



InterSystems ソフトウェアでの .NET の使用法

Version 2024.1
2024-06-03

InterSystems ソフトウェアでの .NET の使用法

InterSystems IRIS Data Platform Version 2024.1 2024-06-03

Copyright © 2024 InterSystems Corporation

All rights reserved.

InterSystems®, HealthShare Care Community®, HealthShare Unified Care Record®, IntegratedML®, InterSystems Caché®, InterSystems Ensemble®, InterSystems HealthShare®, InterSystems IRIS®, および TrakCare は、InterSystems Corporation の登録商標です。HealthShare® CMS Solution Pack™ HealthShare® Health Connect Cloud™, InterSystems IRIS for Health™, InterSystems Supply Chain Orchestrator™, および InterSystems TotalView™ For Asset Management は、InterSystems Corporation の商標です。TrakCare は、オーストラリアおよび EU における登録商標です。

ここで使われている他の全てのブランドまたは製品名は、各社および各組織の商標または登録商標です。

このドキュメントは、インターシステムズ社(住所: One Memorial Drive, Cambridge, MA 02142)あるいはその子会社が所有する企業秘密および秘密情報を含んでおり、インターシステムズ社の製品を稼動および維持するためにのみ提供される。この発行物のいかなる部分も他の目的のために使用してはならない。また、インターシステムズ社の書面による事前の同意がない限り、本発行物を、いかなる形式、いかなる手段で、その全てまたは一部を、再発行、複製、開示、送付、検索可能なシステムへの保存、あるいは人またはコンピュータ言語への翻訳はしてはならない。

かかるプログラムと関連ドキュメントについて書かれているインターシステムズ社の標準ライセンス契約に記載されている範囲を除き、ここに記載された本ドキュメントとソフトウェアプログラムの複製、使用、廃棄は禁じられている。インターシステムズ社は、ソフトウェアライセンス契約に記載されている事項以外にかかるソフトウェアプログラムに関する説明と保証をするものではない。さらに、かかるソフトウェアに関する、あるいはかかるソフトウェアの使用から起こるいかなる損失、損害に対するインターシステムズ社の責任は、ソフトウェアライセンス契約にある事項に制限される。

前述は、そのコンピュータソフトウェアの使用およびそれによって起こるインターシステムズ社の責任の範囲、制限に関する一般的な概略である。完全な参照情報は、インターシステムズ社の標準ライセンス契約に記載され、そのコピーは要望によって入手することができる。

インターシステムズ社は、本ドキュメントにある誤りに対する責任を放棄する。また、インターシステムズ社は、独自の裁量にて事前通知なしに、本ドキュメントに記載された製品および実行に対する代替と変更を行う権利を有する。

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください:

InterSystems Worldwide Response Center (WRC)

Tel: +1-617-621-0700

Tel: +44 (0) 844 854 2917

Email: support@InterSystems.com

目次

1 InterSystems ソフトウェアでの .NET の使用法の概要	1
2 .NET を使用したデータベースへの接続	3
2.1 .NET との接続の確立	3
2.1.1 .NET クライアント接続のトラブルシューティング	3
2.2 .NET 共有メモリ接続	4
2.3 .NET での接続プーリング	4
3 .NET の要件と構成	7
3.1 IRISClient アセンブリの Managed Provider	7
3.2 サポート対象の .NET バージョン	7
3.2.1 サポートされていない IRISClient アセンブリ	7
3.3 IRISClient アセンブリの構成	8
3.3.1 要件	8
3.3.2 IRISClient アセンブリのセットアップ	8
3.3.3 .NET アプリケーションでの IRISClient の宣言	9
3.3.4 Visual Studio の構成	9
3.4 Entity Framework Provider の設定	9
3.4.1 システム要件	9
3.4.2 IrisEF ディレクトリの作成	10
3.4.3 Visual Studio の構成と EF プロバイダのインストール	10
3.4.4 Visual Studio へのファイルのコピー	10
3.4.5 サーバへの Visual Studio の接続	10
3.4.6 NuGet ローカル・リポジトリの構成	11
4 ADO.NET Managed Provider の使用法	13
4.1 ADO.NET Managed Provider クラスの概要	14
4.2 IRISCommand と IRISDataReader の使用法	15
4.3 IRISParameter を使用した SQL クエリの使用法	15
4.4 IRISDataAdapter と IRISCommandBuilder の使用法	16
4.5 トランザクションの使用法	17
5 .NET Managed Provider のクイック・リファレンス	19
5.1 IRISPoolManager クラス	19
5.2 IRISConnection クラス	20
5.3 接続パラメータのオプション	20
5.3.1 必須パラメータ	20
5.3.2 接続プーリング・パラメータ	21
5.3.3 その他の接続パラメータ	22

1

InterSystems ソフトウェアでの .NET の使用法の概要

このドキュメントで取り上げる内容の詳細なリストは、“[目次](#)”を参照してください。

InterSystems IRIS® には、広範囲にわたる堅牢な .NET 接続オプションが用意されています。これには、.NET ADO、.NET オブジェクト、または InterSystems 多次元ストレージを使用したデータベース・アクセスを提供する軽量の SDK、および InterSystems IRIS サーバ・アプリケーションから .NET アプリケーションおよび外部データベースに直接アクセスできるようにするゲートウェイが含まれます。

このドキュメントでは、以下の項目について説明します。

- ・ [.NET を使用したデータベースへの接続](#) – データベース接続 (接続プーリングを含む) の詳細について説明します。
- ・ [.NET の要件と構成](#) – すべての InterSystems .NET ソリューションの設定と構成に関する情報を提供します。
- ・ [ADO.NET Managed Provider の使用法](#) – ADO.NET Managed Provider API のインターシステムズ実装を使用した具体例を示します。
- ・ [.NET Managed Provider のクイック・リファレンス](#) – これらのトピックで取り上げるすべてのメソッドとプロパティをリストして説明します。

