaptive Server Enterprise の『システム管理ガイド』を参照してください。

ChangeDatabase メソッド

説明　オープンしている AseConnection に対する現在のデータベースを変更します。

構文　void **ChangeDatabase(** string *database* **)**

パラメータ　**database：**現在のデータベースの代わりに使用するデータベースの名前。

実装　IDbConnection.ChangeDatabase

Close メソッド

説明　データベース接続をクローズします。

構文　void **Close( )**

実装　IDbConnection.Close

使用法　Close メソッドは、保留中のすべてのトランザクションをロールバックします。次に、接続を接続プールに解放します。接続プールが無効の場合は、接続をクローズします。StateChange イベントの処理中に Close を呼び出すと、以降の StateChange イベントは起動されません。1 つのアプリケーションで Close を複数回呼び出すことができます。

ConnectionString プロパティ

説明

データベースへの接続文字列を表します。

構文

string **ConnectionString**

アクセス

読み込みと書き込み

実装

IDbConnection.ConnectionString

使用法

- ConnectionString は ODBC 接続文字列のフォーマットとできるかぎり同じになるように設計されています。
- ConnectionString プロパティを設定できるのは、接続がクローズされているときだけです。接続文字列値の多くには、対応する読み込み専用プロパティがあります。接続文字列を設定すると、これらのプロパティのすべてが更新されます。ただし、エラーが検出された場合は、どのプロパティも更新されません。AseConnection プロパティは、ConnectionString に含まれている設定だけを返します。
- クローズされている接続に対して ConnectionString をリセットすると、パスワードを含むすべての接続文字列値と関連するプロパティがリセットされます。
- プロパティを設定すると、接続文字列の予備検証が実行されます。アプリケーションが Open メソッドを呼び出すと、接続文字列は完全に検証されます。接続文字列に無効またはサポートされないプロパティが検出された場合は、ランタイム例外が生成されます。
- 値は一重引用符または二重引用符で囲みます。一重引用符または二重引用符を接続文字列の値の一部として使用する場合は、もう一方の引用符で値を囲みます。たとえば、`name="value's"` または `name= 'value"s'` は有効ですが、`name='value's'` または `name= ""value""` は無効です。

  ブランク文字は、値または引用符の中で使用されていないかぎり無視されます。

  キーワードと値の各ペアはセミコロンで区切ります。セミコロンが値の一部の場合は、引用符も使用して区切ってください。

  エスケープ・シーケンスはサポートされず、値の型は無関係です。

  名前では、大文字と小文字を区別しません。接続文字列内に同じプロパティ名が複数回、記述されている場合は、最後の記述に指定されている値が使用されます。
- ユーザ ID やパスワードをダイアログ・ボックスから取得して接続文字列に挿入する場合など、ユーザ入力に基づいて接続文字列を構築するときは注意が必要です。ユーザが規定外の接続文字列をこれらの値に組み込むことを、アプリケーションは禁止する必要があります。

例　次の文は、サーバ mango で実行されている pubs2 という名前の Adaptive Server データベースに接続する接続文字列を設定し、その接続をオープンします。

```
AseConnection conn = new AseConnection( "Data Source=mango;
Port=5000;
UID=sa;
PWD='';
Database='pubs2';
" ):
conn.Open();
```

ConnectionTimeout プロパティ

説明　接続要求がタイムアウトし、エラーが発生するまでの秒数を表します。

構文　int **ConnectionTimeout**

アクセス　読み込み専用

デフォルト　15 秒

実装　IDbConnection.ConnectionTimeout

例　次の文は、ConnectionTimeout を 30 秒に変更します。

```
conn.ConnectionTimeout = 30;
```

CreateCommand メソッド

説明　AseCommand オブジェクトを初期化します。AseCommand のプロパティを使用して、その動作を制御できます。

構文　AseCommand **CreateCommand( )**

戻り値　AseCommand オブジェクト。

使用法　Command オブジェクトは AseConnection に関連付けられます。

Database プロパティ

説明　接続がオープンされた後に使用される現在のデータベースの名前を表します。

構文　string **Database**

アクセス　読み込み専用

実装　IDbConnection.Database

使用法

```
if (conn.Database != "pubs2") conn.ChangeDatabase("pubs2");
```

InfoMessage イベント

説明

Adaptive Server ADO.NET Data Provider が警告メッセージまたは情報メッセージを送信したときに発生します。

構文

event AseInfoMessageEventHandler **InfoMessage**

使用法

イベント・ハンドラが、このイベントに関連するデータを含む AseInfoMessageEventArgs 型の引数を受け取ります。Errors および Message プロパティに、このイベントに固有の情報が記載されます。

NamedParameters

説明

この接続に関連付けられている AseCommand オブジェクトのデフォルトの動作を決定します。詳細については、AseCommand クラス (NamedParameter プロパティ) を参照してください。

構文

bool **NamedParameters**

プロパティ値

プロパティ値は ConnectString (NamedParameters='true'/'false') によって設定されます。または、ユーザが AseConnection のインスタンスを介して直接設定できます。

アクセス

読み込みと書き込み

Open メソッド

説明

事前に指定された接続文字列を使用して、データベースへの接続をオープンします。

構文

void **Open( )**

実装

IDbConnection.Open

使用法

- 接続プール内にあるオープンされている接続を使用できる場合、AseConnection はその接続を使用します。使用できない場合は、データ・ソースへの新しい接続を確立します。
- スコープ外にある AseConnection はクローズされません。そのため、Close または Dispose を呼び出して、明示的に接続をクローズしてください。

State プロパティ

説明　接続の現在のステータスを表します。

構文

```
ConnectionState State
```

アクセス　読み込み専用

デフォルト　Closed

実装

```
IDbConnection.State
```

参照　「接続ステータスの確認」(34 ページ)

StateChange イベント

説明　接続のステータスが変わったときに発生します。

構文

```
event StateChangeEventHandler StateChange
```

使用法　イベント・ハンドラが、このイベントに関連するデータを含む StateChangeEventArgs 型の引数を受け取ります。StateChangeEventArgs の 2 つのプロパティ、CurrentState と OriginalState に、このイベントに固有の情報が記載されます。

TraceEnter、TraceExit イベント

説明　デバッグ目的で、アプリケーション内でのデータベース・アクティビティをトレースします。

構文

```
public delegate void TraceEnterEventHandler(AseConnection
  connection, object source, string method, object[] parameters);
public delegate void TraceExitEventHandler(AseConnection
  connection, object source, string method, object returnValue);
```

使用法　TraceEnter と TraceExit イベントは、独自のトレース方法を組み込むときに使用します。このイベントは、個々の接続インスタンスに固有です。これによって、接続ごとのログをそれぞれ異なるファイルに記録できます。このイベントは無視することも、他のトレース用にプログラムすることもできます。また .NET イベントを使用することで、1 つの接続オブジェクトに対して複数のイベント・ハンドラを設定できます。これによって、ウィンドウとファイルの両方に、同時にイベントのログを出力できます。

AseDataAdapter クラス

説明

DataSet クラスと Adaptive Server の間でリンクの機能を果たします。それには 2 つの重要なメソッドが使用されます。

- Fill() - サーバから取得したデータで DataSet を更新する。
- Update() - DataSet に対して行った変更をサーバに適用する。

AseDataAdapter Fill または Update メソッドを使用するときに接続がオープンされていない場合は、このメソッドで接続をオープンできます。

基本クラス

Component

実装

IDbDataAdapter、IDisposable

使用法

DataSet を使用すると、データをオフラインで操作できます。AseDataAdapter は、DataSet に一連の SQL 文を関連付けるメソッドを提供します。

参照

「AseDataAdapter を使用したデータへのアクセスと操作」(49 ページ)、「データに対するアクセスと操作」(36 ページ)

AseDataAdapter コンストラクタ

説明

AseDataAdapter オブジェクトを初期化します。

構文 1

AseDataAdapter( )

構文 2

AseDataAdapter(AseCommand *selectCommand* **)**

構文 3

AseDataAdapter(string *selectCommandText*, AseConnection *selectConnection* **)**

構文 4

AseDataAdapter(string *selectCommandText*, string *selectConnectionString* **)**

パラメータ

selectCommand：Fill 中に、DataSet に格納するレコードをデータ・ソースから選択するために使用される AseCommand オブジェクト。

selectCommandText：AseDataAdapter の SelectCommand プロパティで使用する Select 文またはストアド・プロシージャ。

selectConnection：データベースへの接続を定義する AseConnection オブジェクト。

selectConnectionString：Adaptive Server データベースへの接続文字列。

例

次のコードは AseDataAdapter オブジェクトを初期化します。

```
AseDataAdapter da = new AseDataAdapter( "SELECT emp_id,
emp_lname FROM employee," conn );
```

