U2 Toolkit for .NET

Version 2.2.2

May 2017

UNDK-222-OH-01

Notices

Edition

Publication date: May 2017
Book number: UNDK-222-OH-01
Product version: U2 Toolkit for .NET 2.2.2

Copyright

Trademarks

Examples

License agreement

Note

Corporate information

Rocket Software, Inc. develops enterprise infrastructure products in four key areas: storage, networks, and compliance; database servers and tools; business information and analytics; and application development, integration, and modernization.

Website: www.rocketsoftware.com

Rocket Global Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA

To contact Rocket Software by telephone for any reason, including obtaining pre-sales information and technical support, use one of the following telephone numbers.

Country	Toll-free telephone number
United States	1-855-577-4323
Australia	1-800-823-405
Belgium	0800-266-65
Canada	1-855-577-4323
China	800-720-1170
France	0800-180-0882
Germany	08-05-08-05-62
Italy	800-878-295
Japan	0800-170-5464
Netherlands	0-800-022-2961
New Zealand	0800-003210
South Africa	0-800-980-818
United Kingdom	0800-520-0439

Contacting technical support

The Rocket Community is the primary method of obtaining support. If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Community and report
a problem, download an update, or read answers to FAQs. To log in to the Rocket Community or to request a Rocket Community account, go to www.rocketsoftware.com/support.

In addition to using the Rocket Community to obtain support, you can use one of the telephone numbers that are listed above or send an email to support@rocketsoftware.com.

1 Table of Contents

1.	Table of Contents	1-3
	U2 Toolkit for .NET Help	4
	New in this release	5
	Supported versions of Microsoft Visual Studio	6
	U2 Toolkit for .NET	7-8
	U2 Toolkit for .NET Provider	9
	Architecture	10-11
	Getting Started	12
	U2 Toolkit for .NET Provider system requirements	13
	Installing U2 Toolkit for .NET Provider	14
	Tracing and logging in U2 Toolkit for .NET Provider	15
	Configuring a trace file	16
	Preparing your U2 accounts	17
	Testing your connection	18-19
	Sample code	20
	Converting IBM.NET projects into U2 Toolkit for .NET applications	21
	Converting UO.NET projects into U2 Toolkit for .NET applications	22
	Connection pooling	23-25
	Connection pooling C# code sample	26-27
	Connection pooling VB.NET code sample	28-29
	Microsoft Performance Monitor support	30-31
	Migrating applications from UniObjects for .NET to U2 Toolkit for .NET	32-34
	Step-by-step migration example (UniObjects for .NET to U2 Toolkit for .NET)	35-36
	Additional keyword support	37
	Additional function support	38-39
	Native Visual Studio Integration	40-41
	Tutorial: Developing an application using ADO.NET (Native integration)	42
	Tutorial: Adding a reference the project (Native integration)	43
	Tutorial: Adding controls to the form (Native integration)	44-45
	Tutorial: Creating event handlers for the button controls (Native integration)	46-48
	Tutorial: Building the application (Native integration)	49
	Tutorial: Accessing U2 data through the DataSet Object Model (Native integration)	50
	DataSet Tutorial: Creating a new Windows application (Native integration)	51
	DataSet Tutorial: Adding a reference the project (Native integration)	52

DataSet Tutorial: Adding a new Dataset to the Windows application (Native integration) 53

Adding DataAdapters and tables to the DataSet (Native integration) 54

DataSet Tutorial: Building the application (Native integration) 55

Tutorial: Creating an EntityDataModel from U2 files (Native integration) 56

Entity Data Model Tutorial: Creating a new Windows application (Native integration) 57

Entity Data Model Tutorial: Adding controls to the form (Native integration) 58

Entity Data Model Tutorial: Installing Entity Framework 6.1.3 using Manage NuGet Packages (Native integration) 59

Entity Data Model Tutorial: Adding a new Entity Data Model to the Windows application (Native integration) 60

Entity Data Model Tutorial: Creating event handlers for the button controls (Native integration) 61

Entity Data Model Tutorial_Building the application (Native integration) 62

Tutorial: Creating a new Entity Data Model using Visual Studio 2013 to work with Entity Framework 6.1.3 63

Tutorial: Installing Entity Framework 6.1.3 using Manage NuGet Packages 64-65

Tutorial: Adding a class object to the project 66-67

Tutorial: Adding an SqlQuery statement 68-69

Tutorial: Developing an application using ADO.NET 70

Tutorial: Adding a reference the project 71

Tutorial: Adding controls to the form 72

Tutorial: Creating event handlers for the button controls 73-74

Tutorial: Building the application 75

Tutorial: Developing an application using UniObjects 76

Tutorial: Adding a reference to a project 77

Tutorial: Adding controls to the form 78

Tutorial: Creating event handlers for the button controls 79-81

Tutorial: Building the application 82

Tutorial: Developing an application using ADO.NET and UniObjects 83

Tutorial: Adding a reference to the application 84

Tutorial: Adding controls to the form 85

Tutorial: Building the application 86

U2 Toolkit for .NET Developer 87

U2 Toolkit for .NET Developer system requirements 88

Installing U2 Toolkit for .NET Developer 89

Authorizing U2 Toolkit for .NET Developer licenses 90

Generating a configuration code 91
Authorizing a license 92
Establishing a server connection using the Visual Studio Server Explorer 93-94
Viewing the advanced settings of a U2 server definition 95-100
Accessing U2 database files through the Dataset object model 101-102
Accessing U2 database files through the Entity Data Model 103
Adding a TableAdapter to a project 104
Developing Applications with Native Visual Studio Integration 105
Using @ phrases and @SELECT phrases with Native Visual Studio Integration 106
Optimistic concurrency 107
Turning optimistic concurrency on/off in Dataset applications 108
Turning optimistic concurrency on/off in Entity Data Model applications 109
Turning optimistic concurrency on and off in subroutines 110
Limitations of U2 Toolkit for .NET Developer 111
U2 Entity Data Provider for .NET 112
Tutorial: Developing an application using the U2 Entity Data Provider 113
Tutorial: Adding a reference to the project 114
Tutorial: Adding controls to the form 115
Tutorial: Creating a new Entity Data Model from a U2 data source 116
Tutorial: Creating event handlers for the button controls 117-118
Tutorial: Building the application 119
Tutorial: Developing an application to call a subroutine using a Native Access connection 120
Tutorial: Adding a reference to the project 121
Tutorial: Adding the subroutine calling code 122-123
Working with Entity Framework 6 124-125
Limitations of the U2 Entity Data Provider 126

U2 Toolkit for .NET Help

Welcome to U2 Toolkit for .NET help

Version 2.2.2

The help provides conceptual, task-based, and reference information about the U2 Toolkit for .NET.

You can search for a word or phrase in these help topics by selecting the Search tab and entering your search topic. To narrow the search results to an exact phrase, enclose the phrase within quotation marks, for example, "dictionary files."

The help topics cover the following areas:

- U2 Toolkit for .NET overview
- U2 Toolkit for .NET Provider
- Installing U2 Toolkit for .NET Provider
- Getting Started
- U2 Toolkit for .NET Developer
- U2 Entity Data Provider
- U2.Data.Client.Assembly

After you install the product, you can learn about the product interface, view tutorials, and find sample codes by reading the topics in the **Getting started** section.

Additional resources

For additional information about U2 products, training, and technical resources go to www.rocketsoftware.com/u2

New in this release

The following features have been added at the 2.2.2 release:

- The **First** keyword now works with a Native connection
    - This allows users to limit the number of items returned by using the **First** keyword at native connection

  The following example uses the **First** keyword to return the first two records:

  ```
  Action=Select;File=STUDENT;Attributes=ID,LNAME,FNAME;Sort.DSND=ID;First=2
  ```

- Support for descending order by using the **Sort.DSND** keyword for native queries

  The following example uses the **Sort.DSND** keyword to return the results in descending order:

  ```
  cmd.CommandText = string.Format("Action=Select;File=STUDENT;
   Attributes=FNAME,LNAME;Sort.DSND=FNAME")
  ```

Supported versions of Microsoft Visual Studio

The following table details the versions of Microsoft Visual Studio that are supported in U2 Toolkit for .NET.

U2 Toolkit for .NET (v2.2.2)	Visual Studio 2017	Visual Studio 2015	Visual Studio 2013	Visual Studio 2010/2012	Visual Studio Express 2015/2013/2012/2010
U2 Toolkit for .NET Provider	Yes	Yes	Yes	Yes	Yes
U2 Toolkit for .NET Developer	Yes	Yes	Yes	Yes	No

For more information about the changes in Visual Studio, refer to the Microsoft website: www.microsoft.com/visualstudio.

U2 Toolkit for .NET

The U2 Toolkit for .NET Provider provides a comprehensive ADO.NET provider, LINQ to Entity provider, and the native UniObjects for .NET API for the U2 databases. Use Microsoft Visual Studio 2010 or later, to build applications and take advantage of the powerful Microsoft .NET Framework and CLR.

The U2 Toolkit for .NET Developer allows you to easily design U2 applications within Visual Studio. U2 Toolkit for .NET works with both 32-bit and 64-bit Windows operating systems.

U2 Toolkit for .NET is made up of three primary components:

- **U2 Toolkit for .NET Provider** (ADO.NET Provider)
- **U2 Entity Data Provider for .NET** (LINQ to Entity)
- **U2 Toolkit for .NET Developer** (Visual Studio Add-ins for Server Explorer Integration)

Developers can use U2 Toolkit for .NET to take advantage of server-based capabilities, such as:

- Automatic Data Encryption (ADE)
- Secure Sockets Layer (SSL)
- **Connection Pooling**
- **TOXML ('FillWithTOXML Method' in the on-line documentation)**
- **TOJSON ('ExecuteJson Method' in the on-line documentation)**

U2 Toolkit for .NET is optimized for connection performance to a U2 database. The U2 Toolkit for .NET Provider is a suite of data-access technologies that are included in the .NET Framework class libraries. ADO.NET helps applications connect to a database and has been designed to be the data access model used by all server processes and applications running on the Microsoft platform. The U2 Toolkit for .NET Developer allows designers to use the drag-and-drop capabilities and code generation (C#/VB.NET) found within Visual Studio to create new U2 applications without the need for extra programming. The Developer is designed to present a simple interface to U2 databases. For example, you can access and manage U2 Connections in Visual Studio Server Explorer, view server-side object properties, retrieve and update data from tables and views, and generate ADO .NET and EDM code using the drag-and-drop functionality.

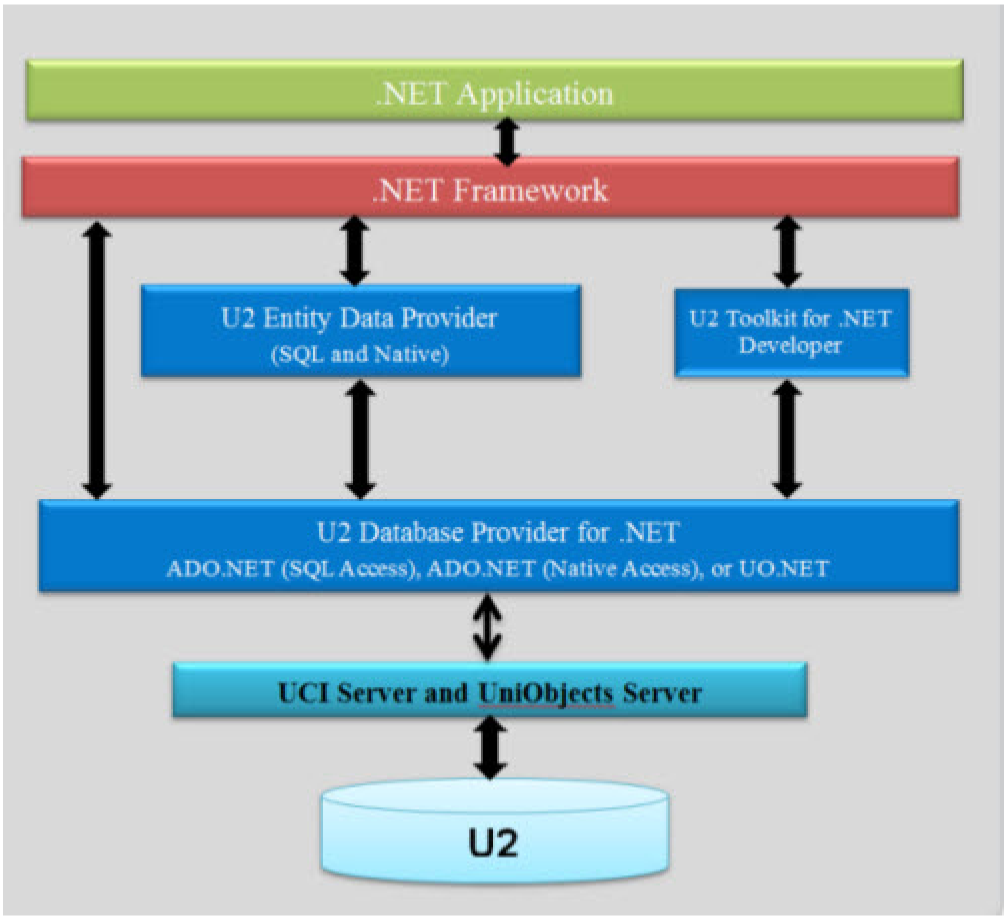
U2 Toolkit for .NET delivers a LINQ to Entity Framework provider for the U2 databases. Instead of relying on a physical storage model to access data, you can query and update data using a conceptual model generated as Entity Data Model (EDM) schemas. You can also develop LINQ to Entity applications to access data in UniData and UniVerse databases.

UniObjects for .NET (UO.NET) is an interface to UniData and UniVerse through Microsoft .NET. UniObjects for .NET is an application program interface (API) designed specifically for use with the U2 native data model. This interface is managed code written in C# Common Language Runtime (CLR). Software developers can use the UniObjects for .NET API and any CLR language to create applications and services. UO.NET supports SSL, NLS, and I18N.

.NET programmers can choose from a wide variety of languages, including C#, VB.NET, C++/CLI, and IronPython. Use the most effective language for your application to access your U2 data.

Architecture

The following diagram describes the component architecture of U2 Toolkit for .NET


.NET Application
.NET Framework
U2 Entity Data Provider
(SQL and Native)
U2 Toolkit for .NET Developer
U2 Database Provider for .NET
ADO.NET (SQL Access), ADO.NET (Native Access), or UO.NET
UCI Server and UniObjects Server
U2

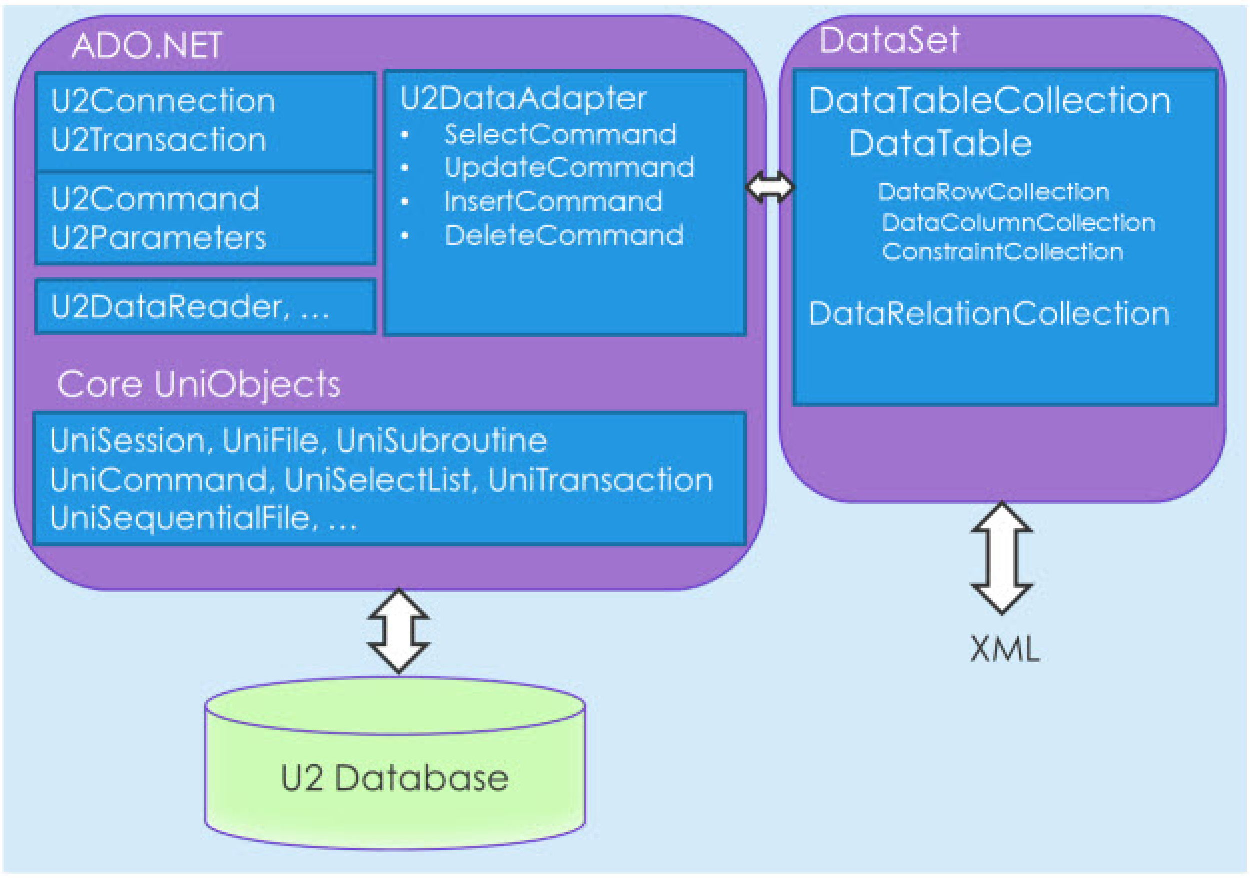
U2 Toolkit for .NET Provider

The U2 Toolkit for .NET Provider is the U2 ADO.NET provider. It is based on the Microsoft ADO.NET 4.0 specification. The provider supports both U2 Native Visual Studio Integration and SQL access. U2 Native Visual Studio Integration is achieved using the UniObjects classes, while SQL is accessed using the ADO.NET classes.

The U2 Toolkit for .NET Provider is the .NET Framework provider. It connects with U2 database servers to access and manipulate data, and execute commands and queries against the database. U2 Toolkit for .NET uses its UniRPC protocol to communicate with the UniData and UniVerse databases. It accesses the U2 data servers directly, without adding an OLE DB or ODBC layer, which increases performance.

Architecture

The following diagram shows the architecture of U2 Toolkit for .NET Provider and its relationship with the UniData and UniVerse databases.



ADO.NET

ADO.NET is a set of classes used by programmers to interact with data accessed in XML format through the OLE DB, ODBC, and JDBC interfaces. It comprises two primary components in the U2 Toolkit for .NET: The .NET Framework Provider and the DataSet class.

The ADO.NET DataSet class is a collection class used to read and write bulk UniRecord transactions in a UniData or UniVerse database.

UniObjects for .NET

UniObjects for .NET is an interface to the UniData and UniVerse databases through Microsoft .NET. UniObjects for .NET is a proprietary middleware application program interface (API) designed specifically for software development in the .NET Framework. This interface is managed code written in C# Common Language Runtime (CLR).

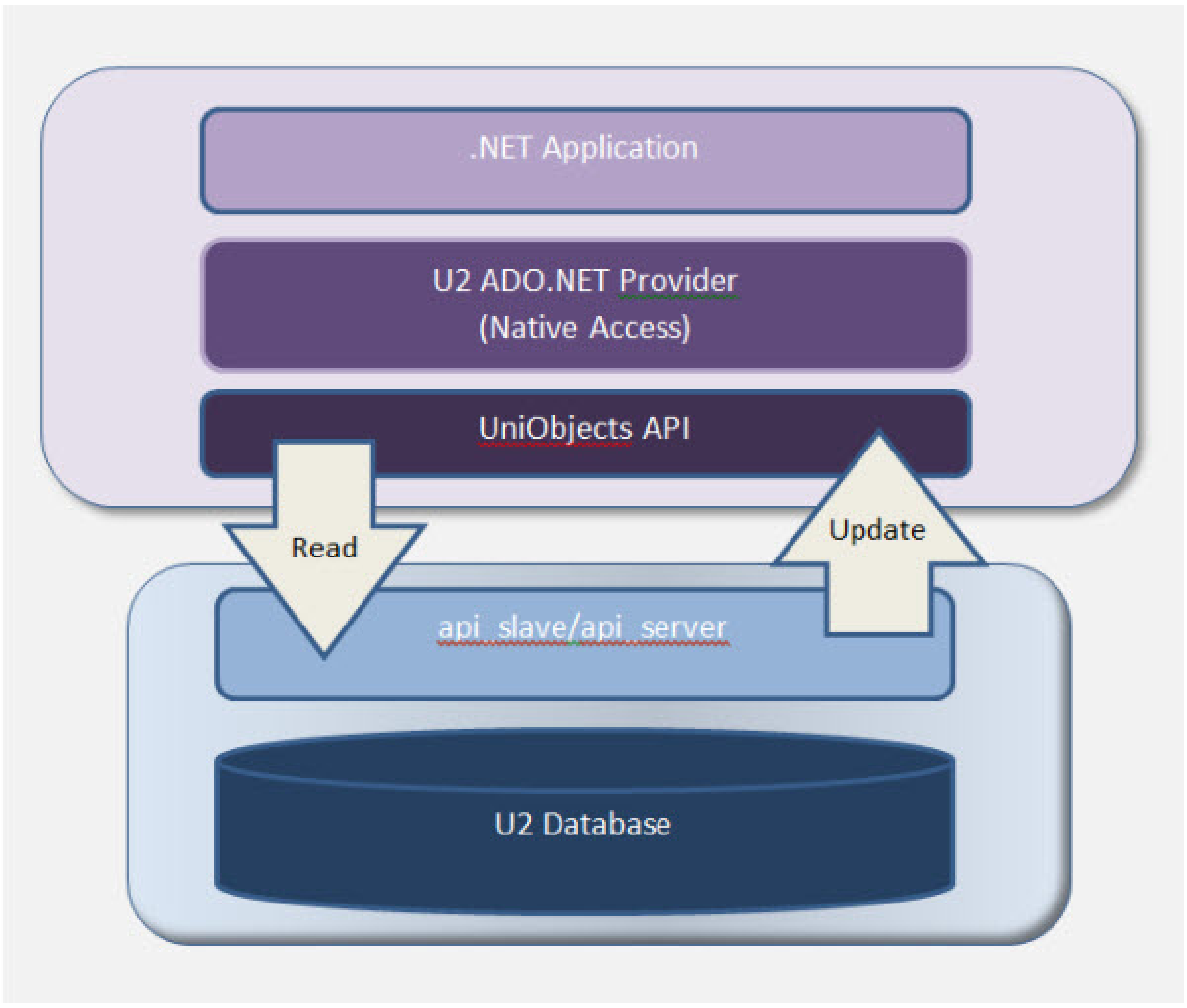
U2 Toolkit for .NET Provider

U2 Toolkit for .NET Provider is the data access model for .NET applications that connect to the UniData and UniVerse databases, providing both native U2 access and SQL access. It contains a collection of classes that allow you to connect to the UniData and UniVerse databases, execute commands, and read and write results, including the following:

- The UniSession class represents an open session to a UniData or UniVerse database.
- The UniFile and UniDictionary classes are used to access all file operations.
- The UniCommand class represents a Basic statement or stored procedure to execute against a UniData or UniVerse database.
- The UniTransaction class represents a Basic transaction to be made in a UniData or UniVerse database.
- The UniDataSet is a collection class used to read and write bulk UniRecord transactions in a UniData or UniVerse database.