関連ドキュメント

以下のドキュメントには、InterSystems IRIS が提供するその他の .NET ソリューションに関する詳細が記載されています。

- ・ “[Native SDK for .NET の使用法](#)” では、以前は ObjectScript を介してのみ使用可能であったリソースに、.NET Native SDK を使用してアクセスする方法について説明しています。
- ・ “[InterSystems XEP による .NET オブジェクトの永続化](#)” では、Event Persistence SDK (XEP) を使用して .NET オブジェクトの永続性を迅速に実現する方法を説明しています。
- ・ “[InterSystems ODBC ドライバの使用法](#)” では、ODBC ドライバを使用して外部アプリケーションからインターシステムズ・データベースにアクセスする方法とインターシステムズ製品から外部 ODBC データ・ソースにアクセスする方法について説明します。

2

.NET を使用したデータベースへの接続

ここでは、**IRISConnection** オブジェクトを使用して、.NET クライアント・アプリケーションとインターシステムズ・サーバ間の接続を作成する方法を説明します。

2.1 .NET との接続の確立

以下に示すコードでは、**USER** というネームスペースへの接続を確立します。接続オブジェクトをインスタンス化する場合に設定できるパラメータの完全なリストについては、“[接続パラメータのオプション](#)”を参照してください。

以下の簡単なメソッドを呼び出して接続を開始できます。

接続をインスタンス化するコードの追加

```
public IRISConnection Conn;
private void CreateConnection(){
    try {
        Conn = new IRISConnection();
        Conn.ConnectionString =
            "Server=localhost; Port=51773; Namespace=USER;"
            + "Password=SYS; User ID=_SYSTEM;";
        Conn.Open();
    }
    catch (Exception eConn){
        MessageBox.Show("CreateConnection error: " + eConn.Message);
    }
}
```

このオブジェクトを作成しておくと、そのオブジェクトを必要とするすべてのクラス間で共有できます。接続オブジェクトは、必要に応じて開いたり閉じたりできます。この操作は、Conn.Open() および Conn.Close() を使用して明示的に実行できます。ADO.NET の **Dataset** を使用している場合は、**DataAdapter** のインスタンスにより、必要に応じて自動的に接続が開閉します。

2.1.1 .NET クライアント接続のトラブルシューティング

インターシステムズ・サーバに接続しようとする .NET クライアントには、以下の情報が必要です。

- ・ サーバ IP アドレス、ポート番号、およびネームスペースを指定する URL。
- ・ 接続でパスワードを使用する場合は、大文字と小文字を区別するユーザ名とパスワードを指定する必要があります。

問題が発生した場合は、以下の点を確認してください。

- ・ サーバ・プロセスがインストールされていて実行中であることを確認します。

- ・ サーバ・プロセスを実行しているマシンの IP アドレスを確認します。
- ・ 待ち受け状態になっているサーバの TCP/IP ポート番号を確認します。
- ・ 接続を確立する際に、有効なユーザ名とパスワードを使用していることを確認します(ユーザ名とパスワードは、管理ポータル (System Administration > Security > Users) を使用して管理できます)。
- ・ 接続 URL に有効なネームスペースが記述されていることを確認します。ネームスペースは、プログラムで使用するクラスとデータを含む必要があります。

2.2 .NET 共有メモリ接続

リモートの InterSystems IRIS インスタンスへの標準の ADO .NET 接続では TCP/IP が使用されます。パフォーマンスを最大限に高めるために、InterSystems IRIS は、InterSystems IRIS インスタンスと同じマシン上で実行される .NET アプリケーション向けに共有メモリ接続も提供します。この接続は、高いコストのかかる可能性があるカーネル・ネットワーク・スタックの呼び出しを回避して、.NET 操作のために最大限の低遅延と高スループットを実現します。

接続でサーバ・アドレス localhost または 127.0.0.1 が指定されている場合、既定で共有メモリが使用されます。実際のマシン・アドレスが指定されている場合は、TCP/IP が使用されます。共有メモリ・デバイスに障害が発生した場合、または共有メモリ・デバイスが利用できない場合、接続は自動的に TCP/IP にフォールバックします。

共有メモリを無効にするには、接続文字列で SharedMemory プロパティを false に設定します。例えば、サーバ・アドレスが localhost に指定されている場合でも、以下の接続文字列は共有メモリを使用しません。

```
"Server=localhost;Port=51774;Namespace=user;Password = SYS;User ID = _system;SharedMemory=false"
```

共有メモリは TLS 接続には使用されません。共有メモリ接続が試行されたかどうか、およびその接続が成功したかどうかの情報は、ログに記録されます。

注釈 共有メモリ接続はコンテナの境界を越えて機能しない

現在のところ、2 つの異なるコンテナ間の共有メモリ接続はサポートされていません。クライアントが localhost または 127.0.0.1 を使用してコンテナの境界を越えて接続を試行した場合、接続モードは既定で共有メモリになり、接続は失敗します。このことは、Docker の `--network host` オプションが指定されているかどうかに関係なく適用されます。サーバ・アドレスの実際のホスト名を指定するか、接続文字列で共有メモリを無効にすることで (上記の例のように)、コンテナ間の TCP/IP 接続を保証できます。