AcceptChangesDuringFill プロパティ

説明　DataTable に DataRow が追加された後で、そのローに対して AcceptChanges を呼び出すかどうかを指定する値を表します。

構文　bool **AcceptChangesDuringFill**

アクセス　読み込みと書き込み

使用法　このプロパティを “true” に設定すると、DataAdapter は DataRow に対して AcceptChanges 関数を呼び出します。“false” に設定すると、AcceptChanges は呼び出されず、新しく追加されたローは挿入対象のローとして扱われます。デフォルトは “true” です。

ContinueUpdateOnError プロパティ

説明　ローの更新中にエラーが発生したときに、例外を生成するかどうかを指定する値を表します。

構文　bool **ContinueUpdateOnError**

アクセス　読み込みと書き込み

使用法

- デフォルトは “false” です。このプロパティを “true” に設定すると、例外を生成せずに更新が続行されます。
- ContinueUpdateOnError が “true” の場合は、ローの更新中にエラーが発生しても例外は返されません。ローの更新がスキップされ、そのローの RowError プロパティにエラー情報が記録されます。DataAdapter は後続のローの更新を継続します。
- ContinueUpdateOnError が “false” の場合は、エラーの発生時に例外が返されます。

DeleteCommand プロパティ

説明　Update() が呼び出されたとき、DataSet 内の削除されたローに対応するデータベース内のローを削除するためにデータベースに対して実行される AseCommand オブジェクトを表します。

構文　AseCommand **DeleteCommand**

アクセス　読み込みと書き込み

使用法　既存の AseCommand オブジェクトに DeleteCommand を割り当てた場合、AseCommand オブジェクトのクローンは作成されません。DeleteCommand は、既存の AseCommand に対する参照を維持します。

Fill メソッド

説明

データベースから取得したデータで、DataSet または DataTable オブジェクトのローを追加またはリフレッシュします。

構文 1

int **Fill(** DataSet *dataSet* **)**

構文 2

int **Fill(** DataSet *dataSet*, string *srcTable* **)**

構文 3

int **Fill(** DataSet *dataSet*, int *startRecord*, int *maxRecords*, string *srcTable* **)**

構文 4

int **Fill(** DataTable *dataTable* **)**

パラメータ

dataSet：レコードとスキーマ (オプション) が書き込まれる DataSet。

srcTable：テーブル・マッピングに使用するソース・テーブルの名前。

startRecord：処理を開始するレコード番号 (0 から始まる)。

maxRecords：DataSet に読み込まれるレコードの最大数。

dataTable：レコードとスキーマ (オプション) が書き込まれる DataTable。

戻り値

DataSet での追加またはリフレッシュが成功したローの数。

使用法

- StartRecord 引数を使用して、DataSet にコピーされるレコード数を制限している場合でも、AseDataAdapter クエリでは、すべての該当レコードがデータベースからクライアントにフェッチされます。結果セットが大きい場合は、パフォーマンスに重大な影響を与える可能性があります。
- 結果セットの前方向への読み込み専用で問題ない場合は、代替方法として、変更を実行する SQL 文 (ExecuteNonQuery) などとともに AseDataReader を使用できます。別の代替方法として、必要な結果セットだけを返すストアド・プロシージャを記述することもできます。
- SelectCommand がローを返さない場合は、DataSet にテーブルは追加されず、例外も発生しません。

参照

サポートされている fill メソッドの一覧については、.NET Framework のドキュメントで DdDataAdapter.fill() の説明を参照してください。

FillError イベント

説明

fill 操作中にエラーが発生したときに返されます。

構文

event FillErrorEventHandler **FillError**

使用法

FillError イベントを使用すると、エラーの発生後に Fill 操作を続行するかどうかをユーザが指定できます。FillError イベントは、次のような場合に発生する可能性があります。

- DataSet に追加するデータを共通言語ランタイム型に変換することで、データの精度が失われる場合。
- 追加するローに含まれるデータが、DataSet 内の DataColumn に強制される制約に違反している場合。

FillSchema メソッド

説明　1 つ以上の DataTable を DataSet に追加して、データ・ソース内のスキーマと一致するようにスキーマを設定します。

構文 1　DataTable[ ] **FillSchema(** DataSet *dataSet*, SchemaType *schemaType* **)**

構文 2　DataTable[ ] **FillSchema(** DataSet *dataSet*, SchemaType *schemaType*, string *srcTable* **)**

構文 3　DataTable **FillSchema(** DataTable *dataTable*, SchemaType *schemaType* **)**

パラメータ　**dataSet**：レコードとスキーマ (オプション) が書き込まれる DataSet。

schemaType：スキーマの挿入方法を指定する SchemaType 値の 1 つ。

srcTable：テーブル・マッピングに使用するソース・テーブルの名前。

dataTable：DataTable。

戻り値　構文 1 と 2 の場合、戻り値は DataSet に追加された DataTable オブジェクトのコレクションに対する参照です。構文 3 の場合、戻り値は 1 つの DataTable に対する参照です。

参照　「AseDataAdapter スキーマ情報の取得」(61 ページ)、.NET Framework help (http://msdn.microsoft.com/)

GetFillParameters

説明　Select 文の実行時にユーザによって設定されるパラメータを取得します。

構文　AseParameter[ ] **GetFillParameters( )**

戻り値　ユーザが設定したパラメータを含む IDataParameter オブジェクトの配列。

実装　IDataAdapter.GetFillParameters

InsertCommand プロパティ

説明　Update() が呼び出されたとき、DataSet 内の挿入されたローに対応するデータベース内のローを追加するためにデータベースに対して実行される AseCommand を表します。

構文　AseCommand **InsertCommand**

アクセス　読み込みと書き込み

使用法

- 既存の AseCommand オブジェクトに InsertCommand を割り当てた場合、AseCommand のクローンは作成されません。InsertCommand は、既存の AseCommand に対する参照を維持します。
- このコマンドによって複数のローが返された場合、AseCommand オブジェクトの UpdatedRowSource プロパティの設定に応じて、それらのローが DataSet に追加されます。

参照

「Update メソッド」(130 ページ)、「AseCommand オブジェクトを使用したローの挿入、更新、削除」(42 ページ)、「AseDataAdapter オブジェクトを使用したローの挿入、更新、削除」(51 ページ)

MissingMappingAction プロパティ

説明

一致するテーブルまたはカラムが受信データにない場合に実行するアクションを決定します。

構文

MissingMappingAction **MissingMappingAction**

アクセス

読み込みと書き込み

プロパティ値

MissingMappingAction 値の 1 つ。デフォルトは Passthrough です。

実装

IDataAdapter.MissingMappingAction

MissingSchemaAction プロパティ

説明

既存の DataSet スキーマが受信データと一致しないときに実行するアクションを決定します。

構文

MissingSchemaAction **MissingSchemaAction**

アクセス

読み込みと書き込み

プロパティ値

MissingSchemaAction 値の 1 つ。デフォルトは Add です。

実装

IDataAdapter.MissingSchemaAction

RowUpdated イベント

説明

更新時、データ・ソースに対してコマンドが実行された後に発生します。更新が試行され、イベントが初期化されます。

構文

event AseRowUpdatedEventHandler **RowUpdated**

使用法

イベント・ハンドラが、このイベントに関連するデータを含む AseRowUpdatedEventArgs 型の引数を受け取ります。次の AseRowUpdatedEventArgs プロパティには、このイベントに固有の情報が記載されます。

- Command
- Errors
- RecordsAffected
- Row
- StatementType
- Status
- TableMapping

詳細については、.NET Framework のドキュメントで OleDbDataAdapter.RowUpdated Event の説明を参照してください。

RowUpdating イベント

説明

更新時、データ・ソースに対してコマンドが実行される前に発生します。更新が試行され、イベントが初期化されます。

構文

event AseRowUpdatingEventHandler **RowUpdating**

使用法

イベント・ハンドラが、このイベントに関連するデータを含む AseRowUpdatingEventArgs 型の引数を受け取ります。次の AseRowUpdatingEventArgs プロパティには、このイベントに固有の情報が記載されます。

- Command
- Errors
- Row
- StatementType
- Status
- TableMapping

詳細については、.NET Framework のドキュメントで OleDbDataAdapter.RowUpdating Event の説明を参照してください。

SelectCommand プロパティ

説明 Fill または FillSchema で、DataSet にコピーする結果セットをデータベースから取得するために使用される AseCommand を表します。

構文 AseCommand **SelectCommand**

アクセス 読み込みと書き込み

使用法

- 事前に作成された AseCommand に SelectCommand を割り当てた場合、AseCommand のクローンは作成されません。SelectCommand は、その AseCommand オブジェクトに対する参照を維持します。
- SelectCommand がローを返さない場合は、DataSet にテーブルが追加されず、例外も発生しません。
- Select 文は、AseDataAdapter コンストラクタ内でも指定できます。