Native Visual Studio Integration

The following diagram shows the architecture of how Native integration is used in U2 Toolkit for .NET Provider, and its relationship with the UniData and UniVerse databases.



Locating the U2 Toolkit driver

When there is more than one instance of the U2 Toolkit driver installed, such as when U2 Toolkit is deployed on an application server, it may not always be apparent where the U2 Toolkit drivers are located. To locate the drivers, first check the Global Assembly Cache (GAC) and then if it is not located there, check the application installation folder.

Getting Started with the U2 Toolkit for .NET Provider

The topics in this section detail how to install the U2 Toolkit for .NET and the specific steps you need to take to prepare your UniData or UniVerse accounts. It provides you with several short tutorials that walk you through the process of developing applications using ADO.NET and UniObjects, as well as a hybrid project that uses both ADO.NET and UniObjects. It also introduces you to the U2 Toolkit for .NET Test Connection Tool.

U2 Toolkit for .NET Provider system requirements

The following requirements must be met in order for the U2 Toolkit for .NET Developer to work correctly on your system.

System requirements

- Microsoft Windows 7, 8.1, 2008, 2012
- Microsoft.NET Framework 4.0, 4.5, 4.6
- Microsoft Visual Studio 2010, 2012, 2013, 2015, 2017
- ADO.NET Entity Framework 4.0, 5.0, 6.1

Supported versions of UniData and UniVerse

- UniData 7.3 or later
- UniVerse 11.2 or later

Supported version of Microsoft Visual Studio

U2 Toolkit for .NET	Visual Studio 2010/2012	Visual Studio 2013	Visual Studio 2015	Visual Studio 2017	Visual Studio Express
U2 Toolkit for .NET Provider	Yes	Yes	Yes	Yes	Yes
U2 Toolkit for .NET Developer	Yes	Yes	Yes	Yes	No

Supported versions the .NET Framework and the Entity Framework

U2 Toolkit for .NET	Microsoft .NET Framework	Microsoft Entity Framework
U2 Toolkit for .NET Provider	4.0	4.0, 5.0
U2 Toolkit for .NET Developer	4.5, 4.6	6.1

Note: Microsoft Visual Studio 2013 and later will not work with Entity Framework 5.0.

SSIS/SSRS (Optional)

Business Intelligence Development Studio	Visual Studio 2010/2012	Visual Studio 2013	Visual Studio 2015	Visual Studio 2017
SQL Access	Yes	Yes	Yes	Yes
Native Visual Studio Integration	Yes	Yes	Yes	Yes

Installing U2 Toolkit for .NET Provider

Complete the following steps to install U2 Toolkit for .NET on Windows.

Note: U2 Toolkit is a free driver. It is available for download from the Rocket Business Connect (RBC) website, at https://u2tc.rocketsoftware.com/matrix.asp.

Prerequisites

U2 Toolkit for .NET Provider system requirements

Procedure

1. From the U2 Toolkit for .NET installation screen, select **U2 Data Provider for .NET**. Click **Next**.
2. After accepting the licensing agreement, click **Next**.
3. By default, the installation process installs U2 Toolkit for .NET in the following directories:

   The installation path for the 32-bit provider on a 64-bit Windows 7 machine is C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider

   The installation path for the 64-bit provider on a 64-bit Windows 7 machine is C:\Program Files\Rocket Software\U2 Toolkit for .NET\U2 Database Provider

   The installation path for the 32-bit provider on a Windows 7/Windows XP machine is C:\Program Files\Rocket Software\U2 Toolkit for .NET\U2 Database Provider

4. Click **Next** to install U2 Toolkit for .NET in the default folder, or click **Browse** to search for a different folder.
5. By default, the installation process adds the Rocket U2 icon to the Program Folders list. Click **Next** to accept this default, or select a different folder from the Existing Folders list and then click **Next**.
6. Review the information listed in the **Start Copying Files** dialog box. If the information is correct, click **Next** to begin copying files. If the information is not correct, click **Back** to make changes.
7. Click **Finish** to complete the installation process.

Tracing and logging in U2 Toolkit for .NET Provider

U2NETDK provides a standard tracing and logging facility to trace the execution of U2NETDK code and to log the data in a user-specified destination. You can set the configuration for tracing and logging using the application's environment variables. By default, tracing and logging are turned off in U2NETDK.

Procedure

1. To access the environment variables, navigate to **Start** > **Control Panel** > **All Control Panel Items** > **System**.
2. Click **Advanced system settings** and select the **Advanced** tab.
3. Select **Environment Variables** to add or edit the environment variables for your system.

In the .NET Framework, there are four predefined trace levels:
1(error)
2(warning)
3(info)
4(verbose)

A U2NETDK application can select one of these four levels and specify a storage destination for the output of tracing and logging. The **UCINETTRACE** environment variable is used to define the log file folder. The **UCINETTRACESWITCH** environment variable is used to define the specific trace levels.

Example

```
UCINETTRACE=c:\temp
UCINETTRACESWITCH=4
```

In this example, tracing is turned on and is set to the 4(verbose) level. The log file name is c:\temp\ucinet_trace_<pid>.txt.

Configuring a trace file

Complete the following steps to configure a trace file.

Procedure

1. To access the environment variables, navigate to **Start > Control Panel > All Control Panel Items > System**.
2. Click **Advanced system settings** and select the **Advanced** tab.
3. Select **Environment Variables** and then set the U2NETDK_INSTALL_LOG environment variable.

For example:
U2NETDK_INSTALL_LOG = c:\temp

Verify installation

You can verify that U2 Toolkit for .NET installed correctly by viewing the contents in your Windows system's Control Panel. Navigate to **Control Panel > All Control Panel Items > Programs and Features**. The **U2 Toolkit for .NET Provider** appears in the list.

Preparing your U2 accounts

Before you can begin using your U2 accounts in SQL-based applications, you must make the U2 data accessible to those applications. The following list details where you can find the appropriate account preparation instructions for your application.

Version	Description
UniData 7.3 or later	To prepare your UniData 7.3 (or higher) accounts, you must use the U2 Metadata Manager (U2 MDM) tool. For more information about the U2 MDM tool, refer to the documentation at: http://docs.rocketsoftware.com/
UniData 7.2 or earlier	To prepare your UniData 7.2 (or prior) accounts, you must use the VSG and Schema API tools. For more information, refer to the documentation at: http://docs.rocketsoftware.com/
UniVerse 11.1 or earlier	To prepare your UniVerse 11.0 (or prior) accounts, you must use the HS.ADMIN tool. For more information about preparing your accounts using HS.ADMIN, refer to the UniVerse ODBC documentation at: http://docs.rocketsoftware.com/

Note

Account preparation is not required for Native Visual Studio Integration.

Testing your connection

While in the .NET developer, you can test the connection to U2 databases using the Test Connection Tool.

Procedure

1. Navigate to **Start > All Programs > Rocket U2 > U2 Toolkit for .NET > U2 Toolkit for .NET Provider**.
2. Select **Test Connection > Test Connection.exe**. The U2 Test Connection window opens, displaying two tabs: ADO.NET (SQL Access) and UniObjects (Native Visual Studio Integration).
3. In the U2 Test Connection dialog box, select either the ADO.NET tab or the UniObjects tab. To test the connection, enter information as described below:

   Database Type: Select **UniData** or **UniVerse**.

   Database: Enter the name of the database to which you are connecting.

   NOTE: The database name can be set to full path or a UniData database name or a UniVerse account name. When connecting to a UniData server using the database name, the name must be defined in both the ud_database file and the system UD.ACCOUNT file.

   Server: Enter the name or IP address of the computer on which UniData or UniVerse is running.

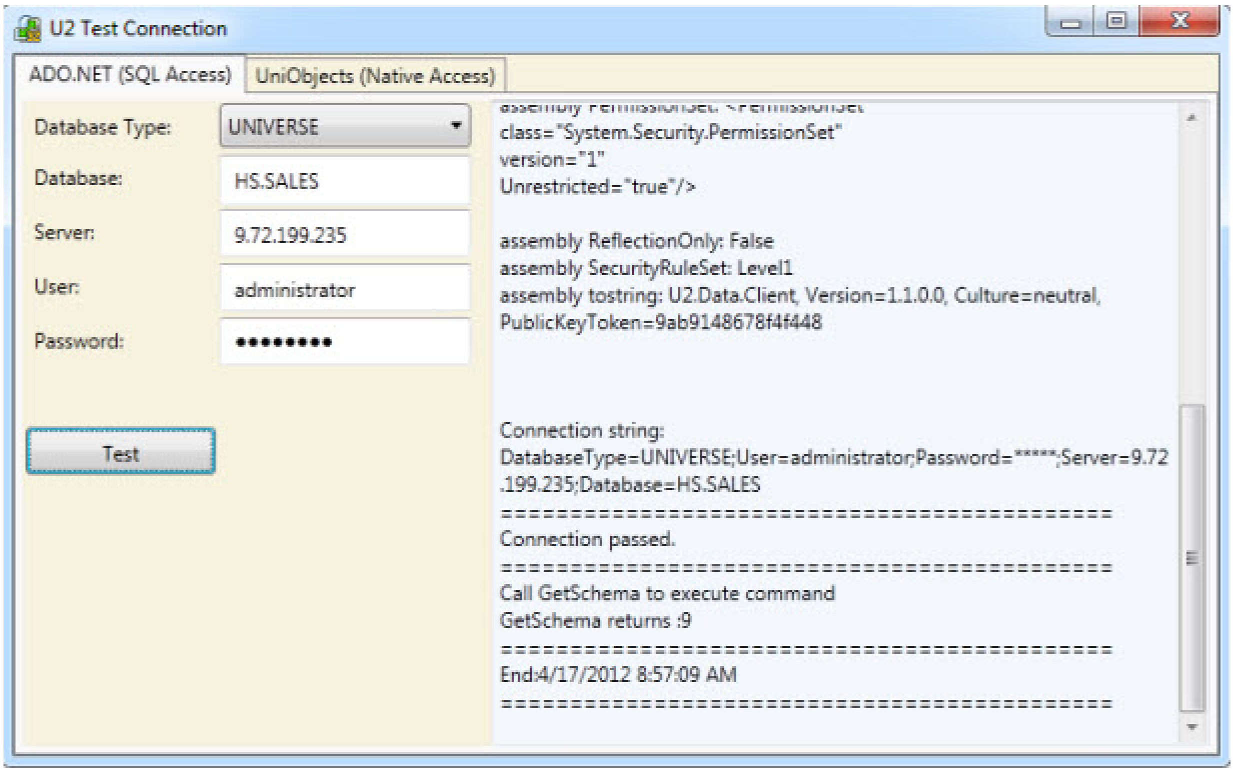
   User: Enter the administrator user name or the user name of a valid user on the server computer running UniData or UniVerse.

   Password: Enter the password for the administrator or user on the server computer.

   AccessMode: Enter the AccessMode for your application. For an ADO.NET connection, enter **ADO.NET.** For a UniObjects connection, enter **Native**.

   RpcServiceType: Select the correct RPC service type for the application. For UniData, select **udcs**. For UniVerse, select **uvcs**.

4. Click **Test**. The results of the connection test appear in the viewing pane, as shown:



Database Type: UNIVERSE
Database: HS.SALES
Server: 9.72.199.235
User: administrator
Password: ●●●●●●●●

```
class="System.Security.PermissionSet"
version="1"
Unrestricted="true"/>

assembly ReflectionOnly: False
assembly SecurityRuleSet: Level1
assembly tostring: U2.Data.Client, Version=1.1.0.0, Culture=neutral,
PublicKeyToken=9ab9148678f4f448


Connection string:
DatabaseType=UNIVERSE;User=administrator;Password=*****;Server=9.72
.199.235;Database=HS.SALES
=============================================
Connection passed.
=============================================
Call GetSchema to execute command
GetSchema returns :9
=============================================
End:4/17/2012 8:57:09 AM
=============================================
```

If you are testing the UniObjects connection, the screen will look slightly different.

Sample code

The U2 Toolkit for .NET Provider installation includes a large number of sample files, which are located by default in the following location:

- On 32-bit machines: C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples
- On 64-bit machines: C:\Program Files\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples

Samples are provided for both UniData and UniVerse and are written in both C# and VB.NET.

The following samples are provided:

- AutomaticDataEncryption
- Connection
- ConnectionPool
- DataAdapter
- DataAdapter_MultiValue
- DataAdapter_TOXML
- DataReader
- Subroutine
- EntityFramework
- EntityFramework_Subroutine
- EntityFramework_CodeFirst
- EntityFramework_CodeFirst2
- EntityFramework_CodeFirst_MultiValue1
- EntityFramework_CodeFirst_MultiValue2
- EntityFramework_ExecuteStoreQuery_UNNEST
- EntityFramework_POCO
- SSLConnection
- UO_UniCommand
- UO_UniDataSet
- UO_UniFile
- UO_UniSelectList
- UO_UniSubroutine

Viewing the samples

1. Open the sample .sln file that you want to view. For example, to view the Connection sample, double-click **Connection.sln** to open the file.
2. Add a reference to the **U2.Data.Client.dll**.
3. Change the connection string in the **Program.cs** file.
4. Compile the code.
5. Run the application.

Converting IBM.NET projects into U2 Toolkit for .NET applications

To convert an IBM.NET project to U2 Toolkit for .NET, modify the namespaces and class names, as shown in the following tables:

Namespaces

C# File

IBM.NET namespace	U2 Toolkit for .NET namespace
IBM.Data.DB2	U2.Data.Client
IBM.Data.DB2Types	U2.Data.ClientTypes

Classes

IBM.NET classes	U2 Toolkit for .NET classes
DB2Connection	U2Connection
DB2Command	U2Command
DB2DataReader	U2DataReader
DB2DataAdapter	U2DataAdapter
DB2Parameter	U2Parameter
DB2Transaction	U2Transaction
DB2ComandBuilder	U2ComandBuilder
DB2Error	U2Error

Converting UO.NET projects into U2 Toolkit for .NET applications

To convert a UniObjects for .NET project into U2 Toolkit for .NET applications, modify the namespaces and classes, as shown in the following tables:

Namespaces

C# File

UO.NET namespace	U2 Toolkit for .NET namespace
IBMU2.UODOTNET	U2.Data.Client.UO

Classes

UO.NET Classes	U2 Toolkit for .NET Classes
UniObjects	Not applicable
UniSession	UniSession (Get UniSession Object from U2Connection Object). See sample code included with the product.
UniCommand	UniCommand (Get UniCommand Object from UniSession Object). See sample code included with the product.
UniSubroutine	UniSubroutine (Get UniSubroutine Object from UniSession Object). See sample code included with the product.
UniSelectList	UniSelectList (Get UniSelectList Object from UniSession Object). See sample code included with the product.
UniTransaction	UniTransaction (Get UniTransaction Object from UniSession Object). See sample code included with the product.
UniSequentialFile	UniSequentialFile (Get UniSequentialFile Object from UniSession Object). See sample code included with the product.

Connection pooling

UniData 7.2 or later and UniVerse 10.3 or later support connection pooling with U2 Toolkit for .NET, UniObjects for Java, and UniObjects for .NET.

The term connection pooling refers to the technology that pools permanent connections to data sources for multiple threads to share. It improves application performance by saving the overhead of making a fresh connection each time one is required. Instead of physically terminating a connection when it is no longer needed, connections are returned to the pool and an available connection is given to the next thread with the same credentials.

You can activate connection pooling in your program (C#/VB.NET), or activate it through a configuration file.

Connection pool size

You can set the minimum and maximum size of the connection pool either in your program or through a configuration file. If you do not define these sizes, the minimum size defaults to 1 and the maximum size defaults to 1. The minimum size determines the initial size of the connection pool.

The size of the connection pool changes dynamically between the minimum and maximum sizes you specify, depending on the system demands. When there are no pooled connections available, U2 Toolkit for .NET either creates another connection, if the maximum connection pool size has not been reached, or keeps the thread waiting in the queue until a pooled connection is released or the request times out. If a pooled connection is idle for a specified time, it is disconnected.

License considerations

The actual size of a connection pool depends on the pooling licenses available on the server. For example, if you set a connection pool to a minimum size of 2 and a maximum size of 100, and you have 16 licenses available, the maximum connection pool size will be 16. If you only have 1 license available, U2 Toolkit for .NET does not create the connection pool at all, since the minimum size of 2 cannot be met.

Connection allocation

After U2 Toolkit for .NET allocates a pooled connection to a thread, the connection remains exclusively attached to that thread until it is explicitly freed by the thread. U2 Toolkit for .NET does not “clean up” the database environment side of a pooled connection before allocating it to a new thread with the same connection credentials. For example, UDT.OPTIONS settings, unnamed common, environment variables, and so forth remain from previous use.

Activating connection pooling

Connection pooling is activated as part of the U2Connection string, as shown in the following example:

```
U2ConnectionStringBuilder conn_str = new U2ConnectionStringBuilder();
```

```
conn_str.UserID = "user";
conn_str.Password = "pass";
conn_str.Server = "localhost";
conn_str.Database = "HS.SALES";
conn_str.ServerType = "UNIVERSE";
conn_str.Pooling = true;
conn_str.MinPoolSize = 1;
conn_str.MaxPoolSize = 5;
string sConnStr = conn_str.ToString();
```

You must specify different credentials for each connection pool in the U2Connection string. The following table describes each parameter of the syntax.

Parameter	Description
UserID	The name of the user connecting to the system.
Password	The password corresponding to the UserID.
Server	The name of the server to which you are connecting
Database	The name of the database to which you are connecting
ServerType	The type of server to which you are connecting. This will be either UniData or Universe.
AccessMode	Uci: Select this mode if using SQL Access. For Example : AccessMode="Uci" Native: Select this mode if using UO/Native Visual Studio Integration. For Example : AccessMode="Native"
RpcServiceType	The name of the rpc service. If you do not specify service_name, UniData and UniVerse default to defcs. If you do specify service_name, the service name must exist in the unirpcservices file.
PersistSecurityInfo	When set to **false** or **no** (strongly recommended), security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state. Resetting the connection string resets all connection string values including the password. Recognized values are **true**, **false**, **yes**, and **no**.

No two application processes can share connection pools.

You can also activate connection pooling in the configuration file, as shown in the following example:

```
<connectionStrings>

<add name="HSSALES_UV"
connectionString="Database=HS.SALES;User
ID=user;Password=pass;Server=192.34.111.4;Pooling=true;ServerType=universe;

PersistSecurityInfo=true;MinPoolSize=1;MaxPoolSize=5"
providerName="U2.Data.Client" />
```

If the User ID and Password are specified and Integrated Security is set to either true or false, the User ID and Password will be used. In other words, in this example, Integrated Security has no impact.

Specifying the size of the connection pool

To specify the size of the connection pool, use MinPoolSize to define the minimum number of connections, and MaxPoolSize to define the maximum number of connections, as shown in the following example:
MinPoolSize = 1;
MaxPoolSize = 5;

If you do not specify the minimum and maximum number of connections, UniData and UniVerse default to 1 for the minimum and 10 for the maximum.

Creating multiple connection pools

You can create as many connection pools as you like by issuing different connection strings.

Multiple connection pool example

```
POOL A
<connectionStrings>


    <add name="HSSALES_UV"
connectionString="Database=HS.SALES;User
ID=user;Password=pass;Server=192.34.111.4;Pooling=true;ServerType=universe;

PersistSecurityInfo=true;MinPoolSize=1;MaxPoolSize=5"
providerName="U2.Data.Client" />
</connectionStrings>

POOL B
<connectionStrings>


    <add name="HSSALES_UV"
connectionString="Database=MYDB;User
ID=user;Password=pass;Server=192.34.111.4;Pooling=true;ServerType=universe;

PersistSecurityInfo=true;MinPoolSize=1;MaxPoolSize=5"
providerName="U2.Data.Client" />
  </connectionStrings>.
```

The following table describes the configuration parameters for connection pooling:

Parameter	Description
Pooling	Gets or sets a Boolean value that indicates whether the connection will be pooled or explicitly opened every time that the connection is requested. By default, this value is **false**.
MinPoolSize	Gets or sets the minimum number of connections allowed in the connection pool for this specific connection string.
MaxPoolSize	Gets or sets the maximum number of connections allowed in the connection pool for this specific connection string.
Connection Lifetime	The length of time a connection can be idle before its returned to the pool (connection lifetime).
Connection Reset	Defines whether or not the current connection will be put in the connection pool when it is closed (connection reset).

When you close a session using connection pooling, U2 Toolkit for .NET does not close the connection. Instead, it makes the connection available in the connection pool.

Connection pooling C# code sample

This code sample shows how to use connection pooling with a native connection against a UniVerse server. It creates a C# .NET console project and adds the U2.Data.Client.dll driver to the project.

Example

```
// Create a C#.NET console project and add the U2.Data.Client.dll driver to the
project
// For SQL connections, it comments out the native mode code and uncomments the SQL
code
// Update the UserID, Password, and Server information

using System;
using U2.Data.Client;
using U2.Data.Client.UO;

namespace ConnectionPoolSample
{
    class Program
    {
        static void Main(string[] args)
        {
            U2Connection con = new U2Connection();
            U2ConnectionStringBuilder csb = new U2ConnectionStringBuilder();

            //Native Mode
            csb.AccessMode = "Native";     //Uncomment this line if using Native mode.
            csb.RpcServiceType = "uvcs";   //Uncomment this line if using Native mode.
Use "udcs"for UniData server.

            csb.Database = "HS.SALES";
            csb.UserID = "user";
            csb.Password = "password";
            csb.Server = "localhost";
            csb.ServerType = "universe";      //Use "unidata"for UniData server.
            csb.Connect_Timeout = 360;
            csb.ConnectionLifeTime = 300;
            csb.ConnectionReset = true;
            csb.MaxPoolSize = 2;
            csb.MinPoolSize = 1;
            csb.Pooling = true;

            con.ConnectionString = csb.ToString();
            con.Open();
            Console.WriteLine("Connected with pooling is true...........");

            //SQL mode
            //string commandString = "SELECT * FROM STATES";
            //U2Command cmd = con.CreateCommand();
            //cmd.CommandText = commandString;
            //cmd.ExecuteNonQuery();

            //Native mode
            string commandString = "LIST STATES";
            UniSession us1 = con.UniSession;
            UniCommand uniCmd = us1.CreateUniCommand();
            uniCmd.Command = commandString;
            uniCmd.Execute();
            // Get response string but not output
            string strNative = uniCmd.Response;

            con.Close();
            Console.WriteLine("Disconnected.........");
            Console.WriteLine("Enter to exit:");
            Console.Read();

        }
    }
}
```

Connection pooling VB.NET code sample

This code sample shows how to use connection pooling with an SQL connection against a UniData server. It creates a VB.NET console project and adds the U2.Data.Client.dll driver to the project.