サーバとクライアントが同じコンテナにある場合は、問題なく共有メモリ接続を使用できます。

2.3 .NET での接続プーリング

接続プーリングは既定で有効になっています。以下の接続文字列パラメータを使用すると、接続プーリングのさまざまな機能を制御できます。

- ・ Pooling – 既定は true です。接続プーリングを使用しない接続を作成するには、Pooling を false に設定します。
- ・ Min Pool Size および Max Pool Size – 既定値は 0 および 100 です。これらのパラメータを設定して、この特定接続文字列で接続プールの最大サイズと最小 (初期) サイズを指定します。

- ・ Connection Reset および Connection Lifetime – Connection Reset を true に設定すると、プールされた接続のリセット・メカニズムが有効になります。Connection Lifetime は、アイドル状態のプールされた接続をリセットするまで待機する秒数を指定します。既定値は 0 です。

例えば以下の接続文字列は、接続プールの初期サイズを 2、接続の最大数を 5 に設定し、最大接続アイドル時間を 3 秒にして接続のリセットを有効にします。

```
Conn.ConnectionString =  
    "Server = localhost;"  
    + " Port = 51774;"  
    + " Namespace = USER;"  
    + " Password = SYS;"  
    + " User ID = _SYSTEM;"  
    + " Min Pool Size = 2;"  
    + " Max Pool Size = 5;"  
    + " Connection Reset = true;"  
    + " Connection Lifetime = 3;"
```

コネクション・プールのさまざまなメソッドとプロパティの詳細は、“[.NET Managed Provider のクイック・リファレンス](#)”を参照してください。

3

.NET の要件と構成

ここでは、インターシステムズの .NET クライアント・アセンブリに関する情報を提供します。[Entity Framework Provider の設定手順](#)についても説明します。

3.1 IRISClient アセンブリの Managed Provider

IRISClient .NET アセンブリは、.NET アプリケーションからインターシステムズ・データベースへのアクセスをサポートする 2 つの補完的なプロバイダを提供しています。

- ・ ADO.NET Managed Provider – インターシステムズにおける ADO.NET データ・アクセス・インタフェースの実装です。標準 ADO.NET Managed Provider クラスを使用した、データへの簡単なリレーショナル・アクセスを提供します（“[ADO.NET Managed Provider クラスの使用法](#)”を参照）。
- ・ Entity Framework Provider – インターシステムズにおける ADO.NET 用オブジェクト・リレーショナル・マッピング (ORM) フレームワークの実装です。これにより、.NET 開発者はドメイン固有のオブジェクトを使用してリレーショナル・データを処理できます（“[Entity Framework Provider の設定](#)”を参照）。

詳細は、“[Using Entity Framework with InterSystems IRIS Data Platform](#)” のコースを参照してください。

IRISClient アセンブリは、全面的に .NET マネージド・コードを使用して実装されるため、.NET 環境での配置が容易です。スレッドセーフであるため、マルチスレッドの .NET アプリケーションで使用できます。

3.2 サポート対象の .NET バージョン

サポート対象の .NET バージョンのリストは、“インターシステムズのサポート対象プラットフォーム”の“サポート対象 .NET Framework”を参照してください。

3.2.1 サポートされていない IRISClient アセンブリ

InterSystems IRIS では、Microsoft でサポートされなくなったいくつかのバージョンの .NET のサポートを終了しています (.NET Framework 2.0、4.0、4.5、および .NET Core 1.0 と 2.1)。dll の場所へのパスを使用する以前のプロジェクトでは、新しいアセンブリ・ファイルの場所に対応するようパスを更新する必要があります。例えば、以前のパスが以下の場合：

```
\<IRIS install location>%dev%dotnet%bin%v4.5%InterSystems.Data.IRISClient.dll
```

その場所は、新しいインストールの下には存在しなくなっているため、以下のように変更する必要があります。

<IRIS install location>%dev%dotnet%bin%v4.6.2%InterSystems.Data.IRISClient.dll

バージョン間の互換性については、新しい 4.6.2 バージョンには下位互換性があり、いずれかの .NET Framework 4.x がインストールされているシステムであれば、アプリケーションは動作します。

ただし、.NET Framework には上位互換性がないため、アプリケーションで特に .NET Framework 4.5 をターゲットとしている場合、.NET Framework 4.6.2 クライアント・ライブラリを依存関係として使用することはできません。この場合の選択肢は以下ようになります。

- ・ アプリケーションのターゲット・フレームワークを 4.6.2 以上に変更する。.NET Framework 4.5 は 2016 年以降、Microsoft によるサポートが終了しているため、この変更によって、確実にサポートされている言語バージョンを使用することにもなります。
- ・ .NET Framework 3.5 バージョンのライブラリを使用する。バージョン 4.0 で導入されている一部の機能にアクセスできなくなる可能性があります。
- ・ 4.5 をターゲットとする古いバージョンのクライアント・ライブラリを使い続ける。古いバージョンには、最新のバグ修正や機能が追加されることはありませんが、アプリケーションの依存関係を変更する必要はありません。InterSystems IRIS サーバが今後のバージョンにアップグレードされると、最終的には古いクライアントとの互換性がなくなるため、これは、一時的な解決策です。

3.3 IRISClient アセンブリの構成

サポートが、全面的に .NET マネージド・コードを使用して **IRISClient** アセンブリに実装されるため、.NET 環境への導入が容易になります。**IRISClient** はスレッドセーフであるため、マルチスレッドの .NET アプリケーションで使用できます。ここでは、**IRISClient** アセンブリのインストールと Visual Studio の構成に関する要件と手順を示します。