TableMappings プロパティ

説明 ソース・テーブルと DataTable とのマスタ・マッピングを提供するコレクションを表します。

構文 DataTableMappingCollection **TableMappings**

アクセス 読み込み専用

使用法

- デフォルト値は空のコレクションです。
- 変更を調整するときに、AseDataAdapter は DataTableMappingCollection コレクションを使用して、データ・ソースで使われるカラム名と DataSet で使われるカラム名を関連付けます。

Update メソッド

説明 データベース内のテーブルを、DataSet に対して行われた変更で更新します。

構文 1 int **Update(** DataSet *dataSet* **)**

構文 2 int **Update(** DataSet *dataSet*, string *srcTable* **)**

構文 3 int **Update(** DataTable *dataTable* **)**

構文 4 int **Update(** DataRow[ ] *dataRows* **)**

パラメータ

dataSet：更新するレコードとスキーマ (オプション) を持つ DataSet。

srcTable：テーブル・マッピングに使用するソース・テーブルの名前。

dataTable：レコードとスキーマ (オプション) で更新する DataTable。

dataRows：データ・ソースの更新に使用する DataRow オブジェクトの配列。

戻り値 DataSet からの更新に成功したローの数。

使用法

Update は、データ・セット内にある挿入、更新、削除された各ローに対して、それぞれ InsertCommand、UpdateCommand、DeleteCommand プロパティを使用して実行されます。

参照

「DeleteCommand プロパティ」(125 ページ)、「InsertCommand プロパティ」(127 ページ)、「UpdateCommand プロパティ」(131 ページ)、「AseDataAdapter オブジェクトを使用したローの挿入、更新、削除」(51 ページ)、.NET Framework のドキュメント

UpdateCommand プロパティ

説明

Update() が呼び出されたとき、DataSet 内の更新されたローに対応するデータベース内のローを更新するためにデータベースに対して実行される AseCommand を表します。

構文

AseCommand **UpdateCommand**

アクセス

読み込みと書き込み

使用法

- 事前に作成された AseCommand に UpdateCommand を割り当てた場合、AseCommand のクローンは作成されません。UpdateCommand は、その AseCommand オブジェクトに対する参照を維持します。
- このコマンドによって複数のローが返された場合、AseCommand オブジェクトの UpdatedRowSource プロパティの設定に応じて、それらのローが DataSet にマージされます。

参照

「Update メソッド」(130 ページ)

AseDataReader クラス

説明

クエリまたはストアド・プロシージャで、前方向への読み込み専用の結果セットが取得されます。

基本クラス

MarshalByRefObject

実装

IDataReader、IDisposable、IDataRecord、IEnumerable、IListSource

使用法

- AseDataReader には、コンストラクタはありません。AseDataReader オブジェクトを取得するには、次のように AseCommand を実行します。

```
AseCommand cmd = new AseCommand("Select emp_id from employee", conn );
AseDataReader reader = cmd.ExecuteReader();
```

- AseDataReader では、前方向にのみ移動できます。より柔軟な方法で結果を操作する必要がある場合は、AseDataAdapter を使用します。
- カーソルの使用時、AseDataReader は必要なだけのローを取得します。詳細については、「AseConnection コンストラクタ」(114 ページ) の ConnectionString プロパティの UseCursor パラメータを参照してください。

参照 「ExecuteReader メソッド」(104 ページ)、「データに対するアクセスと操作」(36 ページ)

Close メソッド

説明 AseDataReader クラスをクローズします。

構文 void **Close( )**

実装 IDataReader.Close

使用法 AseDataReader を使用した後は、Close メソッドを明示的に呼び出してください。

Depth プロパティ

説明 現在のローのネストの深さを示す値を表します。一番外側のテーブルの深さはゼロです。

構文 int **Depth**

アクセス 読み込み専用

プロパティ値 現在のローのネストの深さ。

実装 IDataReader.Depth

Dispose メソッド

説明 オブジェクトに関連付けられたリソースを解放します。

構文 void **Dispose( )**

FieldCount プロパティ

説明　結果セット内のカラム数を表します。

構文

```
int FieldCount
```

アクセス　読み込み専用

プロパティ値　有効なレコード・セットに位置していない場合は 0、それ以外の場合は現在のレコードのカラム数。デフォルトは -1 です。

実装

```
IDataRecord.FieldCount
```

使用法　有効なレコード・セットに位置していない場合、このプロパティの値は 0 です。それ以外の場合は、現在のレコードのカラム数になります。デフォルトは -1 です。

GetBoolean メソッド

説明　指定したカラムの値を Boolean として取得します。

構文

```
bool GetBoolean( int ordinal )
```

パラメータ　**ordinal**：値を取得するカラムを表す序数。番号付けは 0 から始まります。

戻り値　カラムの値。

実装

```
IDataRecord.GetBoolean
```

使用法　変換は実行されません。bit 型のカラムからデータを取得するときに、このメソッドを使用してください。

GetByte メソッド

説明　指定したカラムの値を Byte として取得します。

構文

```
byte GetByte( int ordinal )
```

パラメータ　**ordinal**：値を取得するカラムを表す序数。番号付けは 0 から始まります。

戻り値　カラムの値。

実装

```
IDataRecord.GetByte
```

使用法　変換は実行されません。tinyint 型のカラムからデータを取得するときに、このメソッドを使用してください。

GetBytes メソッド

説明

指定したカラム・オフセットからのバイト・ストリームを、配列としてバッファに読み込みます。読み込みは、指定したバッファ・オフセットから開始されます。

構文

long **GetBytes(** int *ordinal*, long *dataIndex*, byte[ ] *buffer*, int *bufferIndex*, int *length* **)**

パラメータ

ordinal : 値を取得するカラムを表す序数。番号付けは 0 から始まります。

dataIndex : バイトの読み込みを開始するカラム値内のインデックス。

buffer : データの格納先となる配列。

bufferIndex : データのコピーを開始する配列内のインデックス。

length : 指定したバッファにコピーする最大長。

戻り値

読み込まれたバイト数。

実装

IDataRecord.GetBytes

使用法

- GetBytes は、フィールド内の使用可能なバイト数を返します。ほとんどの場合、これは正確なフィールド長になります。ただし、GetBytes を使用してフィールドからバイトをすでに取得している場合は、返されるバイト数がフィールドの実際の長さよりも小さくなることがあります。この現象は、AseDataReader クラスが大規模なデータ構造体をバッファに読み込んでいるときなどに起こります。
- null 参照 (Visual Basic の場合は “Nothing”) のバッファを渡すと、GetBytes はフィールド長をバイト単位で返します。
- 変換は実行されません。image、binary、timestamp、varbinary 型のカラムからデータを取得するときに、このメソッドを使用してください。

GetChar メソッド

説明

指定したカラムの値を文字として取得します。

構文

char **GetChar(** int *ordinal* **)**

パラメータ

ordinal : 値を取得するカラムを表す序数。番号付けは 0 から始まります。

戻り値

カラムの値。

実装

IDataRecord.GetChar

使用法

- 変換は実行されません。tinyint、char(1)、varchar(1) 型のカラムからデータを取得するときに、このメソッドを使用してください。
- このメソッドを呼び出す前に、AseDataReader.IsDBNull を呼び出して null 値をチェックしてください。

参照

「IsDBNull メソッド」(143 ページ)

GetChars メソッド

説明　指定したカラム・オフセットからの文字ストリームを、配列としてバッファに読み込みます。読み込みは、指定したバッファ・オフセットから開始されます。

構文

long **GetChars(** int *ordinal*, long *dataIndex*, char[ ] *buffer*, int *bufferIndex*, int *length* **)**

パラメータ

ordinal：カラムを表す、0 から始まる序数。

dataIndex：read 操作を開始するロー内のインデックス。

buffer：データのコピー先のバッファ。

bufferIndex：read 操作を開始するバッファのインデックス。

length：読み込む文字数。

戻り値　実際に読み込まれた文字数。

実装

IDataRecord.GetChars

使用法

GetChars は、フィールド内の使用可能な文字数を返します。ほとんどの場合、これは正確なフィールド長になります。ただし、GetChars を使用してフィールドから文字をすでに取得している場合は、返される文字数がフィールドの実際の長さよりも小さくなることがあります。この現象は、AseDataReader が大規模なデータ構造体をバッファに読み込んでいるときなどに起こります。

null 参照 (Visual Basic の場合は Nothing) のバッファを渡すと、GetChars はフィールド長を文字単位で返します。

変換は実行されません。text、char、varchar 型のカラムからデータを取得するときに、このメソッドを使用してください。

参照　「BLOB の処理」(68 ページ)