Example

```
VB.NET Sample code for Connection Pool using SQL connection against UniData server

' Create a VB.NET console project and add the U2.Data.Client.dll driver to the project
' For Native connections, it comments out the SQL mode code and uncomments the Native
code
' Update the UserID, Password, and Server information

Imports System
Imports U2.Data.Client
Imports U2.Data.Client.UO

Namespace ConnectionPoolSample_VB
    Class Program
        Shared Sub Main(args As String())

            Try
                Dim con As New U2Connection()
                Dim csb As U2ConnectionStringBuilder = New U2ConnectionStringBuilder()

                'Native Mode
                'csb.AccessMode = "Native"     'Uncomment this line if using Native
mode.
                'csb.RpcServiceType = "udcs"   'Uncomment this line if using Native
mode. Use "uvcs"for Universe server.

                csb.Database = "demo"
                csb.UserID = "user"
                csb.Password = "password"
                csb.Server = "localhost"
                csb.ServerType = "unidata"      'Use "universe"for Universe server.
                csb.Connect_Timeout = 360
                csb.ConnectionLifeTime = 300
                csb.ConnectionReset = True
                csb.MaxPoolSize = 2
                csb.MinPoolSize = 1
                csb.Pooling = True

                con.ConnectionString = csb.ToString()
                con.Open()
                Console.WriteLine("Connected with pooling is true...........")

                'SQL Mode
                Dim commandString As String = "SELECT * FROM STATES"
                Dim cmd As U2Command = con.CreateCommand()
                cmd.CommandText = commandString
                cmd.ExecuteNonQuery()

                'Native Mode
                'Dim us1 As UniSession = con.UniSession
                'Dim cmd As UniCommand = us1.CreateUniCommand()
                'cmd.Command = "LIST STATES"
                'cmd.Execute()
                'Dim response_str As String = cmd.Response   'Get response string but
not output

                con.Close()

            Catch e As Exception

                Dim s As String = e.Message
                If e.InnerException IsNot Nothing Then
                    s &= e.InnerException.Message
                End If

                Console.WriteLine(s)
            Finally
                Console.WriteLine("Disconnected.........")
                Console.WriteLine("Enter to exit:")
                Dim line As String = Console.ReadLine()
            End Try
        End Sub

    End Class
End Namespace
```

Microsoft Performance Monitor support

U2 Toolkit for .NET 1.3.0 and later supports the Microsoft Performance Monitor in two ways:

- Monitoring connections in a connection pool
- Monitoring non-pooled connections

In both cases, U2 users can graph connection pool usage with the U2 Toolkit for .NET performance counters.

Serial number	Performance counter	Description
1	U2.Data.Client: Current # of pooled and non pooled connections	Current number of connections, pooled or not.
2	U2.Data.Client: Current # pooled connections	Current number of connections in all pools associated with the process.
3	U2.Data.Client: Current # connection pools	Current number of pools associated with the process.
4	U2.Data.Client: Peak # pooled connections	The highest number of connections in all pools since the process started.
5	U2.Data.Client: Total # failed connects	The total number of connection open attempts that have failed for any reason.
6	U2.Data.Client: Total # of failed command	Returns the total number of attempts to execute a command or subroutine that failed for any reason since the process started.

Monitoring connections in a connection pool

To minimize the cost of opening connections, U2 Toolkit for .NET uses an optimization technique called connection pooling, which minimizes the cost of repeatedly opening and closing connections. The performance monitor can be used to monitor these connection pools.

For example, if you use a connection string to turn on connection pooling using the following parameters, then you can monitor any or all of the U2 performance counters.

Connection pool example

```
<connectionStrings>

    <add
name="HSSALES_UV" connectionString="Database=HS.SALES;User
ID=user;Password=pass;Server=192.34.111.4;Pooling=true;ServerType=universe;
PersistSecurityInfo=true;MinPoolSize=1;MaxPoolSize=5"
providerName="U2.Data.Client" />
  </connectionStrings>
```

Monitoring non-connections/commands/subroutines

You can also monitor connections, commands, and subroutines made outside of a connection pool, such as for monitoring U2Commands (for SQL access), UniCommands, and UniSubroutines (for Native Visual Studio Integration).

In the following example, a connection string is used with the U2Command : CALL SLEEP 10, to monitor that the command will run for 10 seconds.

```
<connectionStrings>

    <add
name="HSSALES_UV" connectionString="Database=HS.SALES;User
ID=user;Password=pass;Server=192.34.111.4;ServerType=universe;

PersistSecurityInfo=true;"
providerName="U2.Data.Client" />
  </connectionStrings>
```

Prerequisite

You must have administrative privileges to use the performance monitor. If you are not the administrator, navigate to C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\tools. Right-click the **SetupPerformanceCounter.exe** file and select **Run as administrator**. Click the **Create Performance Counter** button and then close the dialog box.

Procedure

1. Navigate to **Start** >**Control panel** > **Administrative tools** > **Performance monitor**.
2. Click the **Performance Monitor** node the tree view. This opens the performance monitor.
3. Click the green plus (+) sign to add the U2 Toolkit for .NET counters.
4. Browse through the available counters and select **.NET Data Provider for U2 Database**. Click **Add**.
5. Click **OK**.
6. A graph of the selected counters appears in the editor. You can select any or all of the counters.

Migrating applications from UniObjects for .NET to U2 Toolkit for .NET

The U2 Toolkit for .NET is an interface to the UniData and UniVerse (U2) databases through the Microsoft .NET framework. U2 users have long had access to the .NET framework using UniObjects for .NET (UODOTNET), however, much of the functionality of UODOTNET is outdated and is now in maintenance mode. U2 is encouraging users to migrate to U2 Toolkit for .NET, which is a much more robust .NET interface and is fully supported.

Migrating UODOTNET applications involves making a few small changes to the following:

- Assembly Reference
- Namespace
- Connection functionality
- Connection Pooling parameters
- Configuration files

Assembly reference

UniObjects for .NET	U2 Toolkit for .NET
UODOTNET.dll	U2.Data.Client.dll

Namespaces

UniObjects for .NET	U2 Toolkit for .NET
IBMU2.UODOTNET	U2.Data.Client U2.Data.Client.UO

Connection functionality

Changes to connection functionality are now made in the **U2Connection** class. Previously, changes were made using the UniObjects class.

UniObjects for .NET	U2 Toolkit for .NET

UniSession us1 = UniObjects.OpenSession("localhost","user","pass","demo","udcs");	U2ConnectionStringBuilder lbdr = new U2ConnectionStringBuilder();
	lbdr.UserID = "user";
	lbdr.Password = "pass";
	lbdr.Server = "localhost";
	lbdr.Database = "demo";
	lbdr.ServerType = "unidata";
	lbdr.AccessMode = "Native";
	lbdr.RpcServiceType = "udcs";
	
	string lConnStr = lbdr.ToString();
	
	U2Connection lConn = new U2Connection();

	lConn.ConnectionString = lConnStr;
	lConn.Open();
	UniSession us1 = lConn.UniSession;

Connection pooling parameters

All connection pool settings must be set in the connection string.

UniObjects for .NET	U2 Toolkit for .NET
UOPooling	Pooling
MinPoolSize	MinPoolSize
MaxPoolSize	MaxPoolSize
IdleRemoveThreshold	ConnectionLifeTime
IdleRemoveExecInterval	ConnectionReset

Configuration files

Make changes to either the **App.Config** file for Windows applications or the **Web.Config** file for web applications.

UniObjects for .NET	U2 Toolkit for .NET

`<UO.NET>`
`  <General>`
`      <add key="SocketTimeOut" value="300000" />`
`  </General>`
`  <ConnectionPooling>`
`    <add key="ConnectionPoolingOn" value="1" />`
`    <add key="ConnectionPoolingOn" value="1" />`
`    <add key="MinimumPoolSize" value="1" />`
`    <add key="MaximumPoolSize" value="16" />`
`    <add key="IdleRemoveThreshold" value="10000" />`
`    <add key="IdleRemoveExecInterval" value="60000" />`
 	
`  </ConnectionPooling>`
`</UO.NET>`
	`<connectionStrings>`
	`  <add name="DemoConnectionString"`
	`    connectionString="Database=demo;User ID=user;Password=pass;`
	`      Server=localhost;Persist Security Info=True;`
	
	`ServerType=unidata;AccessMode=Uci;Pooling=true;`
	`      MinPoolSize=1;MaxPoolSize=10"`
	` providerName="U2.Data.Client" />`
	` </connectionStrings>`

SSL enablement

If SSL is enabled, add the following to the connection string:

U2 Toolkit for .NET
conn_str.SSLConnection = True;
conn_str.SslIgnoreCertificateNameMismatch = true;
conn_str.SslCheckCertificateRevocation = false;
conn_str.SslIgnoreIncompleteCertificateChain = true

Related links

U2 Toolkit for .NET

Converting IBM.NET projects into U2 Toolkit for .NET applications

Converting UO.NET projects into U2 Toolkit for .NET applications

Step-by-step migration example (UniObjects for .NET to U2 Toolkit for .NET)

This topic will walk you through the steps required to migrate an existing UniObjects for .NET application into U2 Toolkit for .NET.

Prerequisite

The following example is that of an existing UniObjects for .NET application, in which you can see the deprecated UO.NET assembly references, namespaces, and connection functions.

Procedure

1. Change the assembly reference.
   - Remove the reference to the **UODOTNET.dll**.
   - Add a reference to the **U2.Data.Client.dll**.
2. Change the namespace.
   - Remove the **IBMU2.UODOTNET** namespace.
   - Add the **U2.Data.Client** namespace and the **U2.Data.Client.UO** namespace.
3. Replace all references to the UniObjects class with the U2Connection class.

Result

The changes to the code should now look similar to the following example.

UniObjects for .NET example

Step 2

```
using System;
using IBMU2.UODOTNET;

namespace IBMU2.Connection
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Connection
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            UniSession us1 = null;

            try
            {
                us1 = UniObjects.OpenSession("localhost", "user
", "pass", "demo", "udcs");
            }
            catch (Exception e)
            {
                if (us1 != null && us1.IsActive)
                {
                    UniObjects.CloseSession(us1);
                    us1 = null;
                }
                Console.WriteLine("");
                string s = "Connection Failed : " + e.Message;
                Console.WriteLine(s);
            }
            finally
            {
                if (us1 != null && us1.IsActive)
                {
                    Console.WriteLine("");
                    string s = "Connection Passed";
                    Console.WriteLine(s);
                    UniObjects.CloseSession(us1);
                }

            }
        }
    }

}
```

Step 3

U2 Toolkit for .NET example

Step 2

```
using System;
//using IBMU2.UODOTNET;
using U2.Data.Client;
using U2.Data.Client.UO;


namespace U2.Connection
{
        /// <summary>
        /// Summary description for Class1.
        /// </summary>
        class Connection
        {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            UniSession us1=null;
            U2Connection lConn = null;
try
{
        //us1 = UniObjects.OpenSess("localhost","user","pass","demo","udcs");

                U2ConnectionStringBuilder lbdr = new
                U2ConnectionStringBuilder();
                lbdr.UserID = "user";
                lbdr.Password = "pass";
                lbdr.Server = "localhost";
                lbdr.Database = "demo";
                lbdr.ServerType = "unidata";
                lbdr.AccessMode = "Native";
                lbdr.RpcServiceType = "udcs";
                string lConnStr = lbdr.ToString();
                lConn = new U2Connection();
                lConn.ConnectionString = lConnStr;
                lConn.Open();
                us1 = lConn.UniSession;
                UniFile lf = us1.CreateUniFile("STUDENT");
        }
        catch(Exception e)
        {
                if(us1 != null && us1.IsActive)
                {
                //UniObjects.CloseSession(us1);
                lConn.Close();

                 us1= null;
         }
                 Console.WriteLine("");
                 string s = "Connection Failed : " + e.Message;
                 Console.WriteLine(s);
         }
         finally
         {
                 if(us1 != null && us1.IsActive)
                 {
                 Console.WriteLine("");
                 string s = "Connection Passed";
                 Console.WriteLine(s);
                 //UniObjects.CloseSession(us1);
                 lConn.Close();
                                 }
                         }
                 }
         }
}
```

Step 3

Additional keyword support

U2 Toolkit allows users to utilize the following additional built-in functions.

First

The FIRST keyword allows users to limit the number of items returned in a native query. When the FIRST keyword is used, the first N records are retrieved according to the natural scan positions of all the records in the file. If the query needs to sort the output, the sorting will be done after the first N records already are collected.

Example

The following example uses the First keyword to return the first two records:

```
Action=Select;File=STUDENT;Attributes=ID,LNAME,FNAME;Sort.DSND=ID;First=2
```

Sort.DSND

The Sort.DSND keyword allows users to sort native queries by descending order.

Example

The following example uses the Sort.DSND keyword to return the results in descending order:

```
cmd.CommandText = string.Format("Action=Select;File=STUDENT;Attributes=FNAME,LNAME;Sort.DSND=FNAME")
```

Additional function support

U2 Toolkit allows users to utilize the following additional built-in keywords. Beginning at v2.2.2, U2 functions can be used in UniObjects without requiring an active connection.

LEFT()

The LEFT() function returns the left part of a character string with the specified number of characters.

Syntax

```
LEFT(field_name,length)
```

Arguments

field_name

Is an expression of character or binary data. *field_name* can be a constant, variable, or column. *field_name* can be of any data type.

length

Is a positive integer that specifies how many characters of the *field_name* will be returned.

Example

```
>SELECT FNAME, SUBSTRING(LNAME FROM 1 FOR 3) FROM CUSTOMER

>SELECT FNAME, SUBSTRING(LNAME FROM 1 FOR 3) FROM CUSTOMER;
First Name..   SUBSTRING ( LNAME FROM 1 FOR 3 )

Diana          Mor
Jill           Kah
Betty          Bur
David          Arg
Kenneth        Wil
Martha         Gil
Andrew         McC
Steven         Hol
Laurie         Pat
Samuel         Smi
Nicole         Orl
Skip           Lew

12 records listed.
```

UniSession.NullSession()

This function allows users to create a new UniDynArray object without an active U2 connection.

Syntax

```
UniSession uss = UniSession.NullSession();
```

Example

```
UniSession uss = UniSession.NullSession();

String AM = Convert.ToChar(254).ToString();
String VM = Convert.ToChar(253).ToString();
String SM = Convert.ToChar(252).ToString();

String strRec = "one" + AM + "twoA" + VM + "twoB" + AM + "three";

// Create dynamic array from string

UniDynArray rec = uss.CreateUniDynArray(strRec);

Console.WriteLine("1: " + rec.Extract(1));
Console.WriteLine("2,1: " + rec.Extract(2, 1));
Console.WriteLine("2,2: " + rec.Extract(2, 2));
Console.WriteLine("3: " + rec.Extract(3));
uss = null;
```

Native Visual Studio Integration

Beginning at version 2.1.0, U2 Toolkit for .NET supports Native Visual Studio Integration to U2 applications through the ADO.NET Provider and LINQ to Entity Provider. Native Visual Studio Integration users are able to fully access U2 files and subroutines without having to make changes to their account dictionaries. For example, U2 Toolkit for .NET can natively access or modify System Builder accounts and subroutines. U2 Toolkit users can seamlessly use single values, multivalues, or multi-subvalues in SQL syntax to create a DataSet, a DataAdapter, or Entity Data Model.

Native Visual Studio Integration allows users to easily perform a variety of functions, such as:

- Accessing multivalue U2 files, fields, and subroutines directly from the ADO.NET provider
- Accessing multivalue U2 files, fields, and subroutines directly from the LINQ-to-Entity provider
- Accessing multivalue U2 files, fields, and subroutines directly from within the Microsoft Visual Studio Server Explorer
- Filtering on dictionary fields using the @ and @SELECT phrases
- Accessing System Builder accounts
- Accessing U2 files with JSON using the **ExecuteJson() ('ExecuteJson Method' in the on-line documentation)** API

Syntax examples

The U2 ADO.NET Provider allows users to use either ANSI SQL syntax or Action syntax to perform SELECT, UPDATE, INSERT, and DELETE operations, as shown:

Syntax	ANSI SQL example	Action syntax example
SELECT	`SELECT CUSTID,FNAME,LNAME,BUY_DATE FROM CUSTOMER`	`Action=Select;File=CUSTOMER;Attributes=CUSTID, PRODID,BUY_DATE, FNAME ,LNAME;Where=CUTID>5;Sort=CUSTID`
UPDATE	`UPDATE CUSTOMER SET FNAME='{0}',BUY_DATE='{1}' WHERE CUSTID=444555`	`Action=Update;File=CUSTOMER;Attributes=CUSTID=?, PRODID=?, BUY_DATE=?, FNAME=?, LNAME=?, ;Where=CUSTID=? AND Z_MV_KEY=?`
INSERT	`INSERT INTO CUSTOMER (CUSTID,FNAME,BUY_DATE) VALUES('{0}','{1}','{2}')`	`Action=Insert;File=CUSTOMER;Attributes=CUSTID=?, PRODID=?, BUY_DATE=?, FNAME=?, LNAME=?`
DELETE	`DELETE FROM CUSTOMER WHERE CUSTID=444555`	`Action=Delete;File=CUSTOMER;Where=CUSTID=? AND PRODID=? AND BUY_DATE=? AND FNAME=? AND LNAME=? AND Z_MV_KEY=?`

Limitations

Complex SQL statements or clauses such as JOIN, GROUPBY, and SAMPLE are not supported at this time.

Code examples

Several code examples are provided as part of the U2 Toolkit for .NET installation package. These code examples are installed to the following default directories:

- C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples\C#\UniVerse\NativeAccess
- C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples\VB.NET\UniVerse\NativeAccess

- C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples\C#\UniData\NativeAccess
- C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples\VB.NET\UniData\NativeAccess
- C:\Program Files\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples\C#\UniVerse\NativeAccess
- C:\Program Files\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples\VB.NET\UniVerse\NativeAccess
- C:\Program Files\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples\C#\UniData\NativeAccess
- C:\Program Files\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples\VB.NET\UniData\NativeAccess

Developing an application using ADO.NET (Native integration)

This example demonstrates how to create a simple application using ADO.NET with a Native Visual Studio Integration connection. The application calls the HS.SALES account in UniVerse. After dropping a DataGridView onto the form, the DataGridView control automatically loads the information from the CUSTOMER file and allows you to make changes to the CUSTOMER file using the U2 CommandBuilder class to update the file.

Prerequisite

- U2 Toolkit for .NET 2.1.0
- UniVerse 10.3 or later
  or
- UniData 7.1 or later
- Visual Studio 2010, 2012, or 2013

Procedure

1. Open a project in Visual Studio. This project was created in Visual Studio 2012.
2. Select the programming language with which you want to work. The examples in this document are all created using C#.
3. Select **File > New Project**.
4. When the **New Project** dialog box opens, select **Windows Forms Application**.
5. In the **name** field, enter a name for the project. The project name in this example is **WindowsFormsApplication1**.
6. In the **location** field, enter the location where the project will reside. The location in this example is **C:\NativeAccessTutorials**.
7. Click **OK**.

Result

The new project opens in the form designer.

The Microsoft Visual Studio Form Design window has three main panes: The form designer, the Solution Explorer, and the Properties pane. You can create and edit your application in the form designer by dragging items from the Visual Studio Toolbox onto the form. The Solution Explorer provides a navigation tree view of all the files associated with your project. The property pane allows you to set the properties of the form and of the individual objects on the form.

Next

Tutorial: Adding a reference the project (Native integration)

Tutorial: Adding a reference the project (Native integration)

You must add a reference to the U2 Data.Client.dll.

Prerequisite

Developing an application using ADO.NET (Native integration)

Procedure

1. In the Solution Explorer, right-click the **References** node and select **Add Reference** from the context menu.
2. Select the **Browse** tab and navigate to (on a 64-bit Windows machine):

   C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\bin\.NETFramework\v4.0
3. Select the **U2.Data.Client.dll** and click **OK**.

Next

Tutorial: Adding controls to the form (Native integration)

Tutorial: Adding controls to the form (Native integration)

After adding a reference to the to the topic, you can add some controls to the form.

Prerequisite

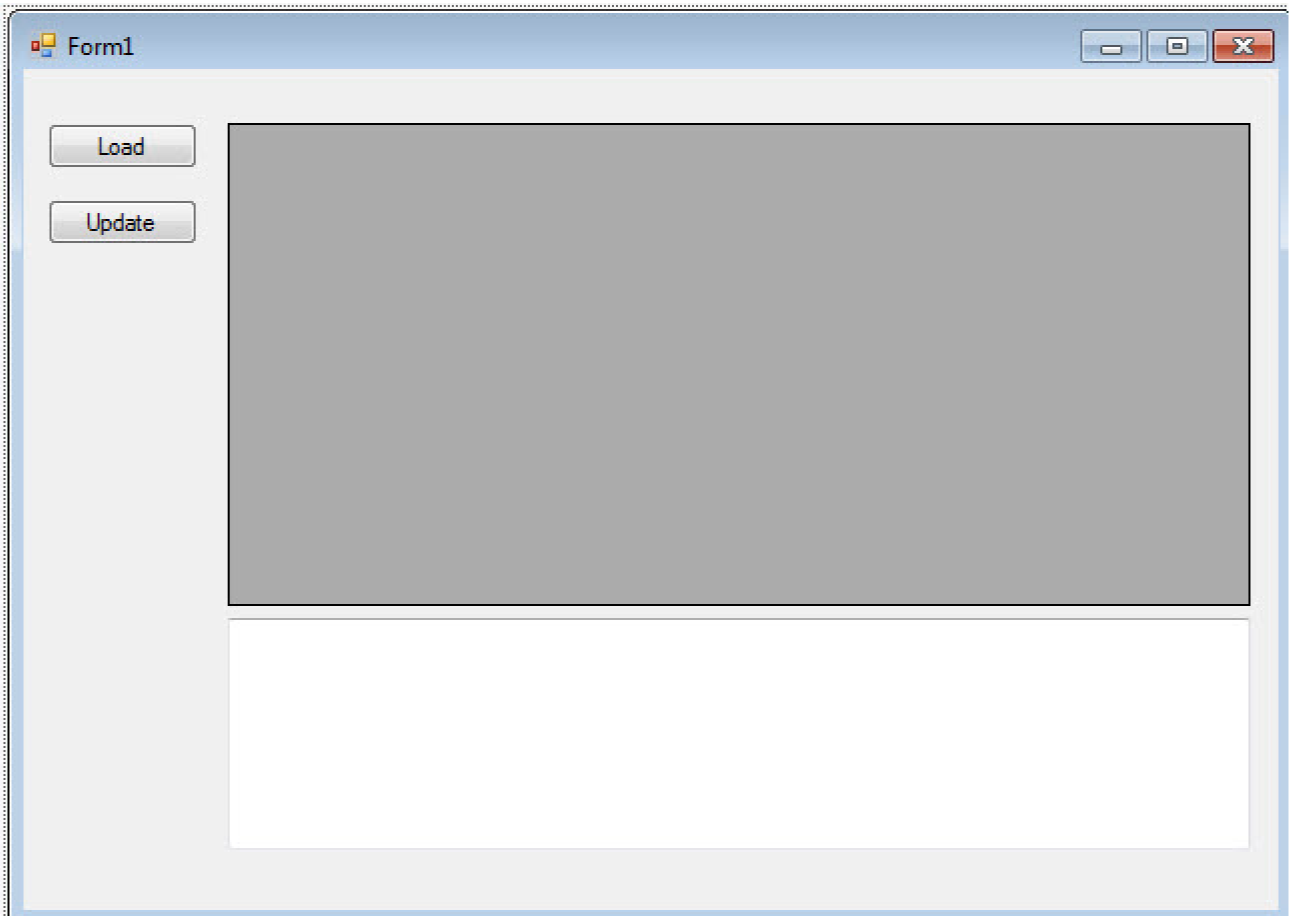
Tutorial: Adding a reference the project (Native integration)

Procedure

1. From the Visual Studio Toolbox, drag two buttons onto the form. In the properties window, change the button properties as follows:
   - Change the Text property of Button1 to **Load**
   - Change the Text property of Button2 to **Update**
2. From the Visual Studio Toolbox, drag a **DataGridView** control onto the form.
3. Add a **TextBox** control onto the form and position it beneath the DataGridView. Change the TextBox properties as follows:
   - Change the Multiline property of **True**

Result

The form should look similar to the following:



Next

Tutorial: Creating event handlers for the button controls (Native integration)

Tutorial: Creating event handlers for the button controls (Native integration)

After updating the properties, create an event handler for each button click. This requires you to add a small amount of code to the form.