3.3.1 要件

- ・ サポート対象バージョンの .NET または .NET Framework
- ・ Visual Studio 2013 以降

.NET クライアント・アプリケーションを実行するコンピュータには InterSystems IRIS は不要です。ただし、クライアントでは、インターシステムズ・サーバへの TCP/IP 接続が可能であること、およびサポート対象バージョンの .NET または .NET Framework を実行していることが必要です。

3.3.2 IRISClient アセンブリのセットアップ

IRISClient アセンブリ (**InterSystems.Data.IRISClient.dll**) は、InterSystems IRIS の他のコンポーネントと共にインストールされます。特別な準備は必要ありません。

- ・ Windows では、InterSystems IRIS をインストールするときに **Setup Type: Development** オプションを選択します。
- ・ InterSystems IRIS をセキュリティ・オプション 2 でインストールした場合、管理ポータルを開いて、**System Administration** > **Security** > **Services** に移動し、%Service_CallIn を選択して、[] ボックスにチェックが付いていることを確認します。InterSystems IRIS をセキュリティ・オプション 1 (最小) でインストールした場合、これには既にチェックが付いているはずです。

重要

クラウド・サービス・インストールの設定

InterSystems IRIS をローカル・インストールで実行していない場合、クライアント・ドライバを手動でダウンロードしてインストールすることが必要な場合があります。このオプションの詳細は、“[Connecting Your Application to InterSystems IRIS](#)” を参照してください。

3.3.3 .NET アプリケーションでの IRISClient の宣言

.NET アプリケーションで **IRISClient** アセンブリを使用するには、**InterSystems.Data.IRISClient.dll** アセンブリの **Using** 文をアプリケーションのネームスペースの先頭に追加する必要があります。

using 文とそれに続くネームスペースの両方が必要です。

```
using InterSystems.Data.IRISClient;
namespace YourNameSpace {
    ...
}
```

サポート対象の .NET および .NET Framework のバージョンごとに、個別の **InterSystems.Data.IRISClient.dll** のバージョンがあります。詳細は、“サポート対象 .NET Framework” を参照してください。

3.3.4 Visual Studio の構成

プロジェクトに **IRISClient** アセンブリ参照を追加するには、以下の手順を実行します。

1. Visual Studio のメイン・メニューで、[] [] を選択します。
2. [] ウィンドウで [] をクリックします。
3. プロジェクトで使用している .NET のバージョンのアセンブリを含む <iris-install-dir>%dev%dotnet%bin のサブディレクトリ (前のセクションにリストされています) を探して、**InterSystems.Data.IRISClient.dll** を選択してから [OK] をクリックします。
4. Visual Studio のソリューション・エクスプローラでは、[参照] の下に **InterSystems.Data.IRISClient** アセンブリが表示されます。

3.4 Entity Framework Provider の設定

Entity Framework は、.NET 開発者がドメイン固有オブジェクトを使用してリレーショナル・データを処理できるようにするオブジェクト・リレーショナル・マップです。InterSystems Entity Framework Provider では、Entity Framework 6 テクノロジを使用してインターシステムズ・データベースにアクセスできます。Entity Framework を使用した実施例を含むチュートリアルについては、“[Using Entity Framework with InterSystems IRIS Data Platform](#)” を参照してください。

InterSystems Entity Framework Provider を構成するには、このセクションの手順を実行します。

3.4.1 システム要件

InterSystems IRIS で Entity Framework Provider を使用するには、以下のソフトウェアが必要になります。

- ・ Visual Studio 2013 以降 (最初のサポート対象リリースは、Update 5 が適用された VS 2013 Professional/Ultimate)。
- ・ サポート対象バージョンの .NET Framework (.NET Core 以降の .NET バージョンはサポートされていません)。
- ・ InterSystems IRIS Entity Framework Provider のディストリビューション (次のセクションで説明します)。

3.4.2 IrisEF ディレクトリの作成

InterSystems IRIS Entity Framework Provider のディストリビューション・ファイルは、install-dir¥dev¥dotnet¥bin¥v4.0.30309 にある IrisEF.zip です。

1. install-dir¥dev¥dotnet¥bin¥v4.0.30309¥IrisEF という名前の新しいディレクトリを作成します。
2. IrisEF.zip のコンテンツを新しいディレクトリに抽出します。

この .zip ファイルには、設定手順で使用する以下のファイルが含まれています。

- ・ setup.cmd – DLL InterSystems.Data.IRISClient.dll および InterSystems.Data.IRISVSTools.dll をインストールします。
- ・ Nuget¥InterSystems.Data.Entity6.4.5.0.0.nupkg – Entity Framework Provider をインストールします。
- ・ CreateNorthwindEFDB.sql – サンプル・データベースの作成に使用されます (詳細は、[“Using Entity Framework with InterSystems IRIS Data Platform”](#) を参照してください)。

3.4.3 Visual Studio の構成と EF プロバイダのインストール

重要 VS 2013 または 2015 を実行している場合は、以下の手順 2 と 3 を逆にし、まず setup.cmd を実行してから、devenv /setup を実行します。

1. [新しい IrisEF ディレクトリ](#)に移動します。次の手順では、IrisEF を現在のディレクトリと想定しています。
2. Visual Studio 開発環境を設定します。
 - ・ Windows で、**すべてのプログラム > Visual Studio 201x > Visual Studio Tools** を選択します。
 - ・ 表示された Windows エクスプローラ・フォルダで **Developer Command Prompt for VS201x > 管理者として実行** を右クリックして、以下を入力します。

```
devenv /setup
```

このコマンドは、お使いのバージョンの Visual Studio へのパスを指定するレジストリ・キーから環境設定を再生成します。

3. コマンド・プロンプトで、setup.cmd を実行します。これにより、InterSystems Entity Framework Provider ファイル、InterSystems.Data.IRISClient.dll と InterSystems.Data.IRISVSTools.dll がインストールされます。