GetDataTypeName メソッド

説明　ソースのデータ型の名前を取得します。

構文

string **GetDataTypeName(** int *index* **)**

パラメータ

index：カラムを表す、0 から始まる序数。

戻り値　バックエンドのデータ型の名前。

実装

IDataRecord.GetDataTypeName

GetDateTime メソッド

説明 指定したカラムの値を DateTime オブジェクトとして取得します。

構文

```
DateTime GetDateTime( int ordinal )
```

パラメータ **ordinal**：カラムを表す、0 から始まる序数。

戻り値 指定されたカラムの値。

実装

```
IDataRecord.GetDateTime
```

使用法

- 変換は実行されません。そのため、取得するデータは DateTime オブジェクトである必要があります。
- このメソッドを呼び出す前に、AseDataReader.IsDBNull を呼び出して null 値をチェックしてください。
- データベース内の対応する Adaptive Server 型が datetime、smalldatetime、date (Adaptive Server 12.5.1 以降)、time (Adaptive Server 12.5.1 以降) の場合は、このメソッドを使用してください。

参照 「IsDBNull メソッド」(143 ページ)

GetDecimal メソッド

説明 指定したカラムの値を Decimal オブジェクトとして取得します。

構文

```
decimal GetDecimal( int ordinal )
```

パラメータ **ordinal**：値を取得するカラムを表す序数。番号付けは 0 から始まります。

戻り値 指定されたカラムの値。

実装

```
IDataRecord.GetDecimal
```

使用法

- 変換は実行されません。decimal、numeric、smallmoney、money 型のカラムからデータを取得するときに、このメソッドを使用してください。
- このメソッドを呼び出す前に、AseDataReader.IsDBNull を呼び出して null 値をチェックしてください。

参照 「IsDBNull メソッド」(143 ページ)

GetDouble メソッド

説明 指定したカラムの値を倍精度浮動小数点数として取得します。

構文

```
double GetDouble( int ordinal )
```

パラメータ **ordinal**：値を取得するカラムを表す序数。番号付けは 0 から始まります。

戻り値 指定されたカラムの値。

実装

```
IDataRecord.GetDouble
```

使用法

- このメソッドを呼び出す前に、AseDataReader.IsDBNull を呼び出して null 値をチェックしてください。
- 変換は実行されません。そのため、取得するデータは倍精度浮動小数点数である必要があります。
- 精度が 16 以上の Adaptive Server データ型 float には GetDouble を使用してください。精度が 16 未満の Adaptive Server データ型 real と float には GetFloat を使用してください。

参照

「IsDBNull メソッド」(143 ページ)

GetFieldType メソッド

説明

オブジェクトのデータ型を取得します。

構文

Type **GetFieldType(** int *index* **)**

パラメータ

index：カラムを表す、0 から始まる序数。

戻り値

オブジェクトのデータ型を取得します。

実装

IDataRecord.GetFieldType

GetFloat メソッド

説明

指定したカラムの値を単精度浮動小数点数として取得します。

構文

float **GetFloat(** int *ordinal* **)**

パラメータ

ordinal：値を取得するカラムを表す序数。番号付けは 0 から始まります。

戻り値

指定されたカラムの値。

実装

IDataRecord.GetFloat

使用法

- 変換は実行されません。そのため、取得するデータは単精度浮動小数点数である必要があります。
- このメソッドを呼び出す前に、AseDataReader.IsDBNull を呼び出して null 値をチェックしてください。
- 精度が 16 未満の Adaptive Server データ型 real と float には GetFloat を使用してください。精度が 16 以上の Adaptive Server データ型 float には GetDouble を使用してください。

参照

「IsDBNull メソッド」(143 ページ)

GetInt16 メソッド

説明　指定したカラムの値を、16 ビットの符号付き整数として取得します。

構文　short **GetInt16(** int *ordinal* **)**

パラメータ　**ordinal：**値を取得するカラムを表す序数。番号付けは 0 から始まります。

戻り値　指定されたカラムの値。

実装　IDataRecord.GetInt16

使用法　変換は実行されません。smallint 型のカラムからデータを取得するときに、このメソッドを使用してください。

GetInt32 メソッド

説明　指定したカラムの値を、32 ビットの符号付き整数として取得します。

構文　int **GetInt32(** int *ordinal* **)**

パラメータ　**ordinal：**値を取得するカラムを表す序数。番号付けは 0 から始まります。

戻り値　指定されたカラムの値。

実装　IDataRecord.GetInt32

使用法　変換は実行されません。int[eger] 型のカラムからデータを取得するときに、このメソッドを使用してください。

GetList メソッド

説明　IListSource を実装します。

構文　IList GetList();

実装　IListSource

使用法　.NET DataGrid オブジェクトの DataSource プロパティを AseDataReader に設定できます。次に、グリッドはこのメソッドを使用して、AseDataReader の結果をそのセルに直接バインドします。

通常、このメソッドを直接使用することはありません。AseDataReader はこのメソッドを実装しているので、次のように実行できます。

```
using(AseCommand cmd = new AseCommand(select total_sales from titles
   where title_id = 'BU1032', conn)
{
   using(AseDataReader rdr = cmd.ExecuteReader())
   {
      MyGrid.DataSource = rdr;
   }
}
```

GetName メソッド

説明　指定したカラムの名前を取得します。

構文

```
string GetName( int index )
```

パラメータ　**index：**0 から始まるカラム・インデックス。

戻り値　指定されたカラムの名前。

実装

```
IDataRecord.GetName
```

GetOrdinal メソッド

説明　名前を指定したカラムの序数を取得します。

構文

```
int GetOrdinal( string name )
```

パラメータ　**name：**カラム名。

戻り値　カラムを表す、0 から始まる序数。

実装

```
IDataRecord.GetOrdinal
```

使用法

GetOrdinal は、最初に大文字と小文字を区別して検索を実行します。それに失敗した場合は、大文字と小文字を区別せずに 2 回目の検索を実行します。

GetOrdinal は、日本語のかなの全角と半角を区別しません。

名前ベースの検索よりも序数ベースの検索の方が効率的なため、ループ内での GetOrdinal の呼び出しは非効率です。GetOrdinal を 1 回呼び出して、その結果を整数の変数に割り当ててループ内で使用すると、処理時間を短縮できます。

GetSchemaTable メソッド

説明　AseDataReader のカラム・メタデータを説明する DataTable を返します。

構文

```
DataTable GetSchemaTable( )
```

戻り値　カラム・メタデータを説明する DataTable。

実装

```
IDataReader.GetSchemaTable
```

使用法　このメソッドは、各カラムのメタデータを次の順序で返します。

- ColumnName
- ColumnOrdinal
- ColumnSize
- DataType
- ProviderType
- IsLong

- AllowDBNull
- IsReadOnly
- IsRowVersion
- IsUnique
- IsKeyColumn
- IsAutoIncrement
- BaseSchemaName
- BaseCatalogName
- BaseTableName
- BaseColumnName

これらのカラムの詳細については、.NET Framework のドキュメントで SqlDataReader.GetSchemaTable の説明を参照してください。

参照 「DataReader スキーマ情報の取得」(48 ページ)

GetString メソッド

説明 指定したカラムの値を文字列として取得します。

構文

string **GetString(** int *ordinal* **)**

パラメータ **ordinal**：値を取得するカラムを表す序数。番号付けは 0 から始まります。

戻り値 指定されたカラムの値。

実装

IDataRecord.GetString

使用法 変換は実行されません。そのため、取得するデータは文字列である必要があります。

このメソッドを呼び出す前に、AseDataReader.IsDBNull を呼び出して null 値をチェックしてください。

参照 「IsDBNull メソッド」(143 ページ)