Prerequisite

Tutorial: Adding controls to the form (Native integration)

Procedure

1. Double-click **Load** to create an event handler for the button.
2. In the code editor, add the following *using* statement to the form:
   ```
   using U2.Data.Client
   ```
3. In the code editor, declare the class member variable;
   ```
   private U2Connection m_Conn;
   private DataSet m_DS;
   private U2DataAdapter m_da;
   ```
4. Add the following code to the **Button1** click event. Note that you may need to modify the connection string settings.

```
private void button1_Click(object sender, EventArgs e)
    {
Try
        {
            this.textBox1.AppendText(Environment.NewLine + "Start..." +
Environment.NewLine);
            U2ConnectionStringBuilder conn_bldr = new U2ConnectionStringBuilder();
            conn_bldr.UserID = "user";
            conn_bldr.Password = "mypass";
            conn_bldr.Server = "192.33.11.31";
            conn_bldr.ServerType = "universe";
            conn_bldr.Database = "HS.SALES";
            conn_bldr.AccessMode = "Native";
            conn_bldr.RpcServiceType = "uvcs";
            conn_bldr.UseFastGetRecordID = true;
            string lConnStr = conn_bldr.ConnectionString;
            m_Conn = new U2Connection();
            m_Conn.ConnectionString = lConnStr;
            m_Conn.Open();
            Console.WriteLine("Connected...");
            U2Command cmd = m_Conn.CreateCommand();
            //CUSTID,FNAME,LNAME : Single Value
            //PRODID, BUY_DATE    : Multi Value
            cmd.CommandText = string.Format("SELECT CUSTID,FNAME,LNAME,
         PRODID,BUY_DATE FROM CUSTOMER WHERE CUSTID > 0 ORDER BY CUSTID ");
            m_da = new U2DataAdapter(cmd);
            m_DS = new DataSet();
            m_da.Fill(m_DS);
            this.dataGridView1.DataSource = m_DS.Tables[0].DefaultView;
            this.textBox1.AppendText("Total Rows:" + m_DS.Tables[0].Rows.Count +
Environment.NewLine);
            //m_Conn.Close();
```

```
                this.textBox1.AppendText(Environment.NewLine + "End...");
            }
          catch (Exception e2)
            {
                string lErr = e2.Message;
                if (e2.InnerException != null)
                {
                    lErr += lErr + e2.InnerException.Message;
                }
                this.textBox1.AppendText(Environment.NewLine + lErr);
            }

        }
```

Note

Edit the login credentials required for the server connection.

4. Return focus to the Form designer and then double-click the **Update** button.
5. Add the following code to the **Button2** click event:

```
  private void button2_Click(object sender, EventArgs e)
      {
          try
          {
              U2CommandBuilder builder = new U2CommandBuilder(m_da);
              m_da.UpdateCommand = builder.GetUpdateCommand();
              DataTable dt = m_DS.Tables[0];
              DataRow[] lDataRowCollection = dt.Select("CUSTID=2");
              int i = 1;
              foreach (DataRow item in lDataRowCollection)
              {
                  item["FNAME"] = item["FNAME"] + "_update";// modify single value
                  DateTime ld = Convert.ToDateTime(item["BUY_DATE"]);
                  DateTime ld2 = ld.AddDays(1);
                  item["BUY_DATE"] = ld2;//modify multi-value
                  i++;

              }
              m_da.Update(m_DS);//use DataAdapter's Update() API


          }
          catch (Exception e2)
          {

              string lErr = e2.Message;
              if (e2.InnerException != null)
              {
                  lErr += lErr + e2.InnerException.Message;
              }
              this.textBox1.AppendText(Environment.NewLine + lErr);
          }
      }
  }
```

Result

The Button1 event handler creates a connection the to the HS.SALES database and then loads the information from the CUSTOMER file into the data grid.

The Button2 event handler uses the U2 CommandBuilder class to generate an update statement. It checks to see whether any changes have been made to the record(s). If it detects any changes, it first checks to see how many changes have been made and then repopulates the DataGrid using the generated Update statement.

Next

Tutorial: Building the application (Native integration)

Tutorial: Building the application (Native integration)

After creating all of the controls and event handlers, the application is ready to build.

Prerequisite

Tutorial: Creating event handlers for the button controls (Native integration)

Procedure

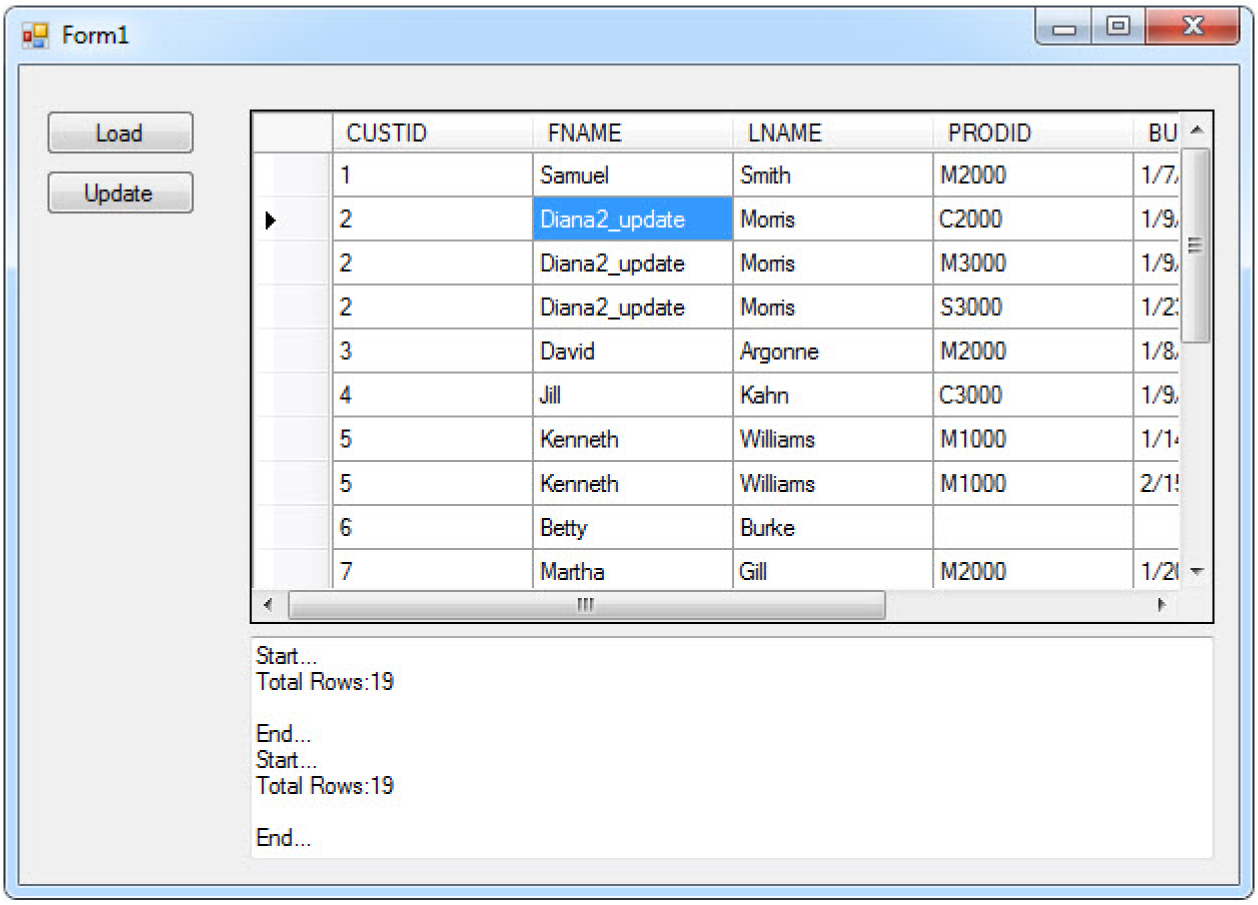
1. Select **Build** > **Build Solution** from the Visual Studio toolbar.
2. Run the application. To do this, click **Debug** on the Visual Studio toolbar, and then select the **Start Without Debugging** option. This opens a working version of the application you just created.

   Tip: If you want to debug the application, you can choose to **Start Debugging** instead.

3. Click **Load**. The customer information populates in the GridView, as shown:
4. In the grid, locate the name **Diana** in the FNAME field of the CUSTOMER table. Click in the cell to edit the name, and change it to **Diana2**. Click **Update**.

Result

The change is saved to your account, as shown:



	CUSTID	FNAME	LNAME	PRODID	BU
	1	Samuel	Smith	M2000	1/7/
▶	2	Diana2_update	Morris	C2000	1/9/
	2	Diana2_update	Morris	M3000	1/9/
	2	Diana2_update	Morris	S3000	1/2:
	3	David	Argonne	M2000	1/8/
	4	Jill	Kahn	C3000	1/9/
	5	Kenneth	Williams	M1000	1/1
	5	Kenneth	Williams	M1000	2/1!
	6	Betty	Burke		
	7	Martha	Gill	M2000	1/2(

```
Start...
Total Rows:19

End...
Start...
Total Rows:19

End...
```

DataSet Tutorial: Accessing U2 data through the DataSet Object Model (Native integration)

You can use the Native Visual Studio Integration to access your data files through the Dataset model. In the Dataset object model, U2 files and attributes are mapped to the DataTable or Column and can be viewed as columns or rows.

This example was created in Visual Studio 2012.

Prerequisite

- U2 Toolkit for .NET 2.1.0
- UniVerse 10.3 or later
  or
- UniData 7.1 or later
- Visual Studio 2010, 2012, or 2013

Procedure

1. Open Visual Studio. If the Server Explorer is not open, select **View Server Explorer**.
2. In the Server Explorer, right-click the **Data Connections** node and then select **Add Connection**.
3. Select **Change** and then select **U2 Database** from the Data source options. Click **OK** to return to the Add Connection dialog box.
4. Enter the appropriate connection information. For this example, enter the following information:
   - Set the Account name to **demo** (for UniData) or **HS.SALES** (for UniVerse)
   - Set the Database type to either **UniData** or **UniVerse**
   - Set the Access mode to **Native (UO Server)**- Set the RpcServiceType to **udcs, uvcs,** or **defcs**
5. Click **Test Connection**. The new connection node should now show in the Server Explorer.
6. Navigate to the new node and then expand the **Stored Procedures** node and the **U2Files** node.

Next Step

DataSet Tutorial: Creating a new Windows application (Native integration)

DataSet Tutorial: Creating a new Windows application (Native integration)

This example demonstrates how to create a simple application using ADO.NET's DataSet with a Native Visual Studio Integration connection. The application calls the HS.SALES account in UniVerse. It loads the information from the CUSTOMER file on to DataSet Designer.

Prerequisite

DataSet Tutorial: Accessing U2 data through the DataSet Object Model (Native integration)

Procedure

1. Open a project in Visual Studio. This project was created in Visual Studio 2012.
2. Select the programming language with which you want to work. The examples in this document are all created using C#.
3. Select **File > New Project**.
4. When the **New Project** dialog box opens, select **Windows Forms Application**.
5. In the **name** field, enter a name for the project. The project name in this example is **TestWalkthrough**.
6. In the **location** field, enter the location where the project will reside. The location in this example is **C:\NativeAccessTutorials**.
7. Click **OK**.

Result

The new project opens in the form designer.

The Microsoft Visual Studio Form Design window has three main panes: The form designer, the Solution Explorer, and the Properties pane. You can create and edit your application in the form designer by dragging items from the Visual Studio Toolbox onto the form. The Solution Explorer provides a navigation tree view of all the files associated with your project. The property pane allows you to set the properties of the form and of the individual objects on the form.

Next

DataSet Tutorial: Adding a reference the project (Native integration)

DataSet Tutorial: Adding a reference the project (Native integration)

You must add a reference to the U2 Data.Client.dll.

Prerequisite

DataSet Tutorial: Creating a new Windows application (Native integration)

Procedure

1. In the Solution Explorer, right-click the **References** node and select **Add Reference** from the context menu.
2. Select the **Browse** tab and navigate to (on a 64-bit Windows machine):

   C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\bin\.NETFramework\v4.0
3. Select the **U2.Data.Client.dll** and click **OK**.

Next

DataSet Tutorial: Adding a new Dataset to the Windows application (Native integration)

DataSet Tutorial: Adding a new Dataset to the Windows application (Native integration)

This example demonstrates how to create a simple application using ADO.NET with a Native Visual Studio Integration connection. The application calls the HS.SALES account in UniVerse. It loads the information from the CUSTOMER file on to a DataGridView control, and allows you to make changes to the CUSTOMER file using the U2 CommandBuilder class to update the file.

Prerequisite

DataSet Tutorial: Creating a new Windows application (Native integration)

Procedure

1. If it is not already available, open the **TestWalkthrough** project in Visual Studio.
2. Select **Project > Add New Item**.
3. Select **DataSet** from the list of available items.
4. Name the DataSet **CustomerDataSet** and then click **Add**.

Result

Visual Studio adds a file to the project called **CustomerDataSet.xsd**, which opens in the DataSet Designer.

Next

DataSet Tutorial: Adding DataAdapters and tables to the DataSet (Native integration)

DataSet Tutorial: Adding DataAdapters and tables to the DataSet (Native integration)

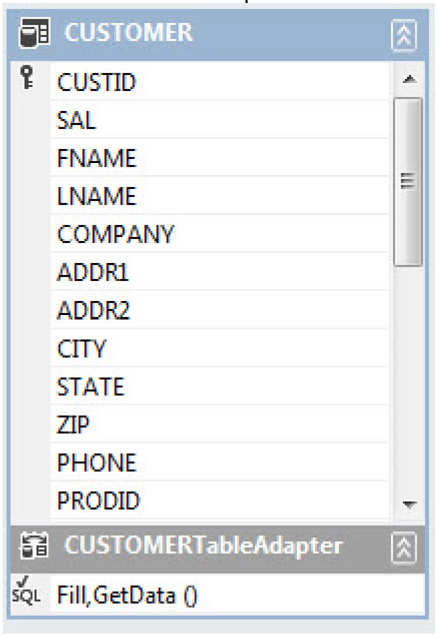
This example demonstrates how to create a simple application using ADO.NET with a Native Visual Studio Integration connection. The application calls the HS.SALES account in UniVerse. It loads the information from the CUSTOMER file on to a DataGridView control, and allows you to make changes to the CUSTOMER file using the U2 CommandBuilder class to update the file.

Prerequisite

DataSet Tutorial: Adding a new Dataset to the Windows application (Native integration)

Procedure

1. If it is not already available, open the **CustomerDataSet.xsd**, which opens in the DataSet Designer.
2. Expand the HS.SALES connection you created in **step 1**, and then select the **U2Files** node.
3. Drag the **CUSTOMER** table onto the DataSet Designer. The CUSTOMER data table and CUSTOMERTableAdapter are added to the DataSet Designer, as shown:



4. Select **View > Other Windows >Data Sources**. This opens the Data Sources menu.
5. From the Solution Explorer, double-click the **Form1.cs** file to open the Windows form designer.
6. From the Data Sources menu, drag the **CUSTOMER** data source onto the form.

Tip

Right-click on the TableAdapter and select **Preview Data** to preview the contents of the U2 database files accessed through this DataSet model.

Next

DataSet Tutorial: Building the application (Native integration)

DataSet Tutorial: Building the application (Native integration)

After creating all of the controls and event handlers, the application is ready to build.

Prerequisite

DataSet Tutorial: Adding DataAdapters and tables to the DataSet (Native integration)

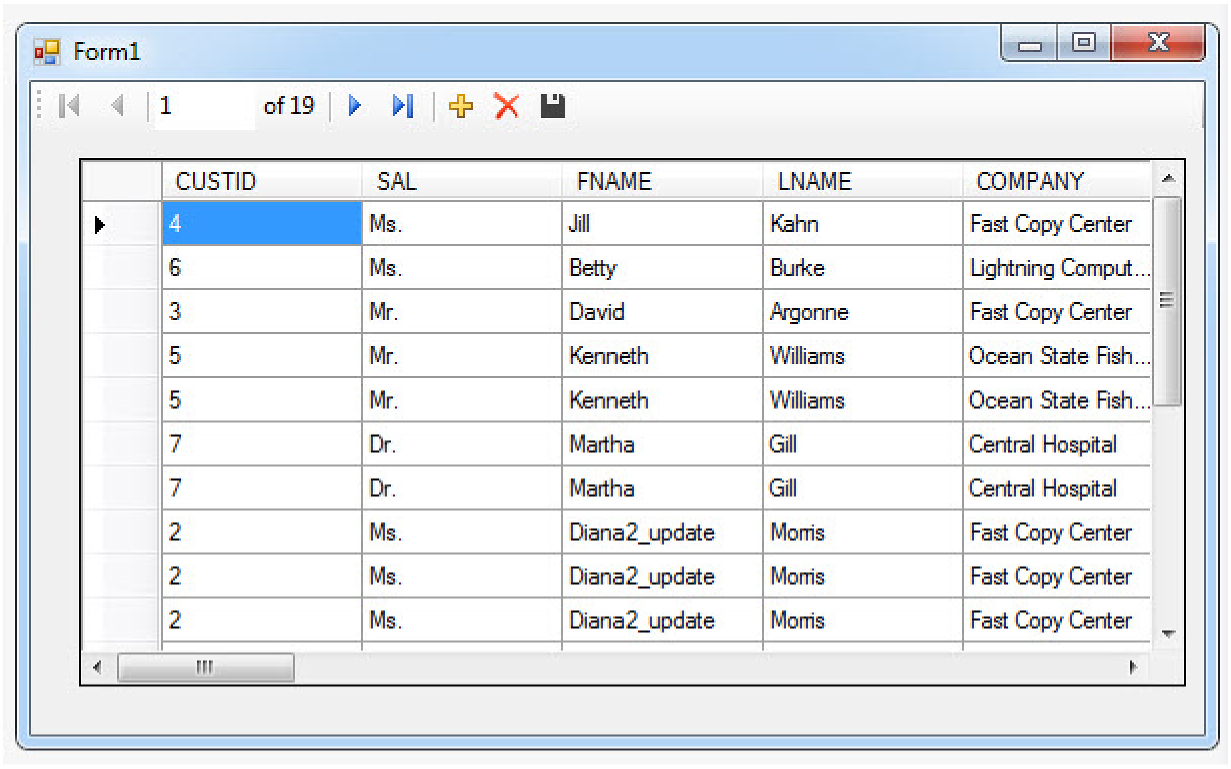
Procedure

1. Select **Build > Build Solution** from the Visual Studio toolbar.
2. Run the application. To do this, click **Debug** on the Visual Studio toolbar, and then select the **Start Without Debugging** option. This opens a working version of the application you just created.

   Tip: If you want to debug the application, you can choose to **Start Debugging** instead.

Result

The results open in a DataSet, as shown:



CUSTID	SAL	FNAME	LNAME	COMPANY
4	Ms.	Jill	Kahn	Fast Copy Center
6	Ms.	Betty	Burke	Lightning Comput...
3	Mr.	David	Argonne	Fast Copy Center
5	Mr.	Kenneth	Williams	Ocean State Fish...
5	Mr.	Kenneth	Williams	Ocean State Fish...
7	Dr.	Martha	Gill	Central Hospital
7	Dr.	Martha	Gill	Central Hospital
2	Ms.	Diana2_update	Morris	Fast Copy Center
2	Ms.	Diana2_update	Morris	Fast Copy Center
2	Ms.	Diana2_update	Morris	Fast Copy Center

Entity Data Model Tutorial: Creating an Entity Data Model from U2 files (Native integration)

You can use the Native Visual Studio Integration to access your data files through the Entity Data Model. You do not need to normalize the account using VSG (UniData) or HS.ADMIN (UniVerse). You can also execute subroutines natively.

This example was created in Visual Studio 2013.

Prerequisite

- U2 Toolkit for .NET 2.2.2
- UniVerse 10.3 or later
  or
- UniData 7.1 or later
- Visual Studio 2013
- .NET Framework 4.5

Procedure

1. Open Visual Studio. If the Server Explorer is not open, select **View Server Explorer**.
2. In the Server Explorer, right-click the **Data Connections** node and then select **Add Connection**.
3. Select **Change** and then select **U2 Database** from the Data source options. Click **OK** to return to the Add Connection dialog box.
4. Enter the appropriate connection information. For this example, enter the following information:
   - Set the Account name to **demo** (for UniData) or **HS.SALES** (for UniVerse)
   - Set the Database type to either **UniData** or **UniVerse**
   - Set the Access mode to **Native (UO Server)**
   - Set the RpcServiceType to **udcs, uvcs,** or **defcs**
5. Click **Test Connection**. The new connection node should now show in the Server Explorer.
6. Navigate to the new node and then expand the **Stored Procedures** node and the **U2Files** node.

Next

Entity Data Model Tutorial: Creating a new Windows application (Native integration)

Entity Data Model Tutorial: Creating a new Windows application (Native integration)

This example demonstrates a simple application using the LINQ to Entity data model. The application calls the HS.SALES account in UniVerse. It loads the information from the CUSTOMER file on to Entity Data Model.

Prerequisite

Entity Data Model Tutorial: Creating an Entity Data Model from U2 files (Native integration)

Procedure

1. Open a project in Visual Studio. This project was created in Visual Studio 2013.
2. Select the programming language with which you want to work. The examples in this document are all created using C#.
3. Select **File > New Project**.
4. When the **New Project** dialog box opens, select **Windows Forms Application**.
5. In the **name** field, enter a name for the project. The project name in this example is **EntityDataModelFromU2File**.
6. In the **location** field, enter the location where the project will reside. The location in this example is **C:\NativeAccessTutorials**.
7. Click **OK**.

Result

The new project opens in the form designer.

The Microsoft Visual Studio Form Design window has three main panes: The form designer, the Solution Explorer, and the Properties pane. You can create and edit your application in the form designer by dragging items from the Visual Studio Toolbox onto the form. The Solution Explorer provides a navigation tree view of all the files associated with your project. The property pane allows you to set the properties of the form and of the individual objects on the form.