3.4.4 Visual Studio へのファイルのコピー

[IrisEF サブディレクトリ](#) IrisEF¥Templates から Visual Studio に以下のファイルをコピーします。

- ・ SSDLToIrisSQL.tt
- ・ GenerateIrisSQL.Utility.ttinclude

<iris-install-dir>¥dev¥dotnet¥bin¥v4.0.30319¥IrisEF¥Templates から

<VisualStudio-install-dir>¥Common7¥IDE¥Extensions¥Microsoft¥Entity Framework Tools¥DBGen にコピーします。

3.4.5 サーバへの Visual Studio の接続

インターシステムズ・データベース・インスタンスに Visual Studio を接続するには、以下の手順を実行します。

1. Visual Studio を開いて、**表示 > サーバー エクスプローラー** を選択します。
2. **[データ接続]** を右クリックして、**[接続の追加]** を選択します。[接続の追加] ダイアログで、以下の手順を実行します。
 - a. **[データ・ソース]** で **[InterSystems IRIS Data Source (.Net Framework Data Provider for InterSystems IRIS)]** を選択します。
 - b. **[サーバー]** を選択します。
 - c. **[ユーザー名]** および **[パスワード]** を入力します。**[接続]** をクリックします。
 - d. リストからネームスペースを選択します。**[OK]** をクリックします。

3.4.6 NuGet ローカル・リポジトリの構成

パッケージ・マネージャを構成し、ローカル の NuGet リポジトリを見つけ出すには、以下の手順を実行します。

1. まだディレクトリを NuGet リポジトリとして作成していない場合は、これを作成します。任意の名前と場所を使用できます。例えば、Visual Studio の既定のプロジェクト・ディレクトリにディレクトリ **NuGet Repository** を作成できます (<yourdoclibraryVS201x>\Projects)。
2. **IrisEF サブディレクトリ** **IrisEF\Nuget** から **InterSystems.Data.Entity6.4.5.0.0.nupkg** ファイルを NuGet リポジトリ・ディレクトリにコピーします。**[OK]** をクリックします。
3. Visual Studio で、**プロジェクト > NuGet パッケージの管理 > 設定 > パッケージ マネージャー > パッケージ ソース** を選択します。
4. プラス記号 (+) をクリックします。**InterSystems.Data.Entity6.4.5.0.0.nupkg** を含むパスを入力します。**[OK]** をクリックします。

4

ADO.NET Managed Provider の使用法

ADO.NET では経験豊富な .NET データベース開発者向けに概要を説明する必要はありませんが、小型のユーティリティ・アプリケーションでたまたま .NET を使用するだけでも、この説明は役立つ可能性があります。このセクションでは、ADO.NET の概要として、簡単なデータベース・クエリを実行し、その結果を処理する方法を紹介します。

InterSystems ADO.NET Managed Provider を使用すると、**Connection**、**Command**、**CommandBuilder**、**DataReader**、**DataAdapter** などの汎用 ADO.NET Managed Provider クラスの完全互換バージョンで、.NET プロジェクトからインターシステムズ・データベースにアクセスできます。.NET アプリケーションを InterSystems IRIS に接続する方法の詳細は、“[Connecting Your Application to InterSystems IRIS](#)”を参照してください。以下のクラスは、標準 ADO.NET Managed Provider クラスのインターシステムズ固有の実装です。

- ・ **IRISConnection** – 指定したインターシステムズ・ネームスペースのデータベースとアプリケーションとの間の接続を表します。**IRISConnection** の使用方法の詳細は、“[インターシステムズ・データベースへの接続](#)”を参照してください。
- ・ **IRISCommand** – **IRISConnection** で指定したネームスペース内のデータベースに対して実行する SQL 文またはストアド・プロシージャをカプセル化します。
- ・ **IRISCommandBuilder** – 単一テーブルのクエリをカプセル化するオブジェクトによって加えられた変更に合わせてデータベースを調整する SQL コマンドを自動で生成します。
- ・ **IRISDataReader** – **IRISCommand** で指定した結果セットをフェッチする方法を提供します。**IRISDataReader** オブジェクトは、結果セットに対する前方向のみの迅速なアクセスを提供します。ただし、ランダム・アクセスには対応していません。
- ・ **IRISDataAdapter** – **IRISConnection** で指定したネームスペース内のデータにマッピングされる結果セットをカプセル化します。これは、ADO.NET **DataSet** への入力やデータベースの更新に使用され、結果セットへの効率的なランダム・アクセス接続を実現します。

この章では、InterSystems ADO.NET Managed Provider クラスを使用したコードの具体例を示します。ここで説明する内容は以下のとおりです。

- ・ [ADO.NET Managed Provider クラスの概要](#) – InterSystems ADO.NET Managed Provider クラスの使用法について簡単に説明します。
- ・ [IRISCommand と IRISDataReader の使用法](#) – 読み取り専用の簡単なクエリの実行方法について説明します。
- ・ [IRISParameter を使用した SQL クエリの使用法](#) – クエリにパラメータを渡す方法について説明します。
- ・ [IRISDataAdapter と IRISCommandBuilder の使用法](#) – データの変更と更新を行います。
- ・ [トランザクションの使用法](#) – トランザクションをコミットまたはロールバックする方法について説明します。