GetUInt16 メソッド

説明　指定したカラムの値を、16 ビットの符号なし整数として取得します。

構文

```
UInt16 GetUInt16( int ordinal )
```

パラメータ　**ordinal**値を取得するカラムを表す序数。番号付けは 0 から始まります。

戻り値　指定されたカラムの値。

使用法　変換は実行されません。そのため、取得するデータは 16 ビットの整数である必要があります。

GetUInt32 メソッド

説明　指定したカラムの値を、32 ビットの符号なし整数として取得します。

構文

```
UInt32 GetUInt32( int ordinal )
```

パラメータ　**ordinal**値を取得するカラムを表す序数。番号付けは 0 から始まります。

戻り値　指定されたカラムの値。

使用法　変換は実行されません。そのため、取得するデータは 32 ビットの整数である必要があります。

GetUInt64 メソッド

説明　指定したカラムの値を、64 ビットの符号なし整数として取得します。

構文

```
UInt64 GetUInt64( int ordinal )
```

パラメータ　**ordinal**値を取得するカラムを表す序数。番号付けは 0 から始まります。

戻り値　指定されたカラムの値。

GetValue メソッド

説明　指定した序数に対応するカラムの値を、ネイティブ・フォーマットで取得します。

構文

```
object GetValue( int ordinal )
```

パラメータ　**ordinal**：値を取得するカラムを表す序数。番号付けは 0 から始まります。

戻り値　対応するカラムの値。

実装　IDataRecord.GetValue

使用法　このメソッドでは、null のデータベース・カラムに対して DBNull が返されます。

GetValues メソッド

説明

現在のローのすべての属性カラムを取得します。

構文

int **GetValues(** object[ ] *values* **)**

パラメータ

values：結果セット内のすべてのローを保持するオブジェクトの配列。

戻り値

配列内のオブジェクトの数。

実装

IDataRecord GetValues

使用法

- ほとんどのアプリケーションでは、各カラムを個別に取得するよりも、GetValues メソッドを使用してすべてのカラムを取得する方が効率的です。
- 対象のローに含まれるカラム数よりも配列長が短い Object 配列を渡すことができます。Object 配列が保持しているデータ量だけが、配列にコピーされます。対象のローに含まれるカラム数よりも配列長が長い Object 配列も渡すことができます。
- このメソッドでは、null のデータベース・カラムに対して DBNull が返されます。
- 指定した序数に対応するカラムの値を、ネイティブ・フォーマットで取得します。

IsClosed プロパティ

説明

AseDataReader がクローズされている場合は “true” を、それ以外の場合は “false” を返します。

構文

bool **IsClosed**

アクセス

読み込み専用

プロパティ値

AseDataReader がクローズされている場合は “true”、それ以外の場合は “false”。

実装

IDataReader.IsClosed

使用法

AseDataReader をクローズした後で呼び出すことができるのは、IsClosed と RecordsAffected プロパティだけです。

IsDBNull メソッド

説明　カラムに null 値が含まれるかどうかを示す値を返します。

構文

bool **IsDBNull(** int *ordinal* **)**

パラメータ　**ordinal：**カラムを表す、0 から始まる序数。

戻り値　指定したカラムの値が DBNull と等価の場合は “true”、それ以外の場合は “false”。

実装

IDataRecord.IsDbNull

使用法　型指定された Get メソッド (GetByte、GetChar など) を呼び出す前に、例外の発生を防止する目的で、このメソッドを呼び出して null のカラム値をチェックしてください。

Item プロパティ

説明　ネイティブ・フォーマットでのカラムの値を表します。C# で、このプロパティは AseDataReader クラスのインデクサになります。

構文 1

object **this[** int *index* **]**

構文 2

object **this[** string *name* **]**

パラメータ　**index：**カラムの序数。

name：カラム名。

アクセス　読み込み専用

実装

IDataRecord.Item

NextResult メソッド

説明　バッチ SQL 文の結果を読み込むときに、AseDataReader を次の結果へ進めます。

構文

bool **NextResult( )**

戻り値　さらに結果セットがある場合は “true”。それ以外の場合は “false”。

実装

IDataReader.NextResult

使用法　バッチ SQL 文の実行によって生成される複数の結果を処理する目的で使用します。

デフォルトでは、データ・リーダは最初の結果に置かれます。

Read メソッド

説明 結果セットの次のローを読み込み、AseDataReader をそのローに進めます。

構文 bool **Read( )**

戻り値 さらにローがある場合は “true” を返します。それ以外の場合は “false” を返します。

実装 IDataReader.Read

使用法 AseDataReader のデフォルトの位置は、最初のレコードの前です。そのため、データへのアクセスを開始するには Read を呼び出す必要があります。

例 次のコードは、結果内の 1 つのカラムの値をリスト・ボックスに書き込みます。

```
while( reader.Read() )
{
   listResults.Items.Add( reader.GetValue( 0 ).ToString() );
}
listResults.EndUpdate();
reader.Close();
```

RecordsAffected プロパティ

説明 SQL 文の実行によって変更、挿入、削除されたローの数を表します。

構文 int **RecordsAffected**

アクセス 読み込み専用

プロパティ値 変更、挿入、または削除されたローの数。影響を受けたローがない場合または文が失敗した場合は 0、Select 文の場合は -1 になります。

実装 IDataReader.RecordsAffected

使用法

- 変更、挿入、または削除されたローの数。影響を受けたローがない場合または文が失敗した場合、値は 0 です。Select 文の場合、値は -1 です。
- このプロパティの値は累積です。たとえば、バッチ・モードで 2 つのレコードが挿入された場合、RecordsAffected の値は 2 になります。
- AseDataReader をクローズした後で呼び出すことができるのは、IsClosed と RecordsAffected プロパティだけです。

AseDbType 列挙型

Adaptive Server のデータ型を指定します。データ型のマッピングの詳細については、表 5-2 を参照してください。

メンバ

- Binary
- Bit
- Char
- Date
- DateTime
- Decimal
- Double
- Float
- Integer
- Image
- LongVarChar
- Money
- Nchar
- Numeric
- NVarChar
- Real
- SmallDateTime
- SmallMoney
- Text
- Time
- TimeStamp
- TinyInt
- UniChar
- UniVarChar
- VarBinary
- VarChar

注意　Numeric と Decimal の精度は、Adaptive Server の精度の 38 ではなく 26 に制限されています。

データ型のマッピング

次の表は、Adaptive Server ADO.NET Data Provider でのデータ型のマッピングを示します。

表 *5-2: Adaptive Server ADO.NET* でのデータ型のマッピング

Adaptive Server データベース型	AseDbType 列挙型	.NET Dbtype 列挙型	.Net クラス名
binary	Binary	Binary	Byte[]
bigint	BigInt	Int64	Int64
bit	Bit	Boolean	Boolean
char	Char	AnsiStringFixedLength	String
date	Date	Date	DateTime
datetime	DateTime	DateTime	DateTime
decimal	Decimal	Decimal	Decimal
double	Double	Double	Double
float (<16)	Real	Single	Single
float (>=16)	Double	Double	Double
image	Image	Binary	Byte[]
int[eger]	Integer	Int32	Int32
money	Money	Currency	Decimal
nchar	NChar	AnsiStringFixedLength	String
nvarchar	NVarChar	AnsiString	String
numeric	Numeric	VarNumeric	Decimal
real	Real	Single	Single
smalldatetime	SmallDateTime	DateTime	DateTime
smallint	SmallInt	Int16	Int16
smallmoney	SmallMoney	Currency	Decimal
text	text	AnsiString	String
time	time	time	DateTime
timestamp	TimeStamp	Binary	Byte[]
tinyint	TinyInt	Byte	Byte
unichar	UniChar	StringFixedLength	String
unitext	Unitext	String	String
univarchar	UniVarChar	String	String
unsignedbigint	UnsignedBigint	UInt64	UInt64
unsignedint	UnsignedInt	UInt32	UInt32
unsignedsmallint	UnsignedSmallInt	UInt16	UInt16
varbinary	VarBinary	Binary	Byte[]
varchar	VarChar	AnsiString	String

AseError クラス

説明　データ・ソースから返された警告またはエラーに関する情報を収集します。

基本クラス　Object

AseError には、コンストラクタはありません。

参照　「エラー処理」(78 ページ)

ErrorNumber プロパティ

説明　エラー・メッセージの番号を表します。

構文　public int **MessageNumber**

アクセス　読み込み専用

Message プロパティ

説明　エラーの簡単な説明を表します。

構文　public string **Message**

アクセス　読み込み専用

SqlState プロパティ

説明　ANSI SQL 標準に従った、Adaptive Server 固有の 5 文字の SQL ステータスを表します。エラーが複数の場所で発生している場合、5 文字のエラー・コードはエラーの発生元を明らかにします。

構文　public string **SqlState**

アクセス　読み込み専用

ToString メソッド

説明　エラー・メッセージの完全なテキストを取得します。

構文　public string **ToString( )**

使用法　戻り値は、“AseError:” の後にメッセージが続く形式の文字列です。次に例を示します。

```
AseError:UserId or Password not valid.
```

説明　メッセージのステータスを表します。MsgNumber に対する変更子として使用されます。

構文　public int **State**

説明 メッセージの重大度を表します。

構文 public int **Severity**

説明 メッセージを送信しているサーバの名前を表します。

構文 public string **ServerName**

説明 メッセージの発生元であるストアド・プロシージャまたはリモート・プロシージャ・コール (RPC) の名前を表します。

構文 public string **ProcName**

説明 コマンド・バッチまたはストアド・プロシージャ内でエラーが発生した箇所の行番号を表します (該当する場合)。

構文 public int **LineNum**

説明 拡張メッセージに関連付けられます。

構文 public int **Status**

説明 このダイアログでアクティブな任意のトランザクションの現在のステータスを表します。

構文 public int **TranState**

説明 Adaptive Server サーバから送信されたエラー・メッセージであるかどうかを表します。

構文 bool **IsFromServer**

使用法 戻り値は “true” または “false” です。

```
if (ex.Errors[0].IsInformation )
    MessageBox.Show("ASE has reported the following error:" +
ex.Errors[0].Message);
```