Next

Entity Data Model Tutorial: Adding controls to the form (Native integration)

Entity Data Model Tutorial: Adding controls to the form (Native integration)

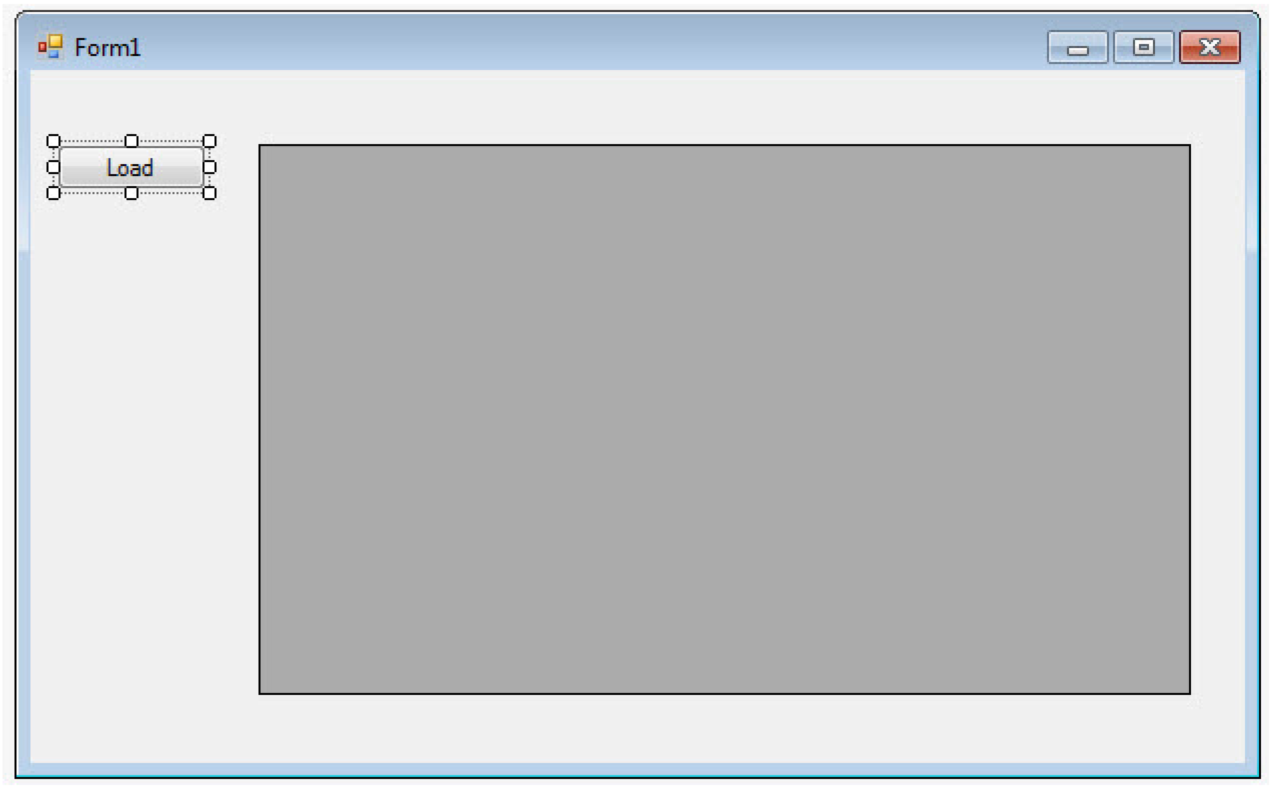
After adding a .dll reference to the project, you can add controls to the form.

Prerequisite

Entity Data Model Tutorial: Creating a new Windows application (Native integration)

Procedure

1. From the Visual Studio Toolbox, drag a button onto the form.
2. In the properties window, change the button Text property of Button1 to **Load**
3. From the Visual Studio Toolbox, drag a **DataGridView** control onto the form. Your form should look similar to the following:



Next

Entity Data Model Tutorial: Installing Entity Framework 6.1.3 using Manage NuGet Packages (Native integration)

Entity Data Model Tutorial: Installing Entity Framework 6.1.3 using Manage NuGet Packages (Native integration)

After adding the controls to the form, you can create an Entity Data Model from your U2 files.

Prerequisite

Entity Data Model Tutorial: Adding controls to the form (Native integration)

Procedure

2. Select **Project -> Manage NuGet Packages**.
3. Select **Browse**, and then select the EntityFramework package and click **Install**. This updates the App.cofig configuration file with the EntityFramework information.
4. In the Solution Explorer pane, double-click the **App.config** file.
5. Update the U2 data provider in the App.config provider list, as shown in the following example:

```
<providers>
      <provider invariantName="U2.Data.Client.4.5"
type="U2.Data.Client.Entity.U2ProviderServices,
 U2.Data.Client.Entity, Version=2.2.2.0, Culture=neutral,
PublicKeyToken=883335d992998a08"/>
</providers>

            Note: Visual Studio 2015 or 2017 projects will use U2.Data.Client.4.6
instead of U2.Data.Client.4.5.


<providers>
      <provider invariantName="U2.Data.Client.4.6"
type="U2.Data.Client.Entity.U2ProviderServices,
U2.Data.Client.Entity, Version=2.2.2.0, Culture=neutral,
PublicKeyToken=7f1dc11a3fe611eb"/>
 </providers>
```

6. Select **Build -> Rebuild Solution** to save your changes and rebuild the project.

Result

Visual Studio adds the Entity Framework 6.1.3 package to the project.

Next

Entity Data Model Tutorial: Adding a new Entity Data Model to the Windows application (Native integration)

Entity Data Model Tutorial: Adding a new Entity Data Model to the Windows application (Native integration)

After adding controls to the form, you are ready to create a new Entity Data Model from a U2 data source.

Prerequisite

Entity Data Model Tutorial: Installing Entity Framework 6.1.3 using Manage NuGet Packages (Native integration)

Procedure

1. If it is not already available, open the **EntityDataModelFromU2File** project in Visual Studio.
2. Select **Project > Add New Item**.
3. Select **ADO.NET Entity Data Model** from the list of available items.
4. Name the model **Customer** and then click **Add**.
5. After the Entity Data Model wizard opens, select **Generate from Database** and then click **Next**.
6. Select the data connection you defined in Step 1, and then choose how the sensitive data will display. Click **Next** to continue.
7. Select the database objects to be included in the Entity Data Model. For this example, select **Tables** and then **CUSTOMER**. Keep all other default selections.
8. Click **Finish**.

Result

Visual Studio adds a file to the project called **Customer.edmx** which opens in the Entity Data Model Wizard.

Next

Entity Data Model Tutorial: Creating event handlers for the button controls (Native integration)

Entity Data Model Tutorial: Creating event handlers for the button controls (Native integration)

After creating the Entity Data Model, you must add some event handlers to the form.

Prerequisite

Entity Data Model Tutorial: Adding a new Entity Data Model to the Windows application (Native integration)

Tip

This example uses the Entity data model (CUSTOMER.EDMX) file. Refer to the C# EntityFramework.sln sample code for details about this file. The sample code is located by default in the following location:

On 32-bit machines: C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples

On 64-bit machines: C:\Program Files\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples

Procedure

After adding the Entity Data Model data source, create an event handler for each button click. This requires you to add a small amount of code to the form.

1. Double-click **Load** to create an event handler for the button. The code editor opens.
2. Add the following code to the **Button1** click event:

```
private void button1_Click(object sender, EventArgs e)
   {
       try
       {
           Entities ctx = new Entities();
           var q = ctx.CUSTOMERs.ToList();
           this.dataGridView1.DataSource = q;
       }
       catch (Exception e2)
       {
           string lErr = e2.Message;
           if (e2.InnerException != null)
           {
               lErr += e2.InnerException.Message;
           }
           MessageBox.Show(lErr);
       }
```

Result

The Button1 event handler creates a connection the to the HS.SALES database and then loads the information from the CUSTOMER account into the data grid.

Next

Entity Data Model Tutorial_Building the application (Native integration)

Entity Data Model Tutorial_Building the application (Native integration)

After creating all of the controls and event handlers, the application is ready to build.

Prerequisite

Entity Data Model Tutorial: Creating event handlers for the button controls (Native integration)

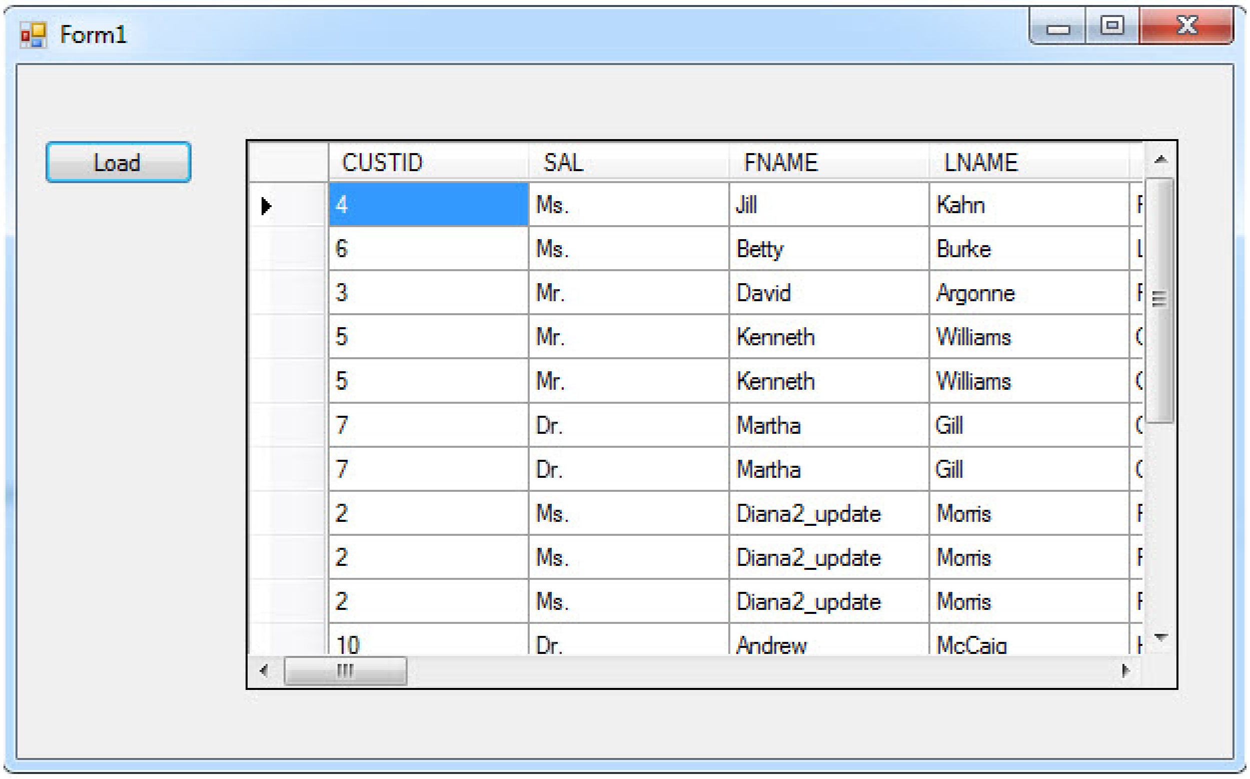
Procedure

1. Select **Build** > **Build Solution** from the Visual Studio toolbar.
2. Run the application. To do this, click **Debug** on the Visual Studio toolbar, and then select the **Start Without Debugging** option. This opens a working version of the application you just created.

   Tip: If you want to debug the application, you can choose to **Start Debugging** instead.

Result

The results open in a DataSet, as shown:



CUSTID	SAL	FNAME	LNAME
4	Ms.	Jill	Kahn
6	Ms.	Betty	Burke
3	Mr.	David	Argonne
5	Mr.	Kenneth	Williams
5	Mr.	Kenneth	Williams
7	Dr.	Martha	Gill
7	Dr.	Martha	Gill
2	Ms.	Diana2_update	Morris
2	Ms.	Diana2_update	Morris
2	Ms.	Diana2_update	Morris
10	Dr.	Andrew	McCaig

Tutorial: Creating a new Entity Data Model using Visual Studio 2013 to work with Entity Framework 6.1.3

In this example, you will create a C# console program that uses the U2 Entity Data Provider with Entity Framework 6.1.3, in a Visual Studio 2013 environment.

Prerequisite

- UniVerse 10.3 or later
  or
- UniData 7.1 or later
- Visual Studio 2013, 2015, 2017

Procedure

1. Open Visual Studio. This project was created in Visual Studio 2013.
2. Select the programming language with which you want to work. The examples in this document are all created using C#.
3. Select **File > New Project**. When the New Project dialog box opens, select **Windows > Console Application**.
4. In the name field, enter a name for the project. The project name in this example is **ConsoleApplication1**.
5. In the location field, enter the location where you want the project to reside. The location in this example is **C:\walkthrough**.
6. Click **OK**.

The Microsoft Visual Studio Form Design window has three main panes: The form designer, the Solution Explorer, and the Properties pane. You can create and edit your application in the form designer by dragging items from the Visual Studio Toolbox onto the form. The Solution Explorer provides a navigation tree view of all the files associated with your project. The property pane allows you to set the properties of the form and of the individual objects on the form.

The new project opens in the form designer.

Next

Tutorial: Installing Entity Framework 6.1.3 using Manage NuGet Packages

Tutorial: Installing Entity Framework 6.1.3 using Manage NuGet Packages

After creating the new Visual Studio Console Application project, you need to add the NuGet package to the project.

Prerequisite

Tutorial: Installing Entity Framework 6.1.3 using Manage NuGet Packages

Procedure

2. Select **Project -> Manage NuGet Packages**.
3. Select **Browse**, and then select the EntityFramework package and click **Install**. This updates the App.cofig configuration file with the EntityFramework information.
4. In the Solution Explorer pane, double-click the **App.config** file.
5. Update the U2 data provider in the App.config provider list with the CustomerContext connection string, as shown in the following example:

   Note: For the U2.Data.Client.Entity 4.6 driver, the PublicKeyToken is “7f1dc11a3fe611eb”. You can use the VS 2013 x86 native sn command with “-T” option to find the public token information.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
     EntityFramework, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"
       requirePermission="false"/>
  </configSections>
  <connectionStrings>
    <add name="CustomerContext" connectionString="Database=HS.SALES;User
ID=administrator;
     Password=password;Server=localhost;Pooling=false;ServerType=universe;
     ConnectTimeout=360;PersistSecurityInfo=true"
providerName="U2.Data.Client.4.5"/>
  </connectionStrings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5"/>
  </startup>
  <entityFramework>
    <providers>
      <provider invariantName="U2.Data.Client.4.5"
type="U2.Data.Client.Entity.U2ProviderServices,
       U2.Data.Client.Entity, Version=2.2.2.0, Culture=neutral,
PublicKeyToken=883335d992998a08"/>
    </providers>
  </entityFramework>
</configuration>
```

6. Select **Build -> Rebuild Solution** to save your changes and rebuild the project.

Result

Visual Studio adds the Entity Framework 6.1.3 package to the project.

Next

Tutorial: Adding a class object to the project

Tutorial: Adding a class object to the project

After adding the NuGet package to the project, you can add a class object to the project.

Prerequisite

Tutorial: Installing Entity Framework 6.1.3 using Manage NuGet Packages

Procedure

1. From the Visual Studio menu, select **Project > Add Class**.
2. Give the class a name. In this example, we use **Customer.cs**.
3. Add the following code in the class editor:

```
using System;

using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
namespace ConsoleApplication1
{
    public class Customer
    {
        Customer()
        {
        }
        [Key]
        public int CUSTID { get; set; }
        public string FNAME { get; set; }
        public string LNAME { get; set; }
        public string FULLADDR { get; set; }
    }
    public class CustomerContext : DbContext
    {
        public CustomerContext()
        {
        }
        public DbSet<Customer> Customers { get; set; }
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Customer>()
                .Property(s => s.FNAME)
                .IsRequired();
            modelBuilder.Conventions.Remove<IncludeMetadataConvention>();
        }
    }
}
```

Next

Tutorial: Adding an SqlQuery statement

Tutorial: Adding an SqlQuery statement

After adding a class object to the project, add a new SqlQuery statement to the Customer entity in the Program.cs file.

Prerequisite

Tutorial: Adding a class object to the project

Procedure

1. In the Solution Explorer pane, double-click the **Program.cs** file.
2. Add the following code in the editor:

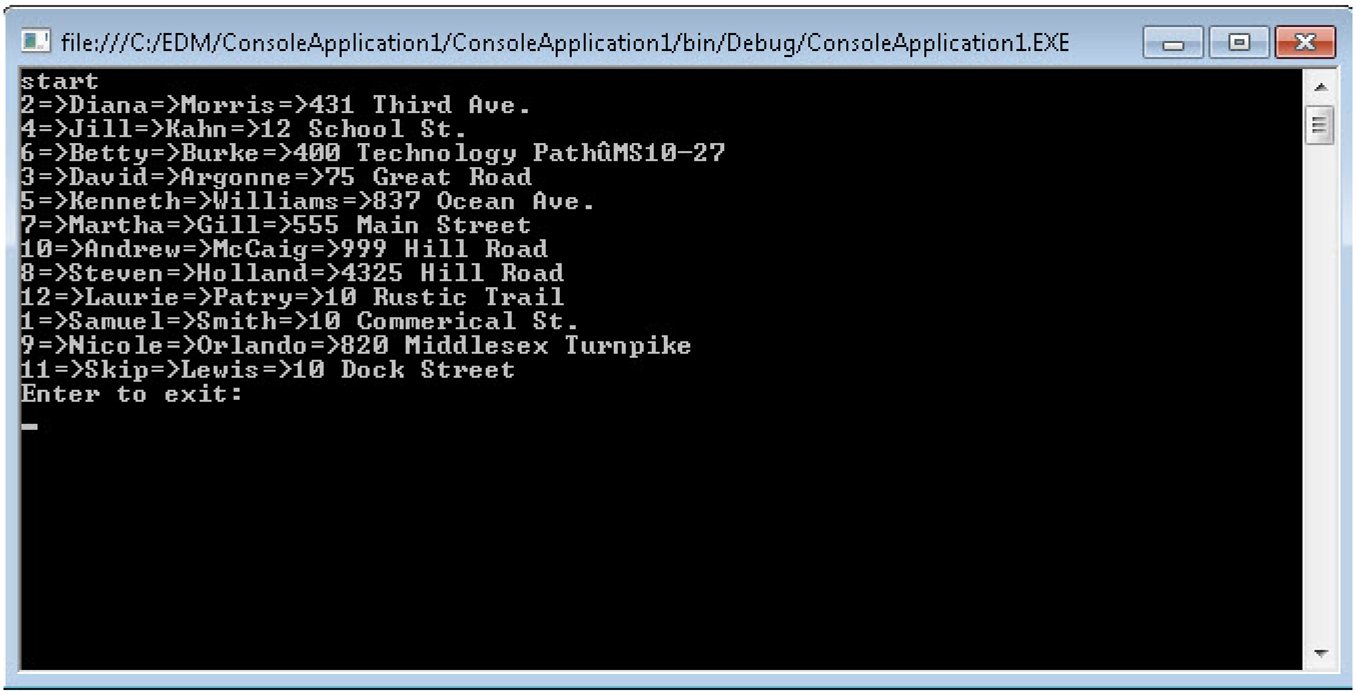
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("start");
                CustomerContext ctx = new CustomerContext();
                var t = ctx.Database.SqlQuery<Customer>("SELECT
CUSTID,FNAME,LNAME,FULLADDR FROM CUSTOMER");
                foreach (Customer item in t)
                {
                    Console.WriteLine(item.CUSTID + "=>" + item.FNAME + "=>" +
item.LNAME + "=>" + item.FULLADDR);
                }
            }
            catch (Exception e)
            {
                if (e.InnerException != null)

{Console.WriteLine(e.InnerException.Message);}
                else
                {Console.WriteLine(e.Message);}
            }
            finally
            {
                Console.WriteLine("Enter to exit:");
                string line = Console.ReadLine();
            }
        }
        }
}
```

3. Click the **Start** button on the Visual Studio menu bar to run the query.

Result

The sample program should run the entity query and return the following result in the console window.


file:///C:/EDM/ConsoleApplication1/ConsoleApplication1/bin/Debug/ConsoleApplication1.EXE

```
start
2=>Diana=>Morris=>431 Third Ave.
4=>Jill=>Kahn=>12 School St.
6=>Betty=>Burke=>400 Technology PathûMS10-27
3=>David=>Argonne=>75 Great Road
5=>Kenneth=>Williams=>837 Ocean Ave.
7=>Martha=>Gill=>555 Main Street
10=>Andrew=>McCaig=>999 Hill Road
8=>Steven=>Holland=>4325 Hill Road
12=>Laurie=>Patry=>10 Rustic Trail
1=>Samuel=>Smith=>10 Commerical St.
9=>Nicole=>Orlando=>820 Middlesex Turnpike
11=>Skip=>Lewis=>10 Dock Street
Enter to exit:
```

Tutorial: Developing an application using ADO.NET

This example demonstrates a simple application using ADO.NET. The application calls the HS.SALES account in UniVerse. It loads the information from the CUSTOMER file on to a DataGridView control, and allows you to make changes to the CUSTOMER file using the U2 CommandBuilder class to update the file.

Procedure

1. Open a project in Visual Studio 2010.
2. Select the programming language with which you want to work. The examples in this document are all created using C#.
3. Select **File > New Project**.
4. When the **New Project** dialog box opens, select **Windows Forms Application**.
5. In the **name** field, enter a name for the project. The project name in this example is **WindowsFormsApplication1**.
6. In the **location** field, enter the location where the project will reside. The location in this example is **C:\walkthrough**.
7. Click **OK**.

Result

The new project opens in the form designer.

The Microsoft Visual Studio Form Design window has three main panes: The form designer, the Solution Explorer, and the Properties pane. You can create and edit your application in the form designer by dragging items from the Visual Studio Toolbox onto the form. The Solution Explorer provides a navigation tree view of all the files associated with your project. The property pane allows you to set the properties of the form and of the individual objects on the form.

Next

Tutorial: Adding a reference the project

Tutorial: Adding a reference the project

You must add a reference to the U2 Data.Client.dll.

Procedure

1. In the Solution Explorer, right-click the **References** node and select **Add Reference** from the context menu.
2. Select the **Browse** tab and navigate to (on a 64-bit Windows machine):

   C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\bin\.NETFramework\v4.0
3. Select the U2.Data.Client.dll and click **OK**.

Next

Tutorial: Adding controls to the form

Tutorial: Adding controls to the form

After adding a reference to the to the topic, you can add some controls to the form.