4.1 ADO.NET Managed Provider クラスの概要

ADO.NET Managed Provider クラスのインターシステムズ実装を使用するプロジェクトは非常にシンプルです。Sample.Person データベースを開いてその項目を読み取る、完成された実用的なコンソール・プログラムを以下に示します。

```
using System;
using InterSystems.Data.IRISClient;

namespace TinySpace {
    class TinyProvider {
        [STAThread]
        static void Main(string[] args) {

            string connectionString = "Server = localhost; Port = 51783; " +
                "Namespace = USER; Password = SYS; User ID = _SYSTEM;";
            using IRISConnection conn = new IRISConnection(connectionString);
            conn.Open();

            using IRISCommand command = conn.CreateCommand();
            command.CommandText = "SELECT * FROM Sample.Person WHERE ID = 1";
            IRISDataReader reader = command.ExecuteReader();
            while (reader.Read()) {
                Console.WriteLine($"TinyProvider output:\r\n " +
                    $"{reader[reader.GetOrdinal("ID")]}: {reader[reader.GetOrdinal("Name")]");
            }
            reader.Close();
        } // end Main()
    } // end class TinyProvider
}
```

このプロジェクトには、以下の重要な機能があります。

- `using` 文を使用すると、**IRISClient** アセンブリにアクセスできます。クライアント・コード用にネームスペースを宣言する必要があります。

```
using InterSystems.Data.IRISClient;

namespace TinySpace {
```

- **IRISConnection** `conn` オブジェクトは、**USER** ネームスペースへの接続を作成して開くのに使います。`conn` オブジェクトは、常に適切に閉じられ、破棄されるように、`using` 宣言を使用して作成されます。

```
string connectionString = "Server = localhost; Port = 51783; " +
    "Namespace = USER; Password = SYS; User ID = _SYSTEM;";
using IRISConnection conn = new IRISConnection(connectionString);
conn.Open();
```

- **IRISCommand** `command` オブジェクトは、**IRISConnection** オブジェクトと SQL 文を使用して、ID が 1 の `Sample.Person` のインスタンスを開きます。

```
using IRISCommand command = conn.CreateCommand();
command.CommandText = "SELECT * FROM Sample.Person WHERE ID = 1";
```

- **IRISDataReader** オブジェクトは、以下のように行のデータ項目へのアクセスに使います。

```
IRISDataReader reader = command.ExecuteReader();
while (reader.Read()) {
    Console.WriteLine($"TinyProvider output:\r\n " +
        $"{reader[reader.GetOrdinal("ID")]}: {reader[reader.GetOrdinal("Name")]");
}
reader.Close();
```

4.2 IRISCommand と IRISDataReader の使用法

読み取り専用の単純なクエリであれば、**IRISCommand** および **IRISDataReader** を使用して実行できます。すべてのデータベース・トランザクションと同様、このようなクエリの場合でも **IRISConnection** オブジェクトを開いておく必要があります。

この例では、既存の接続を使用する新しい **IRISCommand** オブジェクトに SQL クエリ文字列が渡されます。

```
string SQLtext = "SELECT * FROM Sample.Person WHERE ID < 10";
IRISCommand Command = new IRISCommand(SQLtext, Conn);
```

クエリの結果は、**IRISDataReader** オブジェクトに返されます。SQL 文で指定した列名を参照することで、プロパティにアクセスできます。

```
IRISDataReader reader = Command.ExecuteReader();
while (reader.Read()) {
    Console.WriteLine(
        reader[reader.GetOrdinal("ID")] + "\t"
        + reader[reader.GetOrdinal("Name")] + "\r\n\t"
        + reader[reader.GetOrdinal("Home_City")] + " "
        + reader[reader.GetOrdinal("Home_State")] + "\r\n");
};
```

列名ではなく列番号を使用した場合も、同じレポートを生成できます。**IRISDataReader** オブジェクトでは、前方への読み取りのみが可能です。データ・ストリームの先頭に戻るには、この読み取り操作をいったん終了し、クエリを再実行して読み取り操作をもう一度開始する必要があります。

```
reader.Close();
reader = Command.ExecuteReader();
while (reader.Read()) {
    Console.WriteLine(
        reader[0] + "\t"
        + reader[4] + "\r\n\t"
        + reader[7] + " "
        + reader[8] + "\n");
}
```

4.3 IRISParameter を使用した SQL クエリの使用法

IRISParameter オブジェクトは、より複雑な SQL クエリに必要です。以下の例では、**IRISParameter** で指定した文字列から始まる値を Name に持つすべての行からデータを選択しています。

```
string SQLtext =
    "SELECT ID, Name, DOB, SSN "
    + "FROM Sample.Person "
    + "WHERE Name %STARTSWITH ?"
    + "ORDER BY Name";
IRISCommand Command = new IRISCommand(SQLtext, Conn);
```

このパラメータ値は、Name の先頭の文字列が A である行をすべて取得するように設定されています。このパラメータは **IRISCommand** オブジェクトに渡されます。

```
IRISParameter Name_param =
    new IRISParameter("Name_col", IRISDbType.NVarChar);
Name_param.Value = "A";
Command.Parameters.Add(Name_param);
```

注釈 既定では、SQL 文はサーバ上で実行される前に検証されません。これは、クエリごとにサーバを 2 回呼び出す必要があるためです。検証が必要な場合は、**IRISCommand.Prepare()** を呼び出して、サーバに対する SQL 文の構文を検証します。

IRISDataReader オブジェクトでは、前述の例と同様に結果のデータ・ストリームにアクセスできます。

```
IRISDataReader reader = Command.ExecuteReader();
while (reader.Read()) {
    Console.WriteLine(
        reader[reader.GetOrdinal("ID")] + "\t"
        + reader[reader.GetOrdinal("Name")] + "\r\n\t"
        + reader[reader.GetOrdinal("DOB")] + " "
        + reader[reader.GetOrdinal("SSN")] + "\r\n");
};
```