説明 Adaptive Server ADO.NET Data Provider から送信されたエラー・メッセージであるかどうかを表します。

構文 bool **IsFromClient**

使用法 戻り値は “true” または “false” です。

```
if (ex.Errors[0].IsInformation )
    MessageBox.Show("ASE has reported the following error:" +
ex.Errors[0].Message);
```

説明 メッセージがエラーを表すかどうか判断されます。

構文 bool **IsError**

使用法 戻り値は “true” または “false” です。

```
if ( !ex.Errors[0].IsInformation )
    MessageBox.Show("Error:" + ex.Errors[0].Message):
```

説明 メッセージが、状況が正常でない可能性があることを示す警告であるかどうかを表します。

構文　bool **IsWarning**

使用法　戻り値は “true” または “false” です。

```
if ( !ex.Errors[0].IsInformation )
    MessageBox.Show(“Error:” + ex.Errors[0].Message):
```

説明　アクティブなカタログが変更されたなどの情報を知らせるメッセージであるかどうかを表します。

構文　bool **IsInformation**

使用法　戻り値は “true” または “false” です。

```
if ( !ex.Errors[0].IsInformation )
    MessageBox.Show(“Error:” + ex.Errors[0].Message):
```

AseErrorCollection クラス

説明　Adaptive Server ADO.NET Data Provider によって生成されたすべてのエラーを収集します。

基本クラス　オブジェクト

実装　ICollection、IEnumerable

AseErrorCollection には、コンストラクタはありません。通常、AseErrorCollection は AseException.Errors または InfoMessageArgs プロパティから取得されます。

参照　「Errors プロパティ」(150 ページ)、「エラー処理」(78 ページ)

CopyTo メソッド

説明　AseErrorCollection の要素を配列にコピーします。配列内の指定したインデックスからコピーが開始されます。

構文　void **CopyTo(** Array *array*, int *index* **)**

パラメータ　**array**：要素のコピー先となる配列。

index：配列の開始インデックス。

実装　ICollection.CopyTo

Count プロパティ

説明 コレクション内のエラーの数を表します。

構文 int **Count**

アクセス 読み込み専用

実装 ICollection.Count

Item プロパティ

説明 指定したインデックスの位置にあるエラーを表します。

構文 AseError **this[** int *index* **]**

パラメータ **index**：取得するエラーの 0 から始まるインデックス。

プロパティ値 指定したインデックスの位置にあるエラーが記録された AseError。

アクセス 読み込み専用

AseException クラス

説明 Adaptive Server が警告またはエラーを返したときに発生する例外を表します。

基本クラス SystemException

AseException には、コンストラクタはありません。通常、AseException オブジェクトは catch 内で宣言されます。次に例を示します。

```
...
catch (AseException ex )
{
   MessageBox.Show(ex.Message, "Error" );
}
```

参照 「エラー処理」(78 ページ)

Errors プロパティ

説明 1 つ以上の AseError オブジェクトのコレクションを表します。

構文 AseErrorCollection **Errors**

アクセス 読み込み専用

使用法 AseErrorCollection クラスには、AseError クラスのインスタンスが常に 1 つ以上含まれます。

Message プロパティ

説明　エラーを説明するテキストを返します。

構文　string **Message**

アクセス　読み込み専用

使用法　このメソッドは、最初の AseError のメッセージを返します。

AseFailoverException クラス

説明　HA クラスタに設定されているセカンダリ・サーバへの Adaptive Server のフェールオーバが成功したときに返される例外を表します。

基本クラス　AseException

AseFailoverException には、コンストラクタはありません。通常、AseFailoverException オブジェクトは catch 内で宣言されます。次に例を示します。

```
...
catch( AseFailoverException ex )
{
   MessageBox.Show( ex.Message, "Warning!" );
}
```

参照　「AseException クラス」(150 ページ)

Errors プロパティ

説明　データ・ソースから送信された警告のコレクションを表します。

構文　AseErrorCollection **Errors**

アクセス　読み込み専用

Message プロパティ

説明　データ・ソースから送信されたエラーの完全なテキストを表します。

構文　string **Message**

アクセス　読み込み専用

ToString メソッド

説明 InfoMessage イベントの文字列表現を取得します。

構文 string **ToString( )**

戻り値 InfoMessage イベントを表す文字列。

AseInfoMessageEventArgs クラス

説明 InfoMessage イベント・ハンドラに渡されたイベント引数を表します。

基本クラス EventArgs

Errors プロパティ

説明 サーバによって返された実際のエラー・オブジェクトのコレクションを表します。

構文 AseErrorCollection **Errors**

アクセス 読み込み専用

Message プロパティ

説明 エラー・メッセージを表します。

構文 string **Message**

アクセス 読み込み専用

AseInfoMessageEventHandler デリゲート

説明 AseConnection の InfoMessage イベントを処理するメソッドを表します。

構文 void **AseInfoMessageEventHandler (** object *sender*, AseInfoMessageEventArgs *e* **)**

パラメータ **sender**：イベントの発生元。

e：イベント・データが格納された AseInfoMessageEventArgs オブジェクト。

AseParameter クラス

説明　AseCommand のパラメータと、オプションとして DataSet カラムへのマッピングを表します。

基本クラス　MarshalByRefObject

実装　IDbDataParameter、IDataParameter

AseParameter コンストラクタ

構文 1　**AseParameter( )**

構文 2　**AseParameter(** string *parameterName*, object *value* **)**

構文 3　**AseParameter(** string *parameterName*, AseDbType *dbType* **)**

構文 4　**AseParameter(** string *parameterName*, AseDbType *dbType*, int *size* **)**

構文 5　**AseParameter(** string *parameterName*, AseDbType *dbType*, int *size*, string *sourceColumn* **)**

構文 6　**AseParameter(** string *parameterName*, AseDbType *dbType*, int *size*, ParameterDirection *direction*, bool *isNullable*, byte *precision*, byte *scale*, string *sourceColumn*, DataRowVersion *sourceVersion*, object *value* **)**

パラメータ

value：パラメータの値であるオブジェクト。

size：パラメータの長さです。

sourceColumn：マッピング対象のソース・カラムの名前。

parameterName：パラメータの名前。

dbType：AseDbType 値の 1 つ。

direction：ParameterDirection 値の 1 つ。

isNullable：フィールドの値として null を許可する場合は “true”、許可しない場合は “false”。

precision：Value の解決に適用する、小数点以上と以下両方の総桁数。

scale：Value の解決に適用する、小数点以下の桁数。

sourceVersion：DataRowVersion 値の 1 つ。

AseDbType プロパティ

説明 パラメータの AseDbType を表します。

構文 AseDbType **AseDbType**

アクセス 読み込みと書き込み

使用法

- AseDbType と DbType はリンクしています。そのため、DbType を設定すると、AseDbType が、サポートされる AseDbType に変更されます。
- 値は、AseDbType 列挙型のメンバである必要があります。

DbType プロパティ

説明 パラメータの DbType を表します。

構文 DbType **DbType**

アクセス 読み込みと書き込み

使用法

- AseDbType と DbType はリンクしています。そのため、DbType を設定すると、AseDbType が、サポートされている AseDbType に変更されます。
- 値は、DbType 列挙型のメンバである必要があります。

Direction プロパティ

説明 パラメータが入力専用、出力専用、双方向、またはストアド・プロシージャの戻り値パラメータのいずれであるかを示す値を表します。

構文 ParameterDirection **Direction**

アクセス 読み込みと書き込み

使用法 ParameterDirection が出力であり、関連付けられた AseCommand の実行が値を返さなかった場合、AseParameter には null 値が格納されます。最後の結果セットの最後のローが読み込まれたら、Output、InputOut、ReturnValue の各パラメータが更新されます。

IsNullable プロパティ

説明 パラメータが null 値を受け入れるかどうかを示す値を表します。

構文 bool **IsNullable**

アクセス 読み込みと書き込み

使用法 null 値を受け入れる場合は “true”、それ以外の場合は “false” (デフォルト)。null 値は、DBNull クラスを使用して操作します。

ParameterName プロパティ

説明　AseParameter の名前を表します。

構文　string **ParameterName**

アクセス　読み込みと書き込み

実装　IDataParameter.ParameterName

使用法

- Adaptive Server ADO.NET Data Provider は、*@name* パラメータで示される、位置を指定するパラメータを使用します。
- デフォルトは空の文字列です。
- 出力パラメータ (準備のありなしに関係なく) には、ユーザ指定のデータ型が必要です。