Prerequisite

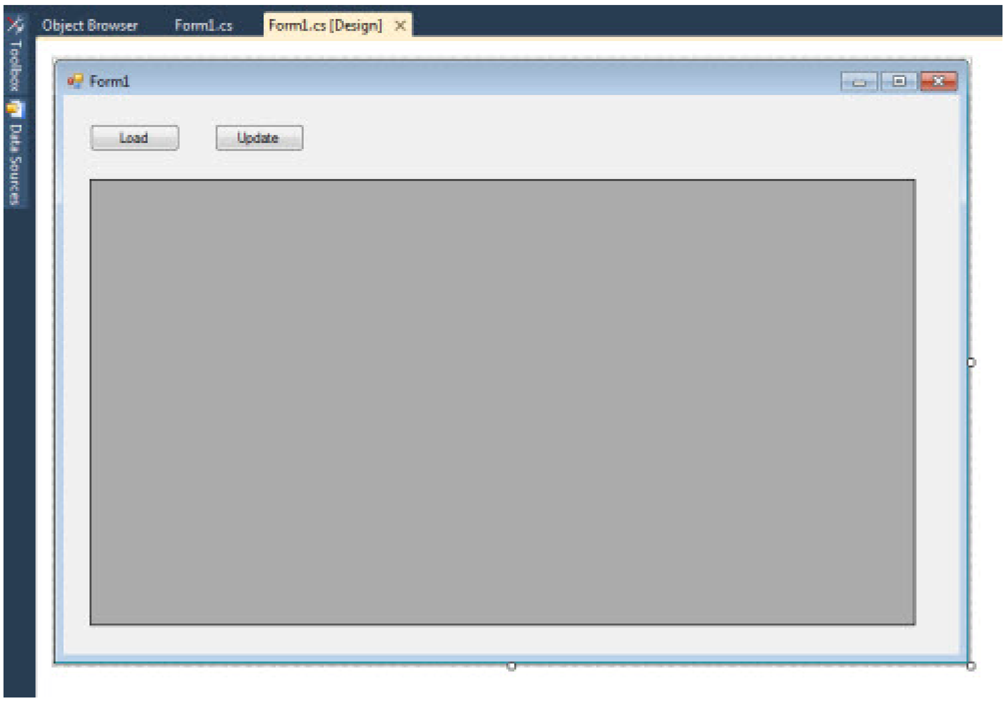
Tutorial: Adding a reference the project

Procedure

1. From the Visual Studio Toolbox, drag two buttons onto the form. In the properties window, change the button properties as follows:
    - Change the Text property of Button1 to **Load**
    - Change the Text property of Button2 to **Update**
2. From the Visual Studio Toolbox, drag a **DataGridView** control onto the form.

Result

The form should look similar to the following:



Next

Tutorial: Creating event handlers for the button controls

Tutorial: Creating event handlers for the button controls

After updating the properties, create an event handler for each button click. This requires you to add a small amount of code to the form.

Prerequisite

Tutorial: Adding controls to the form

Procedure

1. Double-click **Load** to create an event handler for the button.
2. In the code editor, add the following *using* statement to the form:
   ```
   using U2.Data.Client
   ```
3. Add the following code to the **Button1** click event:

```
{
        private DataSet m_DS = new DataSet();
        private U2DataAdapter m_DA = new U2DataAdapter();
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                U2ConnectionStringBuilder conn_str = new U2ConnectionStringBuilder();
                conn_str.UserID = "user";
                conn_str.Password = "pass";
                conn_str.Server = "server";
                conn_str.Database = "HS.SALES";
                conn_str.ServerType = "UNIVERSE";
                conn_str.Pooling = false;
                string s = conn_str.ToString();
                U2Connection con = new U2Connection();
                con.ConnectionString = s;
                con.Open();
                Console.WriteLine("Connected.........................");
                U2Command cmd = con.CreateCommand();
                cmd.CommandText = "SELECT * FROM CUSTOMER";
                m_DA.SelectCommand = cmd;
                m_DA.Fill(m_DS);
                DataTable dt = m_DS.Tables[0];
                this.dataGridView1.DataSource = dt;
                con.Close();
            }
            catch (Exception e2)
            {
                MessageBox.Show(e2.Message);
            }
        }
}
```

Note

Edit the login credentials required for the server connection.

4. Return focus to the Form designer and then double-click the **Update** button.
5. Add the following code to the **Button2** click event:

```
private void button2_Click(object sender, EventArgs e)
        {
            try
            {
                if (m_DS.HasChanges() == false)
                {
                    // Nothing to do
                    return;
                }
                DataTable modifiedTable = m_DS.Tables[0].GetChanges(DataRowState.Modified);
                U2CommandBuilder d = new U2CommandBuilder(m_DA);
                U2Command lUpCmd = d.GetUpdateCommand();
                m_DA.Update(modifiedTable);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

Result

The Button1 event handler creates a connection the to the HS.SALES database and then loads the information from the CUSTOMER file into the data grid.

The Button2 event handler uses the U2 CommandBuilder class to generate an update statement. It checks to see whether any changes have been made to the record(s). If it detects any changes, it first checks to see how many changes have been made and then repopulates the DataGrid using the generated Update statement.

Next

Building the application

Tutorial: Building the application

After creating all of the controls and event handlers, the application is ready to build.

Prerequisite

Tutorial: Creating event handlers for the button controls

Procedure

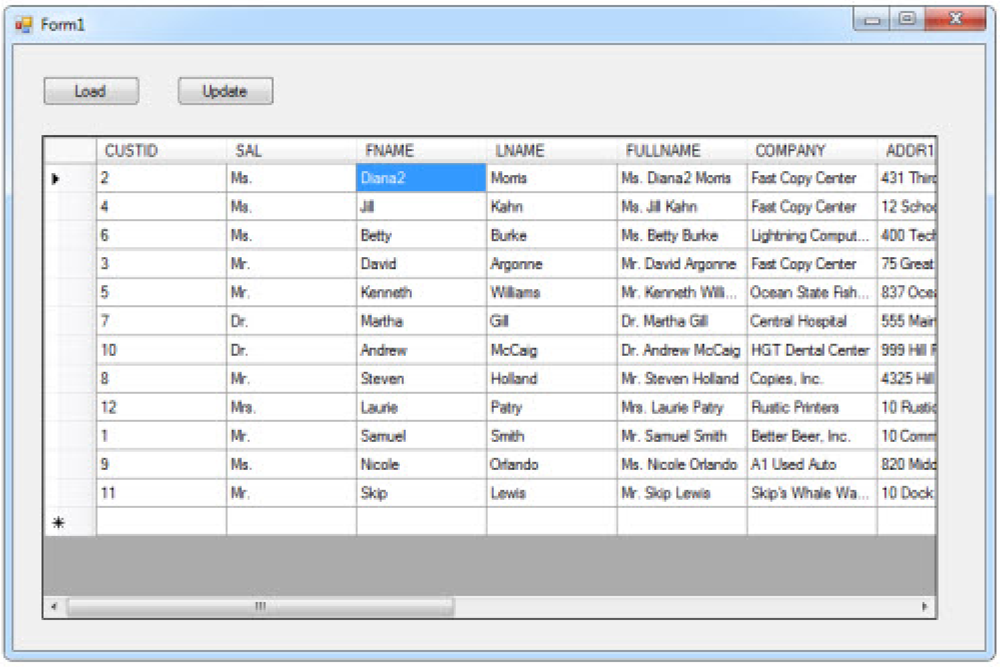
1. Select **Build > Build Solution** from the Visual Studio toolbar.
2. Run the application. To do this, click **Debug** on the Visual Studio toolbar, and then select the **Start Without Debugging** option. This opens a working version of the application you just created.

   Tip: If you want to debug the application, you can choose to **Start Debugging** instead.

3. Click **Load**. The customer information populates in the GridView, as shown:
4. In the grid, locate the name **Diana** in the FNAME field of the CUSTOMER table. Click in the cell to edit the name, and change it to **Diana2**. Click **Update**.

Result

The change is saved to your account, as shown:



	CUSTID	SAL	FNAME	LNAME	FULLNAME	COMPANY	ADDR1
▶	2	Ms.	Diana2	Morris	Ms. Diana2 Morris	Fast Copy Center	431 Thir
	4	Ms.	Jill	Kahn	Ms. Jill Kahn	Fast Copy Center	12 Scho
	6	Ms.	Betty	Burke	Ms. Betty Burke	Lightning Comput...	400 Tec
	3	Mr.	David	Argonne	Mr. David Argonne	Fast Copy Center	75 Great
	5	Mr.	Kenneth	Williams	Mr. Kenneth Will...	Ocean State Fish...	837 Oce
	7	Dr.	Martha	Gill	Dr. Martha Gill	Central Hospital	555 Mai
	10	Dr.	Andrew	McCaig	Dr. Andrew McCaig	HGT Dental Center	999 Hill
	8	Mr.	Steven	Holland	Mr. Steven Holland	Copies, Inc.	4325 Hill
	12	Mrs.	Laurie	Patry	Mrs. Laurie Patry	Rustic Printers	10 Rustic
	1	Mr.	Samuel	Smith	Mr. Samuel Smith	Better Beer, Inc.	10 Comm
	9	Ms.	Nicole	Orlando	Ms. Nicole Orlando	A1 Used Auto	820 Midd
	11	Mr.	Skip	Lewis	Mr. Skip Lewis	Skip's Whale Wa...	10 Dock
*							

Tutorial: Developing an application using UniObjects

This example demonstrates a simple application using UniObjects. The application calls the HS.SALES account in UniVerse. It loads the information from the CUSTOMER file on to a DataGridView control, and allows you to make changes to the CUSTOMER file using the U2 CommandBuilder class to update the file.

Procedure

1. Open Visual Studio 2010.
2. Select the programming language with which you want to work. The examples in this document are all created using C#.
3. Select **File > New Project**.
4. When the **New Project** dialog box opens, select **Windows Forms Application**.
5. In the **name** field, enter a name for the project. The project name in this example is **WindowsFormsApplication_UO**.
6. In the **location** field, enter the location where the project will reside. The location in this example is **C:\walkthrough**.
7. Click **OK**.

Result

The new project opens in the Form designer.

The Microsoft Visual Studio Form Design window has three main panes: The form designer, the Solution Explorer, and the Properties pane. You can create and edit your application in the form designer by dragging items from the Visual Studio Toolbox onto the form. The Solution Explorer provides a navigation tree view of all the files associated with your project. The property pane allows you to set the properties of the form and of the individual objects on the form.

Next

Tutorial: Adding a reference to a project

Tutorial: Adding a reference to a project

You must add a reference to the U2 Data.Client.dll.

Procedure

1. In the Solution Explorer, right-click the **References** node and select **Add Reference** from the context menu.
2. Select the **Browse** tab and navigate to (on a 64-bit Windows machine):

   C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\bin\.NETFramework\v4.0

3. Select the U2.Data.Client.dll and click **OK**.

Next

Tutorial: Adding controls to the form

Tutorial: Adding controls to the form

After adding a reference to the project, you can add controls to the form.

Prerequisite

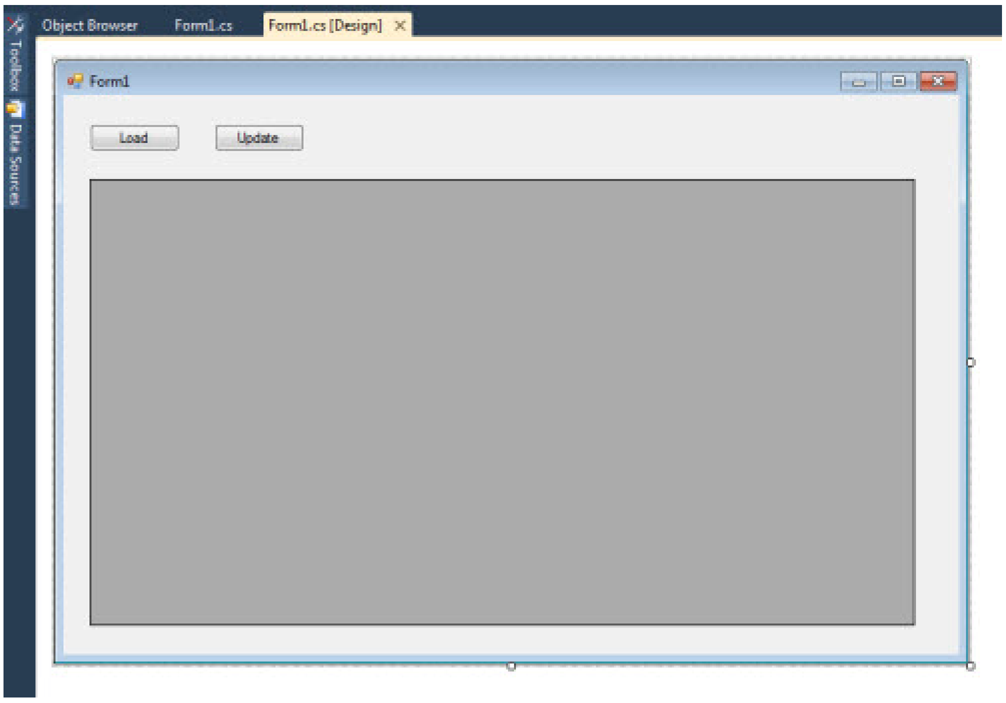
Tutorial: Adding a reference to a project

Procedure

1. From the Visual Studio Toolbox, drag two buttons onto the form. In the properties window, change the button properties as follows:
   - Change the Text property of Button1 to **Load**
   - Change the Text property of Button2 to **Update**
2. From the Visual Studio Toolbox, drag a **DataGridView** control onto the form.

Result

The form should look similar to the following:



Next

Tutorial: Creating event handlers for the button controls

Tutorial: Creating event handlers for the button controls

After updating the properties, create an event handler for each button click. This requires you to add a small amount of code to the form.

Prerequisite

Tutorial: Adding controls to the form

Procedure

1. Double-click **Load** to create an event handler for the button.
2. In the code editor, add the following *using* statement to the form:

   ```
   using U2.Data.Client
   ```

3. Add the following code to the **Button1** click event:

```
{
 public partial class Form1 : Form
 {
     private DataSet m_DS = new DataSet();
     private U2DataAdapter m_DA = new U2DataAdapter();
     public Form1()
     {
         InitializeComponent();
     }
     private void button1_Click(object sender, EventArgs e)
     {
         try
         {
             U2Connection con = GetConnection();
             Console.WriteLine("Connected.........................");
             // get RECID
             UniSession us1 = con.UniSession;
             UniSelectList sl = us1.CreateUniSelectList(2);
             // Select UniFile
             UniFile fl = us1.CreateUniFile("CUSTOMER");
             sl.Select(fl);
             bool lLastRecord = sl.LastRecordRead;
             List lRecIdList = new List();
             while (!lLastRecord)
             {
                 string sRecID = sl.Next();
                 lRecIdList.Add(sRecID);
                 Console.WriteLine("Record ID:" + sRecID);
                 lLastRecord = sl.LastRecordRead;
             }
             UniDataSet uSet = fl.ReadRecords(lRecIdList.ToArray());
             // we just create simple table
             DataTable dt = new DataTable();
             dt.Columns.Add("CUSTID", typeof(int));
             dt.Columns.Add("SAL", typeof(string));
             dt.Columns.Add("FNAME", typeof(string));
             dt.Columns.Add("LNAME", typeof(string));
             // use for each statement to print the record
             foreach (UniRecord item in uSet)
```

```
            {
                try
                {
                    UniDynArray dr = item.Record;

                    DataRow lDataRow = dt.NewRow();
                    lDataRow["CUSTID"] = Convert.ToInt32(item.RecordID);
                    lDataRow["SAL"] = dr.Extract(1).StringValue;
                    lDataRow["FNAME"] = dr.Extract(2).StringValue;
                    lDataRow["LNAME"] = dr.Extract(3).StringValue;

                    dt.Rows.Add(lDataRow);
                }
                catch (Exception)
                {


                }

            }
            dt.AcceptChanges();
            m_DS.Tables.Add(dt);
            this.dataGridView1.DataSource = dt;

            con.Close();
        }
        catch (Exception e4)
        {
            Console.WriteLine(e4.Message);
        }
        finally
        {
            Console.WriteLine("Enter to exit:");
            string line = Console.ReadLine();
        }

    }
              private U2Connection GetConnection()
    {
        U2ConnectionStringBuilder conn_str = new U2ConnectionStringBuilder();
        conn_str.UserID = "user";
        conn_str.Password = "password";
        conn_str.Server = "server";
        conn_str.Database = "HS.SALES";
        conn_str.ServerType = "UNIVERSE";
        conn_str.AccessMode = "Native"; // FOR UO
        conn_str.RpcServiceType = "uvcs"; // FOR UO
        conn_str.Pooling = false;
        string s = conn_str.ToString();
        U2Connection con = new U2Connection();
        con.ConnectionString = s;
        con.Open();
        return con;
    }
  }
```

Note

Edit the login credentials required for the server connection.

4. Return focus to the Form designer and then double-click the **Update** button.
5. Add the following code to the **Button2** click event:

```
private void button2_Click(object sender, EventArgs e)
  {
```

```
            try
            {
                DataTable modifiedTable = m_DS.Tables[0].GetChanges(DataRowState.Modified);
                if (modifiedTable.Rows.Count > 0)
                {
                    U2Connection con = GetConnection();
                    Console.WriteLine("Connected.........................");
                    // get RECID
                    UniSession lUniSession = con.UniSession;
                    // Select UniFile
                    UniFile fl = lUniSession.CreateUniFile("CUSTOMER");
                    UniDynArray udr3 = new UniDynArray(lUniSession);
                    foreach (DataRow item in modifiedTable.Rows)
                    {
                        udr3.Insert(1, -1, (string)item["FNAME"]);
                    }
                    string[] lFields = { "FNAME" };
                    string lRecID = modifiedTable.Rows[0][0].ToString();
                    fl.WriteNamedFields(lRecID, lFields, udr3);
                    con.Close();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
```

Result

The Button1 event handler creates a connection the to the HS.SALES database and then loads the information from the CUSTOMER file into the data grid.

The Button2 event handler uses the U2 CommandBuilder class to generate an update statement. It checks to see whether any changes have been made to the record(s). If it detects any changes, it first checks to see how many changes have been made and then repopulates the DataGrid using the generated Update statement.

Next

Tutorial: Building the application

Tutorial: Building the application

After creating all of the controls and event handlers, the application is ready to build.

Prerequisite

Tutorial: Creating event handlers for the button controls

Procedure

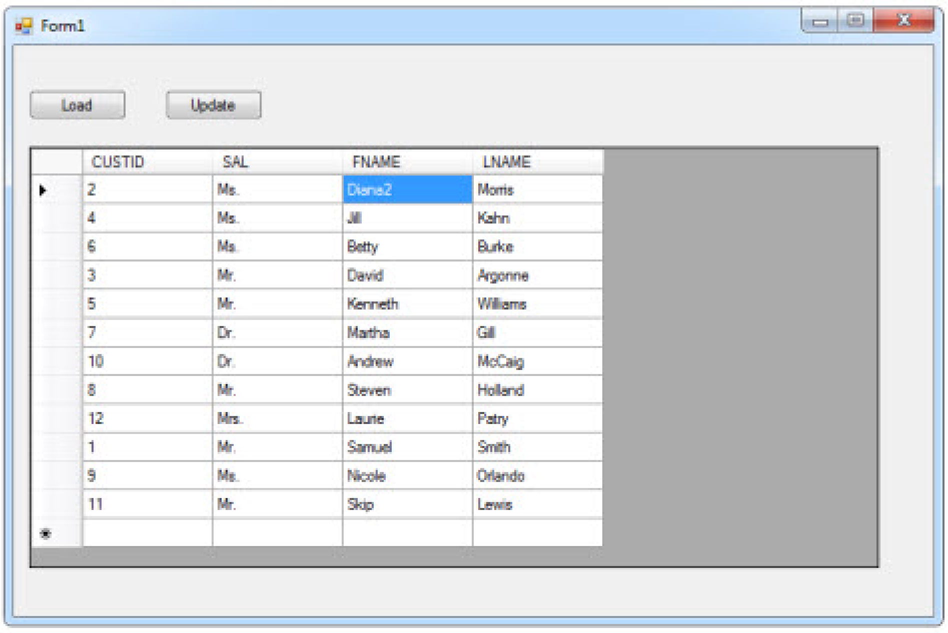
1. Select **Build** > **Build Solution** from the Visual Studio toolbar.
2. Run the application. To do this, click **Debug** on the Visual Studio toolbar, and then select the **Start Without Debugging** option. This opens a working version of the application you just created.

   Tip: If you want to debug the application, you can choose to **Start Debugging** instead.

3. Click **Load**. The customer information populates in the GridView.
4. In the grid, locate the word **Diana** in the FNAME field in the CUSTOMER table. Click in the cell to edit the name, and change it to **Diana2**. Click **Update**.

Result

The change is saved to your account, as shown:



CUSTID	SAL	FNAME	LNAME
2	Ms.	Diana2	Morris
4	Ms.	Jill	Kahn
6	Ms.	Betty	Burke
3	Mr.	David	Argonne
5	Mr.	Kenneth	Williams
7	Dr.	Martha	Gill
10	Dr.	Andrew	McCaig
8	Mr.	Steven	Holland
12	Mrs.	Laurie	Patry
1	Mr.	Samuel	Smith
9	Ms.	Nicole	Orlando
11	Mr.	Skip	Lewis

Tutorial: Developing an application using ADO.NET and UniObjects

This example demonstrates a simple application using both ADO.NET and UniObjects. The application calls the HS.SALES account in UniVerse. It loads the information from the CUSTOMER file on to a DataGridView control, and allows you to make changes to the CUSTOMER file using the U2 CommandBuilder class to update the file.

Procedure

Open a project in Visual Studio 2010

1. Open Visual Studio 2010.
2. Select the programming language with which you want to work. The examples in this document are all created using C#.
3. Select **File > New Project**.
4. When the **New Project** dialog box opens, select **Windows Forms Application**.
5. In the name field, enter a name for the project. The project name in this example is **WindowsFormsApplication_ADO_UO**.
6. In the location field, enter the location where the project will reside. The location in this example is **C:\walkthrough**.
7. Click **OK**.

Result

The new project opens in the Form designer.

The Microsoft Visual Studio Form Design window has three main panes: The form designer, the Solution Explorer, and the Properties pane. You can create and edit your application in the form designer by dragging items from the Visual Studio Toolbox onto the form. The Solution Explorer provides a navigation tree view of all the files associated with your project. The property pane allows you to set the properties of the form and of the individual objects on the form.

Next

Tutorial: Adding a reference to the application

Tutorial: Adding a reference to the application

You must add a reference to the U2 Data.Client.dll.

Procedure

1. In the Solution Explorer, right-click the **References** node and select **Add Reference** from the context menu.
2. Select the **Browse** tab and navigate to (on a 64-bit Windows machine): :

   C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\bin\.NETFramework\v4.0

3. Select the U2.Data.Client.dll and click **OK**.

Next

Tutorial: Adding controls to the form

Tutorial: Adding controls to the form

After adding the references, it is time to add some controls to the form.

Prerequisite

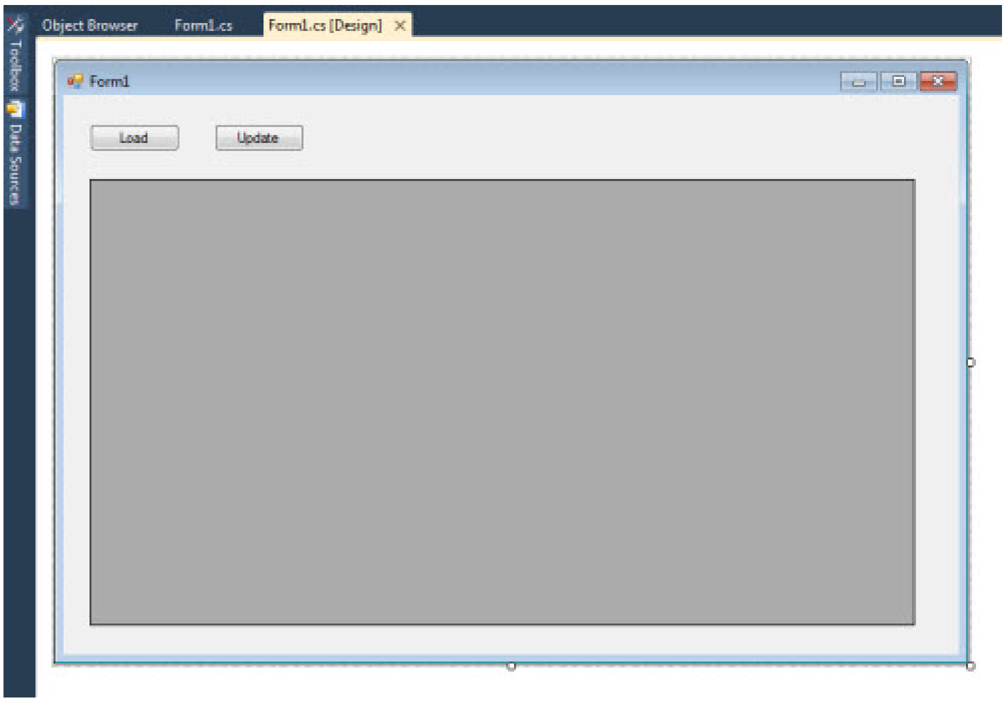
Adding a reference to the application

Procedure

1. From the Visual Studio Toolbox, drag two buttons onto the form. In the properties window, change the button properties as follows:
    - Change the Text property of Button1 to **Load**
    - Change the Text property of Button2 to **Update**
2. From the Visual Studio Toolbox, drag a **DataGridView** control onto the form.

Result

The form should look similar to the following:



Next

Tutorial: Building the application

Tutorial: Building the application

After creating all of the controls and event handlers, the application is ready to build.

Prerequisite

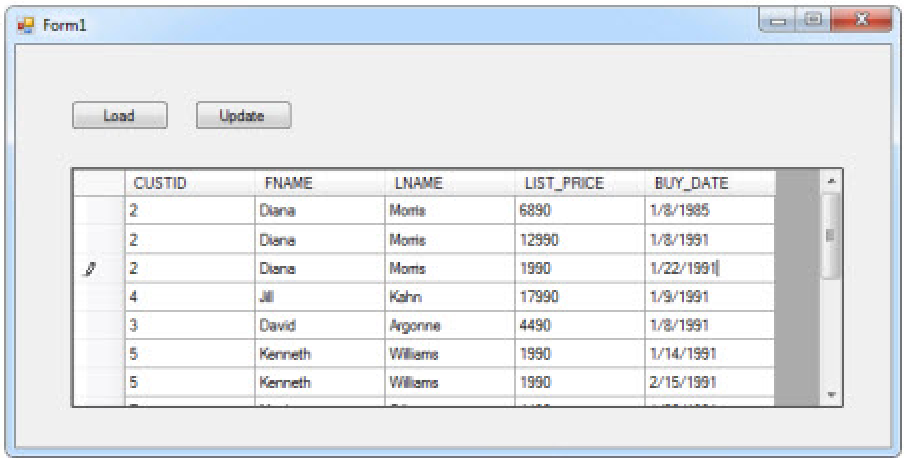
Tutorial: Creating event handlers for the button controls (on-line documentation)

Procedure

1. Select **Build > Build Solution** from the Visual Studio toolbar.
2. Run the application. To do this, click **Debug** on the Visual Studio toolbar, and then select the **Start Without Debugging** option. This opens a working version of the application you just created.
3. Click **Load**. The customer information populates in the GridView.
4. In the grid, locate the BUY_DATE fields for record **2**. Click in the **1/8/1991** cell to edit the date, and change it to **1/8/1995**. Click in another cell to save the change and then select **Update**.

Result

The change is saved to your account, as shown:



CUSTID	FNAME	LNAME	LIST_PRICE	BUY_DATE
2	Diana	Morris	6890	1/8/1985
2	Diana	Morris	12990	1/8/1991
2	Diana	Morris	1990	1/22/1991
4	Jill	Kahn	17990	1/9/1991
3	David	Argonne	4490	1/8/1991
5	Kenneth	Williams	1990	1/14/1991
5	Kenneth	Williams	1990	2/15/1991

U2 Toolkit for .NET Developer

U2 Toolkit for .NET Developer is a Data Designer Extensibility (DDEX) provider. It allows Microsoft Visual Studio components such as the Server Explorer and the data designers to work with the U2 databases. The U2 Server Objects, such as tables, views, and subroutines, are fully integrated within Visual Studio. U2 Toolkit for .NET Developer allows designers to use the drag-and-drop capabilities and code generation (C#/VB.NET) found within Visual Studio to create new U2 applications without the need for extra programming. U2 Toolkit for .NET Developer is designed to present a simple interface to U2 databases. For example, you can access and manage U2 Connections in Visual Studio Server Explorer, view server-side object properties, retrieve and update data from tables and views, and Generate ADO .NET and EDM code using the drag-and-drop technique.

Note

UniData users must normalize their **demo** account for use with SQL. This can be done using the U2 Metadata Manager (U2 MDM). Refer to the U2 MDM Help for information on how to easily use this tool to normalize your data.

Alternatively, Visual Schema Generator (VSG) can also be used to normalize files. If using VSG, you must create a sub-table from the CUSTOMER file in the demo account and include all fields. Ensure the privileges are set to PUBLIC.

Note

UniVerse tables and views are always accessible to ODBC applications, but UniVerse files that are not tables are not. To make UniVerse files accessible to ODBC applications, you must run the ODBC file access utility in the account. Among other things, this utility creates the HS_FILE_ACCESS file, which lists all UniVerse files referenced by F- and Q-pointers in the VOC file. You can edit this file to define exactly which files should be ODBC-accessible. You probably also need to run the HS.UPDATE.FILEINFO program from time to time, which updates the account’s file information cache. Refer to the *UniVerse ODBC Guide* for more information about making UniVerse files accessible.

U2 Toolkit for .NET Developer system requirements

The following requirements must be met in order for the U2 Toolkit for .NET Developer to work correctly on your system.

System requirements

- Microsoft Windows 7 (32-bit/64-bit), Windows 8 (32-bit/64-bit), Windows 8.1 (32-bit/64-bit), and Windows 2012
- Microsoft.NET Framework 4.0, 4.5
- Microsoft Visual Studio 2010, 2012, 2013
- ADO.NET Entity Framework 4, 4.1, 4.2, 5

Supported versions of UniData and UniVerse

- UniData 7.1 or later
- UniVerse 10.3 or later

SSIS/SSRS (Optional)

	.NET Framework 2.0	.NET Framework 4.0	Visual Studio 2008 BIDS	Visual Studio 2010 BIDS	Visual Studio 2012 BIDS	Visual Studio 2013 BIDS
SQL Access	Yes	Yes	Yes	Yes	Yes	Yes
Native Visual Studio Integration	No	Yes	No	Yes	Yes	Yes

Installing U2 Toolkit for .NET Developer

Complete the following steps to install U2 Toolkit for .NET Developer on Windows.

Prerequisites

- **U2 Toolkit for .NET Provider system requirements**
- The U2 Toolkit for .NET Developer in Visual Studio IDE requires a valid license. There is no charge for a Developer license. To obtain a copy of U2 Toolkit for .NET Developer, contact the Rocket Business Connects team at https://u2tc.rocketsoftware.com/main.asp?js=y.

Procedure

1. From the U2 Toolkit for .NET installation screen, select **Install U2 Toolkit for .NET Developer**. Click **Next**.
2. After accepting the licensing agreement, click **Next**.
3. By default, the installation process installs U2 Toolkit for .NET Developer in the following directories:
   - The installation path on a 64-bit Windows 7 machine is C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Toolkit for .NET Developer
   - The installation path on a 64-bit Windows 7 machine is C:\Program Files\Rocket Software\U2 Toolkit for .NET\U2 Toolkit for .NET Developer
   - The installation path on a Windows 7/Windows XP machine is C:\Program Files\Rocket Software\U2 Toolkit for .NET\U2 Toolkit for .NET Developer
4. Click **Next** to install U2 Toolkit for .NET in the default folder, or click **Browse** to search for a different folder.
5. By default, the installation process adds the Rocket U2 icon to the Program Folders list. Click **Next** to accept this default, or select a different folder from the Existing Folders list and then click **Next**.
6. Click **Close** to complete the installation process.

Next step

Authorizing U2 Toolkit for .NET Developer licenses

Authorizing U2 Toolkit for .NET Developer licenses

The U2 Toolkit for .NET Developer in Visual Studio IDE, requires a license. This license is per machine. The License Manager controls validity of the license.

To obtain a license, you must generate a configuration code from within the tool. After generating this code, request an authorization code from the U2BusinessConnect site. The license can then be authorized in the tool using the authorization code.

The U2BusinessConnect site can be accessed from the following links:

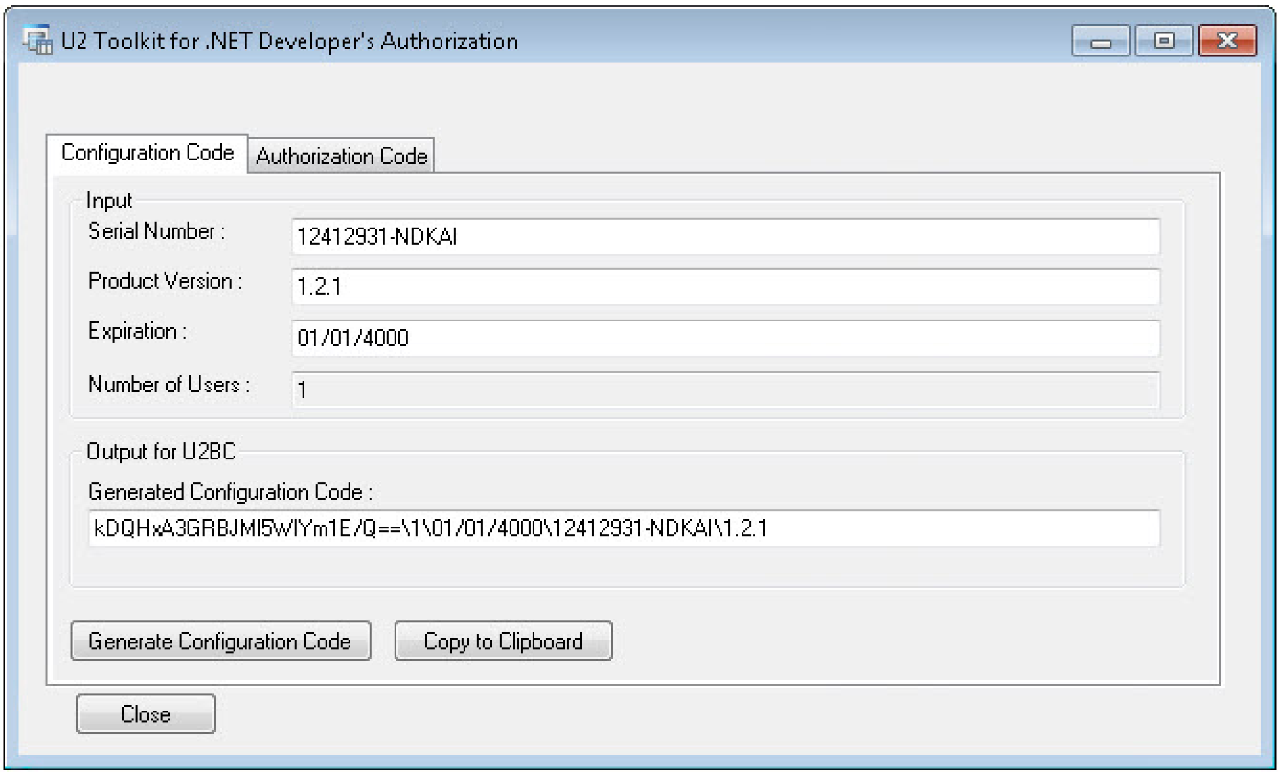
US: https://u2tc.rocketsoftware.com/authprod.asp
International: https://u2tcint.rocketsoftware.com/authprod.asp

Generating a configuration code

U2 Toolkit for .NET Developer requires a valid license for development. A configuration code is needed when obtaining the authorization code from the U2BusinessConnect site.

Procedure

1. Open the authorization tool by navigating to **Start** > **Rocket U2** > **U2 Toolkit for .NET** > **U2 Toolkit for .NET Developer** > **Authorization** > **Authorization Tool**. Select the **Configuration Code** tab.



Serial Number : 12412931-NDKAI
Product Version : 1.2.1
Expiration : 01/01/4000
Number of Users : 1
Generated Configuration Code :
kDQHxA3GRBJMI5WlYm1E/Q==\1\01/01/4000\12412931-NDKAI\1.2.1

2. Enter the required information and then click **Generate Configuration Code**. A generated configuration code displays. Click **Copy to Clipboard** to copy the code.
3. At this point, you must obtain an authorization code from U2BusinessConnect, using the configuration code. To obtain the authorization code, visit the appropriate URL below and click **Authorize Products**.

   US: https://u2tc.rocketsoftware.com/authprod.asp
   International: https://u2tcint.rocketsoftware.com/authprod.asp

Note

If user access control is enabled, licensing must be done in administration mode.

Next

Authorizing a license

Authorizing a license

After obtaining an authorization code from the U2BusinessConnect site, the U2 Toolkit for .NET license must be authorized.

Prerequisite

Generating a configuration code

Procedure

1. Open the authorization tool by navigating to **Start** > **Rocket Software** > **U2 Toolkit for .NET** > **U2 Toolkit for .NET Developer** > **Authorization** > **Authorization**. Select the **Authorization Code** tab.
2. Paste the authorization code obtained from the U2BusinessConnect site into the **Authorization Code** box. If the authorization code is on the clipboard, click **Copy from Clipboard** to paste the code into the authorization field.
3. Click **Authorize**.

Establishing a server connection using the Visual Studio Server Explorer

To work with UniData or UniVerse accounts and data, you must enable your computer to connect to the server on which the accounts and data reside. Your computer (the client) requires a U2 server definition to make a connection with the server.

A U2 server is a defined connection to a server computer on which U2 accounts and data are stored. All existing U2 servers on the client computer are listed in the U2 Resource view. The U2 server definition is stored on the client computer on which it was created, and is not shared across a network. One or several users can create multiple U2 server definitions on the same client computer.

Creating U2 server definitions

To administer UniData or UniVerse accounts and data, create a U2 server definition that enables the computer to connect to the U2 data server on which the accounts and data are stored.

Procedure

1. Start Microsoft Visual Studio.
2. Select **View > Server Explorer** from the Visual Studio menu. The Server Explorer displays the server resources in a treeview, which includes the following top-level nodes: Data Connections and Servers.
3. In the Server Explorer, right-click the **Data Connections** node, and then click **Add connection**.
4. Select the **General** tab.
5. In the **Server name** field, enter a unique name to identify the U2 server definition. The name cannot contain a slash (/) or backslash (\) character.
6. In the **Login details** fields, enter the correct login credentials.
7. In the Account name field, enter the name of the account to which you are connecting.

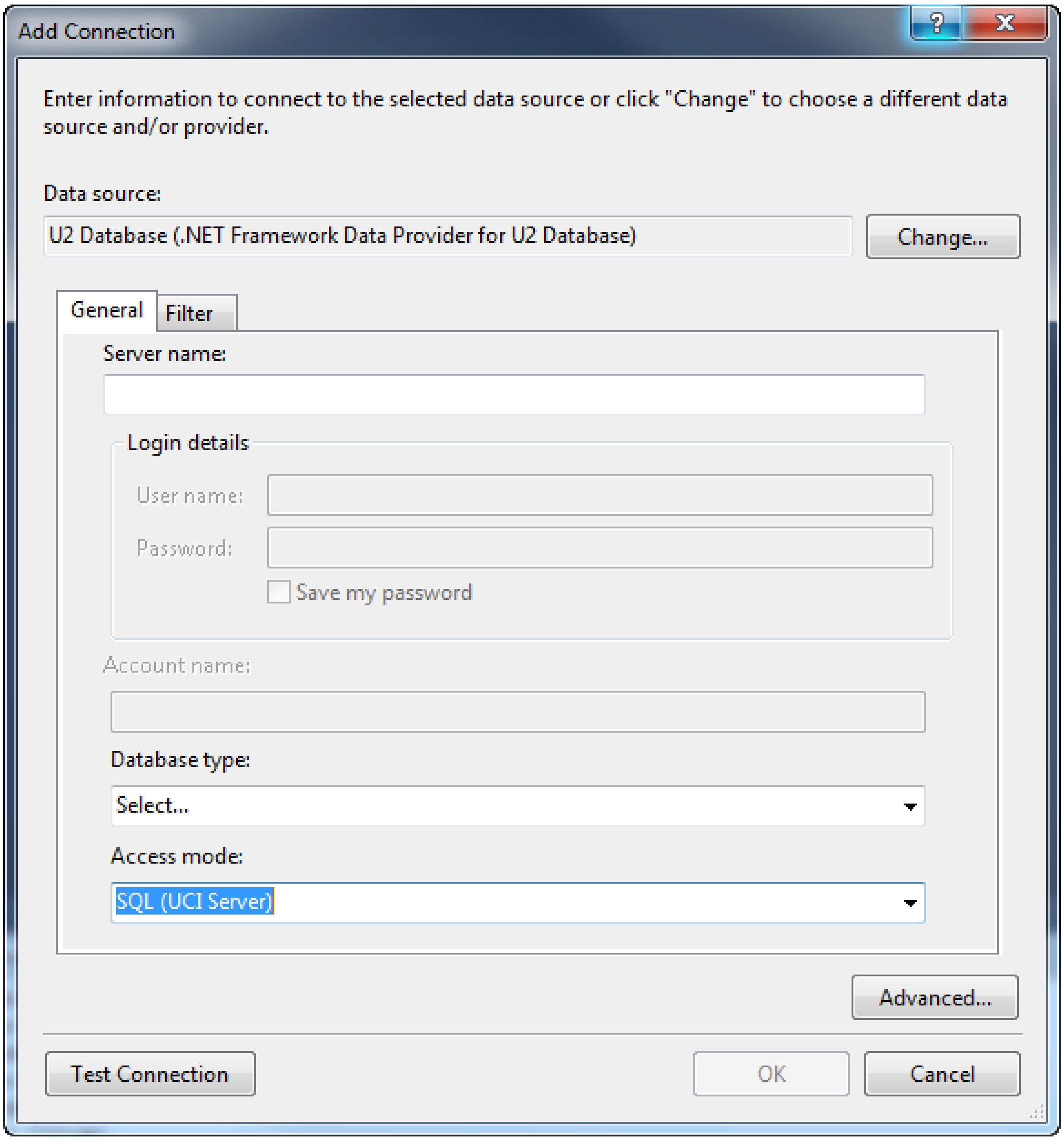
   NOTE: The database name can be set to full path or a UniData database name or a UniVerse account name. When connecting to a UniData server using the database name, the name must be defined in both the ud_database file and the system UD.ACCOUNT file.

8. From the Database type field options, select **UniData** or **UniVerse**.
9. In the **Access mode** field, select the access mode. In the current release, Native (UO Server) is not supported.
10. Optional. To view or edit the table subroutine, views, or schema filters, click the **Filters** tab and modify the information.
11. Optional. To view or edit the protocol, port number, and other advanced settings defining the connection, click **Advanced**. Make changes to any of the fields in the following categories
    - Data
    - Initialization
    - Misc
    - Pooling
    - Schema keywords
    - Security
    - Source

    See **Viewing the advanced settings of a U2 server definition** to view details about each of these properties.

12. To save the U2 server definition, click **OK**.

Example


Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:

U2 Database (.NET Framework Data Provider for U2 Database)

Server name:

Login details

User name:

Password:

Save my password

Account name:

Database type: Select...

Access mode: SQL (UCI Server)

Result

U2 Toolkit for .NET creates a directory for the U2 server, registering the server definition so the tool can find it in future sessions. The name of the new U2 server is added to the list in the Server Explorer.

Viewing the advanced settings of a U2 server definition

On the advanced settings page of the server definition, you can view or edit the protocol, port number, and other advanced settings that define the connection. You can also specify commands to run when you connect to the U2 server.

Prerequisite

Establishing a server connection using the Visual Studio Server Explorer

Procedure

1. Click **Advanced** on the Add Connections page.
2. Make changes to any of the fields in the following categories:

Property	Description	Remarks
Data		
ConnectionString ('ConnectionString Property' in the on-line documentation)	The connection string used to connect to the data source.	The connection string that includes settings, such as the database name, needed to establish the initial connection. The ConnectionString can be set only when the connection is closed.
Initialization		
AccessMode ('AccessMode Property' in the on-line documentation)	This property gets or sets the AccessMode to be used when connecting to a U2 database.	For SQL Access, the AccessMode=**Uci**. For Native Access, the AccessMode=**Native**. The default value is **Uci**.
AppendParameterMarker ('AppendParameterMarker Property' in the on-line documentation)	This property allows you to append append the "@" character in the parameter name. For Example, if parameter name is 'p1' then it changes to "@p1" if this property is set **True**.	This property is useful if the generated code ignores the "@" variable for the parameter. The default value is **false**.
CacheMetaDataOnConnection ('CacheMetaDataOnConnection Property' in the on-line documentation)	This property gets or sets a value indicating whether metadata, such as schema or tables, should be cached when connecting to a U2 database.	This property returns a value of **true** if the metadata is cached when a connection is made to U2 database; otherwise, a value of **false** is returned. The default value is **true**.
CodePage ('CodePage Property' in the on-line documentation)	This property gets or sets the code page identifier of the current encoding.	This property is equivalent to the Encoding.CodePage property.
Connect Timeout ('Connect_Timeout Property' in the on-line	This property is used to set the amount of time	You can set the amount of time a connection waits to time out by

Property	Description	Remarks
documentation)	between when a request is sent and when the request terminates and throws a Timeout error.	using the ConnectTimeout or Connection Timeout keywords in the connection string. A value of 0 indicates no limit, and should be avoided in a ConnectionString because an attempt to connect waits indefinitely.
ExpandMultiValueRows ('ExpandMultiValueRows Property' in the on-line documentation)	This property gets or sets a value that indicates whether to expand multivalued rows.	This property returns a value of **true** if multivalued rows are expanded; otherwise, a value of **false** is returned. The default value is **false**.
MAXFETCHBUFF ('MaxFetchBuff Property' in the on-line documentation)	This property gets and sets the maximum buffer size. It controls the maximum buffer size on the server to hold data rows.	The server usually fills this buffer with as many rows as possible before sending data to the client. If any single row exceeds the length of MAXFETCHBUFF, SQLFetch fails and you should increase the value of this parameter.
MAXFETCHCOLS ('MaxFetchCols Property' in the on-line documentation)	This property gets and sets the maximum number of columns. It controls the maximum number of column values the server can put in the buffer before sending data to the client.	The maximum number of columns.
NamedParameters ('NamedParameters Property' in the on-line documentation)	This parameter gets or sets a value that indicates whether to treat arguments as named parameters. For example, if set to **true**, then the argument @ARG is treated as a function parameter. If set to **false**, then @ARG will not be treated as a function parameter.	Named arguments improve the readability of the code by identifying what each argument represents.
NativeFetch ('NativeFetch Property' in the on-line documentation)	This property is used to get data in chunks in order to improve performance.	The NativeFetch property should be used when fetching a large number of record IDs. For example, if there are 50,000 record IDs and NativeFetch=10000 is specified, the fetch will occur five times for performance gains.
QueryTimeout ('QueryTimeout Property' in the on-line documentation)	This property gets or sets the QueryTimeout keyword value, a value specifying the default number of seconds	The functionality of QueryTimeout Property and ConnectionTimeOut property is the same. It allows users to Get or Set the wait time before

Property	Description	Remarks
	to wait for an SQL query to execute. The default value is 300 seconds.	terminating the attempt to execute a command and generating an error. If it is used in a connection string, the QueryTimeout property and ConnectionTimeOut property are included. The maximum value of these two property will be used for TimeOut value.
RpcServiceType ('RpcServiceType Property' in the on-line documentation)	This property gets or sets the value of the RPC service type. By default, in for native access (UniObjects/UO.NET API or Native Visual Studio Integration), the UniVerse value is **uvcs** and the value for UniData is **udcs**. In SQL AccessMode, the UniVerse value is **uvserver** and the value for UniData is **udserver**.	A passed value (for example: uvcs or udcs) that should exist in the unirpcservices file.
SleepAfterClose ('SleepAfterClose Property' in the on-line documentation)	This property gets or sets the length of time (in milliseconds) to wait after the connection is closed.	This property is useful for closing the socket properly. The default value is **0**.
ThrowMismatchDataTypeException ('ThrowMismatchDataTypeException Property' in the on-line documentation)	This property gets or sets the Boolean value used to hand the data type exception that occurs during the fetch process. The default value is **false**.	Returns a value of **true** if 'ThrowMismatchDataTypeException' is enabled; otherwise a value of **false** is returned.
UseATSelect ('UseATSelect Property' in the on-line documentation)	Use this property to filter out dictionary items defined in the "@" or "@SELECT" phrase.	This property requires server-side dictionary modification. You must specify an "@" phrase or an "@SELECT" phrase to list the fields and attributes in a file. If both an "@" phrase and an "@SELECT" phrase are specified, the fields defined in the "@" phrase take precedence.
UseFastGetRecordID ('UseFastGetRecordID Property' in the on-line documentation)	This property allows users to get all record IDs in one server call. This value is **true** by default.	This property may help to improve performance as it gets all record IDs in one server call.
UseIPv6 ('UseIPv6 Property' in the on-line documentation)	This property allows users to connect to servers using the IPv6 protocol. This	This property requires server-side IPv6 functionality. UniVerse (11.2.4) and UniData (7.4.1) will have server-

Property	Description	Remarks
	property returns **true** if UseIPv6 is enabled; otherwise **false**. False is the default.	side IPv6 functionality. If the server supports IPv6 and the UseIPv6 property is enabled, the connection is made using the IPv6 protocol. If the server does not support IPv6 functionality and is enabled, the connection is made using the IPv4 protocol.
UseMDM ('UseMDM Property' in the on-line documentation)	This property gets or sets a Boolean value that indicates how to retrieve metadata information.	If this property is **true**, metadata information such as Tables, Views, Columns are retrieved different way. You will use this property if you normalize your account using U2 Metadata Manager.
UseSameSocketOnClose ('UseSameSocketOnClose Property' in the on-line documentation)	This property gets or sets the criteria needed to reuse the socket after it closes.	To reuse a socket after it closes, set this value to **true**. The default value is false.
Misc		
ServerType ('ServerType Property' in the on-line documentation)	This property gets or sets database type. It indicates whether the connection is to be made to a UniData or UniVerse database.	The type of the connected server. The default value is ("UNIVERSE") until the connection is opened.
Pooling		
Connection LifeTime ('ConnectionLifeTime Property' in the on-line documentation)	This property specifies how long a connection stays active in the connection pool before it is ended and then created again.	A value of zero (0) causes pooled connections to have the maximum connection timeout. The default value is 60 seconds.
Connection Reset ('ConnectionReset Property' in the on-line documentation)	This property gets or sets a Boolean value that indicates whether the connection is reset when drawn from the connection pool.	The value set by the ConnectionReset property. If no value is supplied, the default is true.
Max Pool Size ('MaxPoolSize Property' in the on-line documentation)	This property gets or sets the maximum number of connections in the connection pool.	This property corresponds to the "Max Pool Size" key within the connection string.
Min Pool Size ('MinPoolSize Property' in the on-line documentation)	This property gets or sets the minimum number of connections in the connection pool.	This property corresponds to the "Min Pool Size" key within the connection string.
Pooling ('Pooling Property' in the on-line documentation)	This property gets or sets a Boolean value indicating whether a connection pool	This property corresponds to the "Pooling" key within the connection string.

Property	Description	Remarks
	is used for the connection.	
Security		
ClientCertificatePath ('ClientCertificatePath Property' in the on-line documentation)	This property gets or sets the path to the certificate.	The path of the PKCS7 signed file from which to create the X.509 certificate.
Password ('Password Property' in the on-line documentation)	This property gets or sets the password connected to your U2 database account.	This property corresponds to the "Password" and "pwd" keys within the connection string. If Password has not been set and you retrieve the value, the return value is Empty. To reset the password for the connection string, pass null to the Item property.
PersistSecurityInfo ('PersistSecurityInfo Property' in the on-line documentation)	This property gets or sets a Boolean value that defines whether security-sensitive information is returned as part of the connection.	This property corresponds to the "Persist Security Info" and "persistsecurityinfo" keys within the connection string.
SslCheckCertificateRevocation ('SslCheckCertificateRevocation Property' in the on-line documentation)	This property gets or sets the flag that indicates whether the certificate revocation list is checked during authentication.	This property gets or sets the flag that indicates whether the certificate revocation list is checked during authentication.
SSLConnection ('SSLConnection Property' in the on-line documentation)	This property is used to create a Secure Socket Layer (SSL) connection.	The default value is **false**.
SslIgnoreCertificateNameMismatch ('SslIgnoreCertificateNameMismatch Property' in the on-line documentation)	This property gets or sets the flag that indicates whether to ignore name mismatch errors on the server certificate during authentication.	Name mismatch errors occur when the name specified in the certificate is different from the name of the server machine which provides the certificate.
SslIgnoreIncompleteCertificateChain ('SslIgnoreIncompleteCertificateChain Property' in the on-line documentation)	This property gets or sets the flag that indicates whether an incomplete chain error is ignored during authentication.	This property gets or sets the flag that indicates whether to ignore name mismatch errors on the server certificate during authentication. Name mismatch errors occur when the name specified in the certificate is different from the name of the server machine which provides the certificate.
UserID ('UserID Property' in the on-line documentation)	This property gets or sets the User ID being used to connect to the U2 database.	This property corresponds to the "User ID", "user", and "uid" keys within the connection string.
WalletID ('WalletID Property' in the	This property gets or sets	This property gets or sets

Property	Description	Remarks
on-line documentation)	encryption keys.	encryption keys.
WalletPwd ('WalletPwd Property' in the on-line documentation)	This property gets or sets the encryption password.	This property gets or sets the encryption password.
Source		
Database ('Database Property' in the on-line documentation)	This property gets or sets the name of the database associated with the connection.	You need to provide a valid database name. For example, **demo**.
Server ('Server Property' in the on-line documentation)	This property gets or sets the value that specifies the host and port number to which you want to connect.	You need to provide a valid database name. For example, **localhost**.

3. To save any changes made to the advanced setting and return to the main page, click **OK**.

Accessing U2 database files through the Dataset object model

You can use the U2 Toolkit for .NET Developer to access your data files through the Dataset model. In the Dataset object model, data files are mapped to the U2 RDMS tables and can be viewed as columns and rows.

Procedure

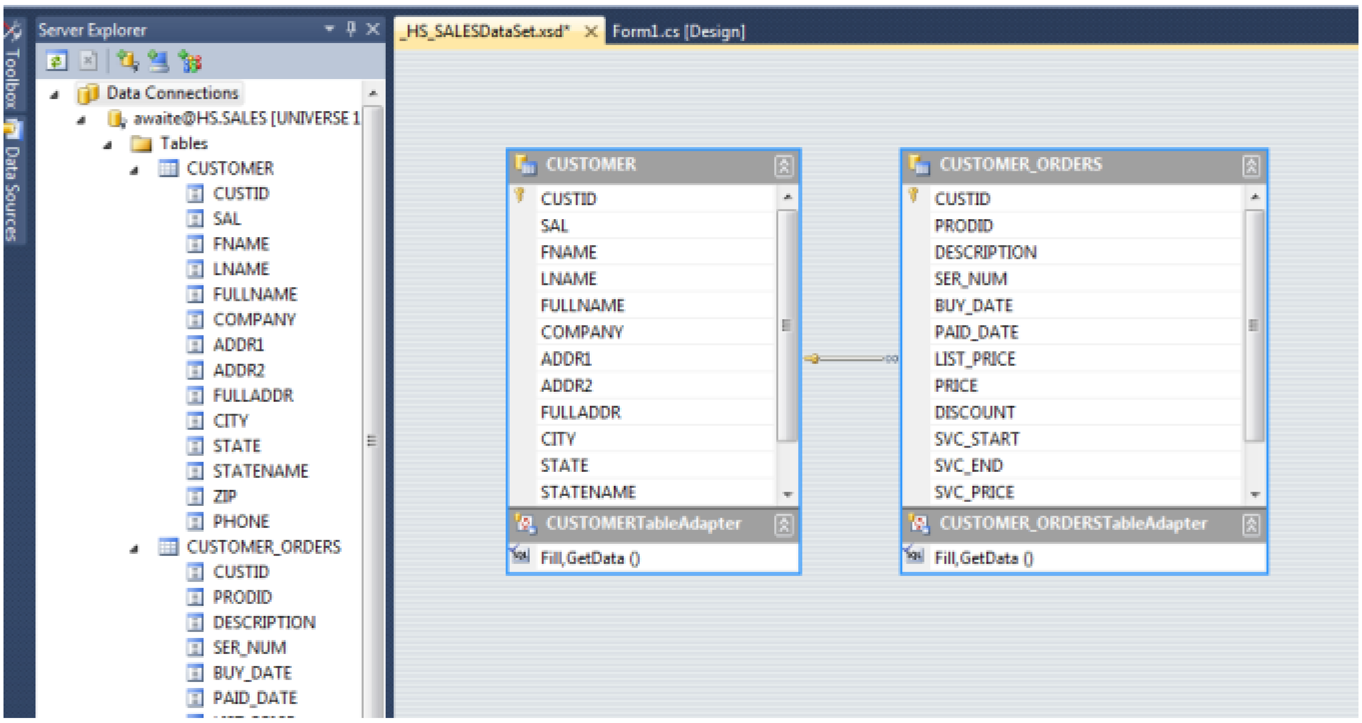
1. In Visual Studio create a new **Windows Forms Application** project and go to **Data > Show Data Sources**.

   Note: To view the data source wizard in Visual Studio 2012, select **View > Other windows > Data Sources**.
2. In the **Data Sources** window, click **Add New Data Source**.
3. Choose a **Data Source Type** from which the application will get the data. In this example, select **Database** and then click **Next.**
4. Choose a **Database Model** to determine the types of data objects your application will use. In this example, select **Dataset** and then click **Next.**
5. Choose the **Data Connection** your application will use to connect to the database. For this example, select **New Connection** to establish a data connection to the local U2 database account.
6. Enter the required information to connect to the account of your chosen database (HS.SALES for UniVerse or demo for UniData). For this example, select **SQL (UCI Server)** as the **Access Mode**.
7. Click **Test Connection** to verify this information and then click **OK.**

   Note: In the current release, **Native (UO Server)** is not supported.
8. Select the new **Data Connection** and then click **Next.**
9. Accept the default name to **Save the Connection String to the Application Configuration file** and then click **Next.**
10. Choose **Database Objects** to make them accessible in your Visual Studio project. For this example, select **CUSTOMER** and **CUSTOMER_ORDERS** from the **Tables** list and then click **Finish**.

Result

The U2 database files are now mapped to the Dataset model and are located in the project file, which is denoted by the **.xsd** extension and should look similar to the following:



You can now data-bind items from the Dataset model by dragging them from the **Data Sources** window onto any forms or existing controls in your project.

Tip

Right-click in the *.xsd window and select **Preview Data** to preview the contents of the U2 database files accessed through this Dataset model.

Accessing U2 database files through the Entity Data Model

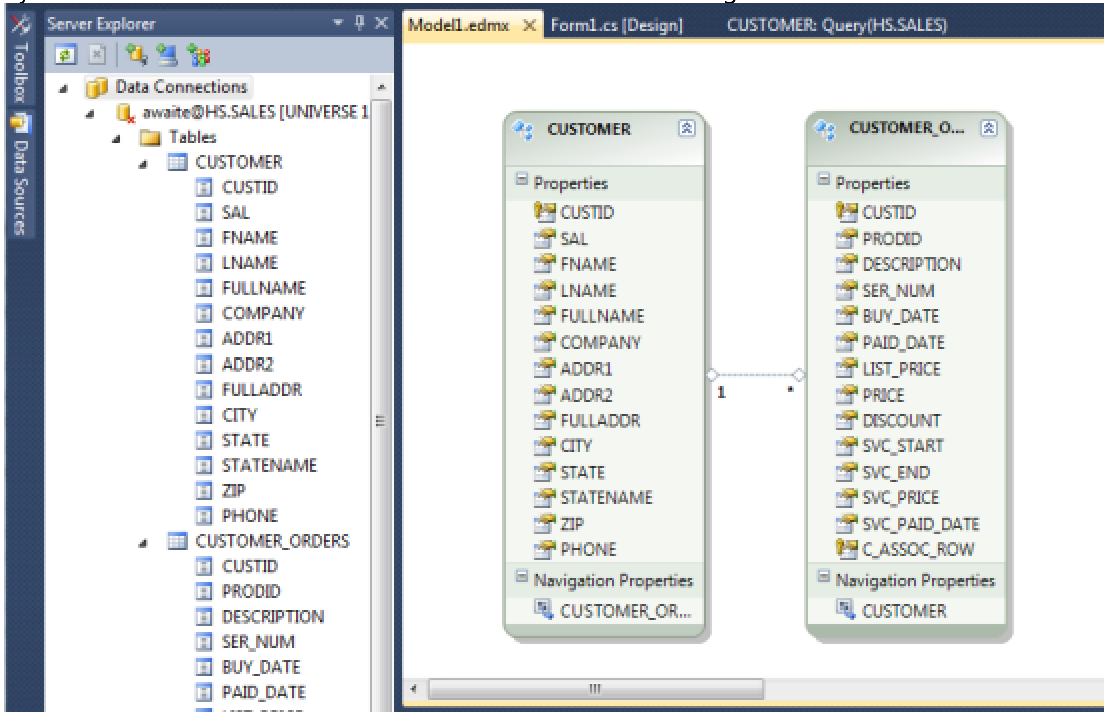
You can use the U2 Toolkit for .NET Developer to expose U2 database files through the Entity Data Model. In the Entity Data Model, data files are mapped to specific object classes and are not inherited.

Procedure

1. In Visual Studio create a new **Windows Forms Application** project and go to **Data > Show Data Sources**.
2. In the **Data Sources** window, click **Add New Data Source**.
3. Choose the **Data Source Type** from which the application will get the data and then click **Next**. For this example, select **Database**.
4. Choose a **Database Model** which determines the types of data objects your application will use and then click **Next**. For this example, select **Entity Data Model**.
5. Choose the contents of the Entity Data Model. Select **Generate from Database** to generate the Entity Data Model from your U2 database and then click **Next**. Classes are generated from the model automatically.
6. Choose the **Data Connection** the application will use to connect to the database. Select **New Connection** to establish a data connection to your local U2 database account and then click **Next**.
7. Choose the **Database Objects** to include in your model. For this example, select **CUSTOMER** and **CUSTOMER_ORDERS** from the **Tables** list then click **Finish**.

Result

The U2 database files are now mapped to the Entity Data Model and are located in the project file, which is denoted by the **.edmx** extension and should look similar to the following:



You can now data-bind items from this Entity Data Model by dragging from the **Data Sources** window onto the forms or existing controls in the project.

Adding a TableAdapter to a project

In Visual Studio, TableAdapters connect to UniData or UniVerse, execute a query or stored procedure, and either returns a new data table containing the returned data or fills an existing DataTable object with the returned data. Updated data from your application can also be sent back to either UniData or UniVerse using a TableAdapter.

Prerequisite

An existing project in Visual Studio that contains a data source. When you drag any table, view, or subroutine into a project, it creates DataSet . A DataSet consists of two files:
• .xsd file
• .Designer.cs file

Procedure

1. From the Solution Explorer, double-click the .xsd file for the project to open the file in the editor.
2. Select **Data > Add > TableAdapter** from the Visual Studio menu. Alternatively, you can drag a TableAdapter from the DataSet Toolbox, onto the form.
3. Specify the connection string used to connect to the database. Click **Next**.
4. Choose the command type (SQL statements or stored procedures) the TableAdapter will use and click **Next**.
5. Optional. Select **Advanced Options** to optionally generate Insert, Update, and Delete statements, turn optimistic concurrency on/off, or add a Select statement to refresh the data table.
6. Add the SQL statement to define what should be loaded into the table. Click **Next**.
7. Select the methods that the TableAdapter will use to load and save the data between the application and the database. Click **Next**.
8. Select **Finish** to exit the TableAdapter wizard.

Developing Applications with Native Visual Studio Integration

Beginning at version 2.1.0, U2 Toolkit for .NET supports Native Visual Studio Integration to U2 applications through the ADO.NET Provider, LINQ to Entity Provider, and Visual Studio
add-ins. Visual Studio users are able to fully access U2 files and subroutines without having to make changes to their account dictionaries. For example, U2 Toolkit for .NET can natively access or modify System Builder accounts and subroutines. U2 Toolkit users can seamlessly use single values, multivalues, or multi-subvalues from within the Visual Studio Explorer.

Server Explorer integration

When you connect to U2 Toolkit for .NET using Native Visual Studio Integration, all account files and all globally cataloged subroutines populate in the Server Explorer in one of two folders: U2 Files or Stored Procedures. The U2 Files folder contains all files and attributes in the account. The Stored Procedures folder contains all of the globally cataloged subroutines in the account. Only U2 data files (such as A,D,I,S, or V field types) display in the Server Explorer.

Entity Data Model and LINQ to Entity

Native Visual Studio Integration supports both Database First models and Code First development. This allows developers to create entity data models from U2 files and subroutines. When an entity is added to the Visual Studio project using the Entity Data Model wizard, the file U2 files is mapped to the entity. Developers can also map u2 subroutines to functions.

DataSets and TableAdapters

In Visual Studio, TableAdapters are used to communicate between an application and the U2 databases. When a file is added to a new dataset using a DataTable, a TableAdapter is also added. These TableAdapters contain all of the functionality required for users to utilize create, read, update, and delete functions for U2 files and subroutines, without having to make any programmatic changes. The TableAdapter sends the request to the database and then returns the query information to the application and populates the information in the DataTable.

Using @ phrases and @SELECT phrases with Native Visual Studio Integration

Native Visual Studio Integration allows users to view U2 accounts, fields, and attributes in the Server Explorer. A U2 account often contains a lot of fields, and it can be helpful to be able to filter out the fields you do not want to see. This can be done by specifying required fields using an @ phrase or and @SELECT phrase.

To filter out dictionary items in the Visual Studio Server Explorer, you can specify your dictionary items in an @ phrase or @SELECT phrase. If both an @ phrase and an @SELECT phrase are specified, the @ phrase will take precedence. If the dictionary contains neither an @SELECT nor an @ phrase, all dictionary items will be populated.

@ phrase example