4.4 IRISDataAdapter と IRISCommandBuilder の使用法

連続的な読み取り専用アクセス以外のアクセスをアプリケーションで行う必要がある場合は、**IRISCommand** および **IRISDataReader** クラスでは不十分です。このような場合は、**IRISDataAdapter** および **IRISCommandBuilder** クラスによって完全なランダム読み取り/書き込みアクセスを提供できます。以下に示す例では、これらのクラスを使用して、一連の **Sample.Person** 行を取得し、1 つの行を読み取って変更し、行を削除して新しい行を追加し、変更内容をデータベースに保存します。

IRISDataAdapter コンストラクタは、**IRISCommand** と同様に、SQL コマンドと **IRISConnection** オブジェクトをパラメータとして受け取ります。この例では、**Name** が **A** または **B** で始まるすべての **Sample.Person** 行のデータが結果セットに含まれます。**Adapter** オブジェクトは、この結果セットを **Person** という名前のテーブルにマッピングします。

```
string SQLtext =
    " SELECT ID, Name, SSN "
    + " FROM Sample.Person "
    + " WHERE Name < 'C' "
    + " ORDER BY Name ";
IRISDataAdapter Adapter = new IRISDataAdapter(SQLtext, Conn);
Adapter.TableMappings.Add("Table", "Person");
```

Adapter オブジェクトに対して **IRISCommandBuilder** オブジェクトを作成します。**Adapter** オブジェクトによってマッピングされたデータが変更されると、**Adapter** は **Builder** によって生成された SQL 文を使用して、データベース内の対応する項目を更新します。

```
IRISCommandBuilder Builder = new IRISCommandBuilder(Adapter);
```

ADO の **DataSet** オブジェクトを作成し、**Adapter** を使ってデータを格納します。このオブジェクトには、**PersonTable** オブジェクトを定義するために使用されるテーブルが 1 つだけ含まれています。

```
System.Data.DataSet DataSet = new System.Data.DataSet();
Adapter.Fill(DataSet);
System.Data.DataTable PersonTable = DataSet.Tables["Person"];
```

簡単な **foreach** コマンドを使用して、**PersonTable** 内の各行を読み取ることができます。この例では、1 行目の **Name** を保存し、それを **"Fudd, Elmer"** に変更します。データを出力すると、すべての名前がアルファベット順に並んでいますが、1 行目だけは **F** で始まります。データを出力した後で、1 行目の **Name** を元の値にリセットします。どちらの変更も、**DataSet** 内のデータに対してのみ行われています。データベース内の元のデータはまだ変更されていません。

```
if (PersonTable.Rows.Count > 0) {
    System.Data.DataRow FirstPerson = PersonTable.Rows[0];
    string OldName = FirstPerson["Name"].ToString();
    FirstPerson["Name"] = "Fudd, Elmer";

    foreach (System.Data.DataRow PersonRow in PersonTable.Rows) {
        Console.WriteLine("\t"
            + PersonRow["ID"] + ":\t"
            + PersonRow["Name"] + "\t"
            + PersonRow["SSN"]);
    }
    FirstPerson["Name"] = OldName;
}
```

次のコードでは、1 行目を削除するようマーク付けし、次に新しい行を作成して追加します。ここでも、これらの変更は DataSet オブジェクトに対してのみ行われています。

```
FirstPerson.Delete();

System.Data.DataRow NewPerson = PersonTable.NewRow();
NewPerson["Name"] = "Budd, Billy";
NewPerson["SSN"] = "555-65-4321";
PersonTable.Rows.Add(NewPerson);
```

最後に、Update() メソッドを呼び出します。Adapter は、今度は **IRISCommandBuilder** のコードを使用して、DataSet オブジェクトの Person テーブルに含まれる現在のデータでデータベースを更新します。

```
Adapter.Update(DataSet, "Person");
```

4.5 トランザクションの使用法

Transaction クラスは、SQL トランザクションの指定に使用します (トランザクションを使用する方法の概要は、“InterSystems SQL の使用法” の “トランザクション処理” を参照してください)。以下の例では、SSN が一意でない場合、トランザクション Trans が失敗して、ロールバックされます。

```
IRISTransaction Trans =
    Conn.BeginTransaction(System.Data.IsolationLevel.ReadCommitted);
try {
    string SQLtext = "INSERT into Sample.Person(Name, SSN) Values(?,?)";
    IRISCommand Command = new IRISCommand(SQLtext, Conn, Trans);

    IRISParameter Name_param =
        new IRISParameter("name", IRISDbType.NVarChar);
    Name_param.Value = "Rowe, Richard";
    Command.Parameters.Add(Name_param);

    IRISParameter SSN_param =
        new IRISParameter("ssn", IRISDbType.NVarChar);
    SSN_param.Value = "234-56-3454";
    Command.Parameters.Add(SSN_param);

    int rows = Command.ExecuteNonQuery();
    Trans.Commit();
    Console.WriteLine("Added record for " + SSN_param.Value.ToString());
}
catch (Exception eInsert) {
    Trans.Rollback();
    WriteErrorMessage("TransFail", eInsert);
}
```


5

.NET Managed Provider のクイック・リファレンス

この章は、以下の拡張クラスとオプションに関するクイック・リファレンスです。

- ・ [IRISPoolManager クラス](#) – InterSystems 接続プーリングに関連するメソッド。
- ・ [IRISConnection クラス](#) – 接続プールをクリアするためのメソッド。
- ・ [接続パラメータ・オプション](#) – サポートされるすべての接続パラメータをリストします。