Precision プロパティ

説明　Value プロパティを表すために使用する最大桁数を表します。

構文　byte **Precision**

アクセス　読み込みと書き込み

実装　IDbDataParameter.Precision

使用法

- このプロパティの値は、Value プロパティを表すために使用する最大桁数です。デフォルト値は 0 で、これは Adaptive Server ADO.NET Data Provider が Value プロパティの精度を設定することを意味します。
- Precision プロパティは、decimal と numeric 型の入力パラメータにのみ使用します。

Scale プロパティ

説明　Value の解決に適用する小数点以下の桁数を表します。

構文　byte **Scale**

アクセス　読み込みと書き込み

実装　IDbDataParameter.Scale

使用法　デフォルトは 0 です。Scale プロパティは、decimal と numeric 型の入力パラメータにのみ使用します。

Size プロパティ

説明　カラム内のデータのバイト単位での最大サイズを表します。

構文　int **Size**

アクセス　読み込みと書き込み

実装　IDbDataParameter.Size

使用法

- このプロパティの値は、カラム内のデータのバイト単位での最大サイズです。デフォルト値は、パラメータ値から推論されます。
- Size プロパティは、binary と string 型で使用します。
- 可変長のデータ型では、Size プロパティはサーバに送信するデータの最大量を示します。たとえば、Size プロパティを使用して、サーバに送信するデータ量を、最初の 100 バイトまでの文字列値に制限できます。
- 明示的に設定しない場合は、指定したパラメータ値の実際のサイズから、このサイズが推論されます。固定長のデータ型では、Size の値は無視されます。この情報は参照目的で取得できます。この場合、パラメータ値をサーバに送信するときに Adaptive Server ADO.NET Data Provider が使用する最大バイト数が返されます。

SourceColumn プロパティ

説明　DataSet にマッピングされており、値の読み込みや書き込みに使用されるソース・カラムの名前を表します。

構文　string **SourceColumn**

アクセス　読み込みと書き込み

実装　IDbDataParameter.SourceColumn

使用法　SourceColumn に空の文字列以外が設定されている場合、パラメータの値は SourceColumn 名で指定されるカラムから取得されます。Direction が Input に設定されている場合、値は DataSet から取得されます。Direction が Output に設定されている場合、値はデータ・ソースから取得されます。Direction が InputOutput の場合は、両方の組み合わせです。

SourceVersion プロパティ

説明　Value を読み込むときに使用する DataRowVersion を表します。

構文　DataRowVersion **SourceVersion**

アクセス　読み込みと書き込み

実装　IDbDataParameter.SourceVersion

使用法

パラメータ値が Current と Original のどちらに設定されているかを判別するために、Update 中に UpdateCommand によって使用されます。これによって、プライマリ・キーが更新できるようになります。InsertCommand と DeleteCommand では、このプロパティは無視されます。このプロパティは、Item プロパティ、または DataRow オブジェクトの GetChildRows メソッドによって使用される DataRow のバージョンに設定されます。

ToString メソッド

説明

ParameterName が含まれる文字列を表します。

構文

string **ToString( )**

アクセス

読み込みと書き込み

Value プロパティ

説明

パラメータの値を表します。

構文

object **Value**

アクセス

読み込みと書き込み

実装

IDataParameter.Value

使用法

- 入力パラメータでは、値は、サーバに送信される AseCommand にバインドされます。出力パラメータと戻り値パラメータでは、AseCommand の完了時と AseDataReader をクローズした後に、値が設定されます。
- null パラメータ値をサーバに送信するときは、null ではなく DBNull を指定します。システムでの null 値は、値を持たない空のオブジェクトです。
- アプリケーションでデータベースの型を指定する場合は、Adaptive Server ADO.NET Data Provider がデータをサーバに送信するときに、バインドされた値がその型に変換されます。Adaptive Server ADO.NET Data Provider が IConvertible インタフェースをサポートする場合は、どのような型の値でも変換されます。指定した型が値と互換性を持たないときは、変換エラーが発生する場合があります。
- Value の設定によって、DbType と AseDbType の両方のプロパティを推論できます。
- Value プロパティは Update によって上書きされます。

AseParameterCollection クラス

説明 AseCommand のすべてのパラメータと、オプションとして DataSet カラムへのマッピングを表します。

基本クラス オブジェクト

実装 ICollection、IEnumerable、IDataParameterCollection

使用法 AseParameterCollection には、コンストラクタはありません。AseParameterCollection は AseCommand.Parameters プロパティから取得します。

参照 「Parameters プロパティ」(105 ページ)

Add メソッド

説明 AseParameter を AseCommand に追加します。

構文 1 public AseParameter **Add(** AseParameter *P* **)**

構文 2 public AseParameter **Add(** object *P***)**

構文 3 public AseParameter **Add(** string *name*, AseDbType *dataType* **)**

構文 4 public AseParameter **Add(** string *name*, object *value* **)**

構文 5 public AseParameter **Add(** string *name*, AseDbType *dataType*, AseParameter *size***)**

構文 6 public AseParameter **Add(** string *name*, AseDbType *dataType*, int *size, string sourceColumn***)**

構文 7 public AseParameter **Add(** string *parameterName*, AseDbType *dbType*, int *size*, ParameterDirection *direction*, Boolean *isNullable*, Byte *precision*, Byte *scale*, string *sourceColumn*, DataRowVersion *sourceVersion*, object *value***)**

パラメータ **value**：構文 1 と 2 では、*value* はコレクションに追加する AseParameter オブジェクトです。構文 3 では、*value* は接続に追加するパラメータの値です。

parameterName：パラメータの名前。

aseDbType：AseDbType 値の 1 つ。

size：カラムの長さ。

sourceColumn：ソース・カラムの名前。

戻り値 Add は、AseCommand によって使用されるパラメータ・リストにパラメータを挿入します。戻り値は、リストに追加された新しいパラメータです。

Clear メソッド

説明　コレクションからすべての項目を削除します。

構文　void **Clear( )**

実装　IList.Clear

Contains メソッド

説明　AseParameter がコレクション内にあるかどうかを示す値を返します。

構文 1　bool **Contains(** object *value* **)**

構文 2　bool **Contains(** string *value* **)**

パラメータ　**value**：対象となる AseParameter オブジェクトの値。構文 2 の場合は名前。

戻り値　AseParameter オブジェクトがコレクション内にある場合は “true”、それ以外の場合は “false”。

実装

- 構文 1 では IList.Contains を実装。
- 構文 2 では IDataParameterCollection.Contains を実装。

CopyTo メソッド

説明　AseParameter オブジェクトを AseParameterCollection から、指定した配列にコピーします。

構文　void **CopyTo(** array *array* int *index* **)**

パラメータ　**array**：AseParameter オブジェクトのコピー先となる配列。

index：配列の開始インデックス。

実装　ICollection.CopyTo

Count プロパティ

説明　コレクション内にある AseParameter オブジェクトの数を表します。

構文　int **Count**

アクセス　読み込み専用

実装　ICollection.Count

IndexOf メソッド

説明 コレクション内での AseParameter の位置を取得します。

構文 1 int **IndexOf(** object *value* **)**

構文 2 int **IndexOf(** string *parameterName* **)**

パラメータ **value**：位置を調べる AseParameter オブジェクト。

parameterName：位置を調べる AseParameter オブジェクトの名前。

戻り値 コレクション内での AseParameter の 0 から始まる位置。

実装

- 構文 1 では IList.IndexOf を実装。
- 構文 2 では IDataParameterCollection.IndexOf を実装。

Insert メソッド

説明 コレクション内の指定したインデックスに AseParameter を挿入します。

構文 void **Insert(** int *index* object *value* **)**

パラメータ **index**：パラメータの挿入先となる、コレクション内の 0 から始まるインデックス。

value：コレクションに追加する AseParameter。

実装 IList.Insert

Item プロパティ

説明 指定したインデックスまたは名前を持つ AseParameter を表します。

構文 1 AseParameter **this[** int *index* **]**

構文 2 AseParameter **this[** string *parameterName* **]**

パラメータ **index**：取得するパラメータの 0 から始まるインデックス。

parameterName：取得するパラメータの名前。

プロパティ値 AseParameter

アクセス 読み込みと書き込み

使用法 C# では、このプロパティは AseParameterCollection クラスのインデクサになります。

Remove メソッド

説明　メソッドに渡された AseParameter をコレクションから削除します。

構文

```
void Remove( object value )
```

パラメータ　**value**：コレクションから削除する AseParameter オブジェクト。

実装　IList.Remove

RemoveAt メソッド

説明　パラメータのインデックスまたは名前に基づいて、コレクションからパラメータを削除します。

構文 1

```
void RemoveAt( int index )
```

構文 2

```
void RemoveAt( string parameterName )
```

パラメータ　**index**：削除するパラメータの 0 から始まるインデックス。

parameterName：削除する AseParameter オブジェクトの名前。

実装

- 構文 1 では IList.RemoveAt を実装。
- 構文 2 では IDataParameterCollection.RemoveAt を実装。

AseRowUpdatedEventArgs クラス

説明　RowUpdated イベントのデータを提供します。

基本クラス　RowUpdatedEventArgs

AseRowUpdatedEventArgs コンストラクタ

説明　AseRowUpdatedEventArgs クラスの新しいインスタンスを初期化します。

構文

```
AseRowUpdatedEventArgs( DataRow dataRow, IDbCommand
command, StatementType statementType, DataTableMapping tableMapping )
```