5.1 IRISPoolManager クラス

`IRISClient.IRISPoolManager` クラスを使用すると、接続プーリングをプログラムで監視して制御できます。使用可能なステティック・メソッドは以下のとおりです。

`ActiveConnectionCount()`

```
int count = IRISPoolManager.ActiveConnectionCount();
```

すべてのプールにおける確立されている接続の合計数。合計数には、アイドル状態および使用中の両方の接続が含まれます。

`IdleCount()`

```
int count = IRISPoolManager.IdleCount();
```

すべてのプールにおけるアイドル状態の接続の合計数。

```
int count = IRISPoolManager.IdleCount(conn);
```

接続オブジェクト `conn` に関連付けられたプールのアイドル状態接続の合計数。

`InUseCount()`

```
int count = IRISPoolManager.InUseCount();
```

すべてのプールにおける使用中の接続の合計数。

```
int count = IRISPoolManager.InUseCount(conn);
```

接続オブジェクト `conn` に関連付けられたプールの使用中接続の合計数。

RecycleAllConnections()

```
IRISPoolManager.RecycleAllConnections(bool remove);
```

すべてのプールにおける接続をリサイクルします。

```
IRISPoolManager.RecycleConnections(conn,bool remove)
```

接続オブジェクト conn に関連付けられたプール内の接続をリサイクルします。

RemoveAllIdleConnections()

```
IRISPoolManager.RemoveAllIdleConnections();
```

すべての接続プールからアイドル状態の接続を削除します。

RemoveAllPoolConnections()

```
IRISPoolManager.RemoveAllPoolConnections();
```

接続の状態にかかわらず、すべての接続とすべてのプールを削除します。

5.2 IRISConnection クラス

ClearPool()

```
IRISConnection.ClearPool(conn);
```

接続 conn に関連付けられた接続プールをクリアします。

ClearAllPools()

```
IRISConnection.ClearAllPools();
```

接続プール内のすべての接続を削除してプールをクリアします。

5.3 接続パラメータのオプション

以下の各テーブルは、接続文字列で使用可能なすべてのパラメータを示しています。

- ・ [必須パラメータ](#)
- ・ [接続プーリング・パラメータ](#)
- ・ [その他の接続パラメータ](#)

5.3.1 必須パラメータ

以下のパラメータはすべての接続文字列に必要です（“[接続の作成](#)”を参照してください）。

server

代替名 : ADDR, ADDRESS, DATA SOURCE, DATASOURCE, HOST, NETWORK ADDRESS, NETWORKADDRESS

IP アドレスまたはホスト名。例 : Server = localhost

port

接続の TCP/IP ポート番号を指定します。例 : Port = 51774

namespace

代替名 : DATABASE, INITIAL CATALOG

接続先のネームスペースを指定します。例 : Namespace = USER

password

代替名 : PWD

ユーザのパスワード。例 : Password = SYS

user id

代替名 : USERID, UID, USER, USERNAME, USR

ユーザのログイン名を設定します。例 : User ID = _SYSTEM

5.3.2 接続プーリング・パラメータ

以下のパラメータは接続プーリングのさまざまな機能を定義します (“[接続プーリング](#)” を参照してください)。

connection lifetime

代替名 : CONNECTIONLIFETIME

接続のリセット・メカニズムが有効な場合に、アイドル状態のプールされた接続をリセットするまで待機する秒単位の長さ。既定は 0 です。

connection reset

代替名 : CONNECTIONRESET

プールされた接続のリセット・メカニズムを有効にします (CONNECTION LIFETIME と共に使用)。既定は false です。

max pool size

代替名 : MAXPOOLSIZE

この特定の接続文字列に対する接続プールの最大サイズ。既定は 100 です。

min pool size

代替名 : MINPOOLSIZE

この特定の接続文字列に対する接続プールの最小または初期サイズ。既定は 0 です。

pooling

接続プーリングを有効にします。既定は `true` です。

5.3.3 その他の接続パラメータ

以下のパラメータを必要に応じてオプションで設定できます。

application name

アプリケーション名を設定します。

connection timeout

代替名：**CONNECT TIMEOUT**

失敗と見なすまでに接続の確立を試行する、秒単位の時間の長さを設定します。既定は 30 です。この設定が OS で定義された既定の最大値より大きい場合は無視されます。OS 固有の値は、通常 20 ～ 40 秒程度です。

current language

このプロセスの言語を設定します。

logfile

ロギングを有効にしてログ・ファイルの場所を設定します。

packet size

TCP のパケット・サイズを設定します。既定は 1024 です。

PREPARSE CACHE SIZE

リサイクル処理を適用する前に解析前のキャッシュで保持する SQL コマンド数の上限を設定します。既定は 200 です。

sharedmemory

localhost または 127.0.0.1 で共有メモリ接続を有効または無効にします。例：`SharedMemory=false` は共有メモリを無効にします。既定は `true` です。

so rcvbuf

TCP の受信バッファ・サイズを設定します。既定値は 0 です (システムの既定値を使用)。

so sndbuf

TCP の送信バッファ・サイズを設定します。既定値は 0 です (システムの既定値を使用)。

ssl

SSL/TLS でクライアント・サーバ接続を保護するかどうかを指定します ("InterSystems IRIS との通信に SSL/TLS を使用するのための .NET クライアントの構成" を参照)。既定は `false` です。

tcp nodelay

TCP nodelay オプションを設定します。既定は `true` です。

transaction isolation level

接続の `System.Data.IsolationLevel` 値を設定します。

workstation id

プロセス ID のワークステーション名を設定します。