パラメータ　**dataRow**：Update によって送信された DataRow。

command：Update の呼び出し時に実行される IDbCommand。

statementType：実行するクエリの種類を指定する StatementType 値の 1 つ。

tableMapping：Update によって送信された DataTableMapping を表します。

Command プロパティ

説明 Update を呼び出したときに実行される AseCommand を表します。

構文 AseCommand **Command**

アクセス 読み込み専用

Errors プロパティ

説明 コマンドの実行時に、Adaptive Server によって生成されたすべてのエラーを表します。RowUpdatedEventArgs から継承されます。

構文 Exception **Errors**

プロパティ値 コマンドの実行時に、Adaptive Server によって生成されたエラーを表します。

アクセス 読み込みと書き込み

RecordsAffected プロパティ

説明 SQL 文の実行によって変更、挿入、削除されたローの数を表します。RowUpdatedEventArgs から継承されます。

構文 int **RecordsAffected**

プロパティ値 変更、挿入、または削除されたローの数。影響を受けたローがない場合または文が失敗した場合は 0、Select 文の場合は -1 です。

アクセス 読み込み専用

Row プロパティ

説明 Update によって送信された DataRow を表します。RowUpdatedEventArgs から継承されます。

構文 DataRow **Row**

アクセス 読み込み専用

StatementType プロパティ

説明 実行された SQL 文の種類を表します。RowUpdatedEventArgs から継承されます。

構文 StatementType **StatementType**

アクセス 読み込み専用

使用法 StatementType は、Select、Insert、Update、または Delete のいずれかです。

Status プロパティ

説明　Command プロパティの UpdateStatus を表します。RowUpdatedEventArgs から継承されます。

構文

```
UpdateStatus Status
```

プロパティ値　次の UpdateStatus 値のいずれか。Continue (デフォルト)、ErrorsOccurred、SkipAllRemainingRows、または SkipCurrentRow。

アクセス　読み込みと書き込み

TableMapping プロパティ

説明　Update によって送信された DataTableMapping を表します。RowUpdatedEventArgs から継承されます。

構文

```
DataTableMapping TableMapping
```

アクセス　読み込み専用

AseRowUpdatingEventArgs クラス

説明　RowUpdating イベントのデータを提供します。

基本クラス

```
RowUpdatingEventArgs
```

AseRowUpdatingEventArgs コンストラクタ

説明　AseRowUpdatingEventArgs クラスの新しいインスタンスを初期化します。

構文

```
AseRowUpdatingEventArgs( DataRow row, IDbCommand command,
StatementType statementType, DataTableMapping tableMapping )
```

パラメータ

row：更新する DataRow。

command：更新時に実行する IDbCommand。

statementType：実行するクエリの種類を指定する StatementType 値の 1 つ。

tableMapping：Update によって送信された DataTableMapping を表します。

Command プロパティ

説明　Update の実行時に実行する AseCommand を表します。

構文

```
AseCommand Command
```

アクセス　読み込みと書き込み

Errors プロパティ

説明 コマンドの実行時に、Adaptive Server によって生成されたすべてのエラーを表します。RowUpdatingEventArgs から継承されます。

構文 Exception **Errors**

プロパティ値 コマンドの実行時に、Adaptive Server によって生成されたエラーを表します。

アクセス 読み込みと書き込み

Row プロパティ

説明 Update によって送信された DataRow を表します。RowUpdatingEventArgs から継承されます。

構文 DataRow **Row**

アクセス 読み込み専用

StatementType プロパティ

説明 実行された SQL 文の種類を表します。RowUpdatingEventArgs から継承されます。

構文 StatementType **StatementType**

アクセス 読み込み専用

使用法 StatementType は、Select、Insert、Update、または Delete のいずれかです。

Status プロパティ

説明 Command プロパティの UpdateStatus を表します。RowUpdatingEventArgs から継承されます。

構文 UpdateStatus **Status**

プロパティ値 次の UpdateStatus 値のいずれか。Continue (デフォルト)、ErrorsOccurred、SkipAllRemainingRows、または SkipCurrentRow。

アクセス 読み込みと書き込み

TableMapping プロパティ

説明　Update によって送信された DataTableMapping を表します。RowUpdatingEventArgs から継承されます。

構文　DataTableMapping **TableMapping**

アクセス　読み込み専用

AseRowUpdatedEventHandler デリゲート

説明　AseDataAdapter の RowUpdated イベントを処理するメソッドを表します。

構文　void **AseRowUpdatedEventHandler (** object *sender*, AseRowUpdatedEventArgs *e* **)**

パラメータ　**sender :**イベントの発生元。

eイベント・データが格納された AseRowUpdatedEventArgs。

AseRowUpdatingEventHandler デリゲート

説明　AseDataAdapter の RowUpdating イベントを処理するメソッドを表します。

構文　void **AseRowUpdatingEventHandler (** object *sender*, AseRowUpdatingEventArgs *e* **)**

パラメータ　**sender :**イベントの発生元。

eイベント・データが格納された AseRowUpdatingEventArgs。

AseTransaction クラス

説明　SQL トランザクションを表します。

基本クラス　オブジェクト

実装　IDbTransaction

使用法

- AseTransaction には、コンストラクタはありません。AseTransaction オブジェクトを取得するには、AseConnection.BeginTransaction() メソッドを使用します。
- トランザクションにコマンドを関連付けるには、AseCommand.Transaction プロパティを使用します。

参照 「BeginTransaction メソッド」(119 ページ)、「トランザクション処理」(75 ページ)、「AseCommand オブジェクトを使用したローの挿入、更新、削除」(42 ページ)

Commit メソッド

説明 データベース・トランザクションをコミットします。

構文 void **Commit( )**

実装 IDbTransaction.Commit

Connection プロパティ

説明 トランザクションに関連付けられている AseConnection オブジェクトを表します。トランザクションが無効になっている場合は null 参照 (Visual Basic の場合は “Nothing”) です。

構文 AseConnection **Connection**

アクセス 読み込み専用

使用法 1 つのアプリケーションが複数のデータベース接続を保持し、それぞれの接続で 0 または 1 つのトランザクションが実行される場合があります。このプロパティを使用すると、BeginTransaction によって作成された特定のトランザクションに関連付けられている接続オブジェクトを識別できます。

IsolationLevel プロパティ

説明 このトランザクションの独立性レベルを指定します。

構文 IsolationLevel *IsolationLevel*

アクセス 読み込み専用

プロパティ値 このトランザクションの独立性レベル。ReadCommitted (デフォルト)、ReadUncommitted、RepeatableRead、または Serializable を指定できます。

実装 IDbTransaction.IsolationLevel

Rollback メソッド

説明 データベース・トランザクションをロールバックします。

構文 void **Rollback( )**

実装 IDbTransaction.Rollback

使用法 Rollback は同期的です。最初に BeginTransaction() を呼び出してから、返された AseTransaction に対して Rollback を呼び出します。

TraceEnterEventHandler デリゲート

説明　TraceEnter イベントを処理するメソッドを表します。

構文

```
void TraceEnterEventHandler( AseConnection connection,
  Object source, string method, Object[] parameters)
```

パラメータ

connectionイベントが発生した接続。

sourceイベントをトリガしたオブジェクト。

method入力されたメソッド。

parameters入力されたメソッドのパラメータ。

TraceExitEventHandler デリゲート

説明　TraceExit イベントを処理するメソッドを表します。

構文

```
void TraceExitEventHandler( AseConnection connection, Object source, string
method, Object[] returnValue)
```

パラメータ

connectionイベントが発生した接続。

sourceイベントをトリガしたオブジェクト。

method終了したメソッド。

returnValue終了したメソッドの戻り値。

索引

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

あ

い

え

お

か

く

こ

さ

し

す

せ

ち

て

と

に

ね

は

ひ

ふ

め