```
@ phrase
001 PH
002 ID, SV1, SV2, MV1,MV2
```

@SELECT phrase example

```
@SELECT  phrase
001 PH
002 ID, SV1, SV2, MV1,MV2
```

Tip

Rocket does not recommend using wild card (*) queries when using Native Visual Studio Integration.

For more information about using @ and @SELECT phrases, refer to *UniVerse System Description* or *Using UniData SQL*.

Optimistic concurrency

In a development environment, a multiuser development system is often used. In this situation, multiple users can be working on the same project simultaneously without affecting another person's work. In order to prevent the work of one user from affecting anyone else, one of two concurrency controls are typically used: pessimistic and optimistic.

Pessimistic concurrency

Pessimistic transactions behave differently from optimistic transactions, in that they exclusively lock anything read from the database. This eliminates the possibility of concurrent changes to the same data, but also increases the potential for dead locking among different transactions. Since no concurrent read is possible, performance and scalability can suffer in comparison with optimistic transactions. U2 Toolkit for .NET supports pessimistic concurrency using the U2Transaction class (ADO.NET) or the UniTransaction class (UO.NET).

Optimistic concurrency

Optimistic concurrency improves performance because record locking is not required, as record locking consumes additional server resources.

U2 Toolkit for .NET (ADO.NET) and Visual Studio use optimistic concurrency because the data architecture is based on disconnected data (DataSet). Therefore, you need to add business logic to resolve issues with optimistic concurrency.

If you choose to use optimistic concurrency, there are two general ways to determine if changes have occurred: the version approach (true version numbers or date-time stamps) and the saving-all-values approach.

U2 Toolkit for .NET uses saving-all-values approach.
U2 Toolkit for .NET does not currently support the version approach.

You can set/configure Optimistic concurrency using U2 Toolkit for .NET and Visual Studio during:
- DataSet Generation/DataSet Designer
- Entity Data Model Generation/EDM Designer

Subroutines

U2 Toolkit for .NET supports optimistic concurrency in subroutines.

Turning optimistic concurrency on/off in Dataset applications

You can control the optimistic concurrency settings by completing the following steps.

Prerequisite

Adding a TableAdapter to a project

Procedure

1. From the Solution Explorer, double-click the .xsd file for the project to open the file in the editor.
2. Right-click on the TableAdapter and select **Configure**.
3. Click **Advanced Options**.
4. Select or deselect the **Use optimistic concurrency** option and then click **OK**.
5. Select **Finish** to exit the wizard.

Turning optimistic concurrency on/off in Entity Data Model applications

You can control the optimistic concurrency settings by completing the following steps.

Prerequisites

An existing project in Visual Studio that contains a data source.

Procedure

1. From the Solution Explorer, double-click the .edmx file for the project to open the file in the editor.
2. Click the field in the entity where you want to set optimistic concurrency.
3. Select **Concurrency Mode** from the Properties pane.
4. Select **Fixed** to turn optimistic concurrency on. Select **None** to turn it off.

Note

Optimistic concurrency can only be set on the individual fields in an entity, and not on the entity itself.

Turning optimistic concurrency on and off in subroutines

You can use the following U2Parameter class methods to utilize optimistic concurrency in subroutines:

- MV_To_DataTable ( )
- DataTable_To_MV_OC( )
- SetOptimisticConcurrencyError( )

Procedure

1. Execute the subroutine to get the multivalued data and then convert the multivalued data into a .NET DataTable object.
2. Modify the DataTable object. For example,
   dt.Rows[0][0]="foo";
   dt.AcceptChanges();
3. Call the DataTable_To_MV_OC( ) method to convert the modified DataTable into multivalued data.
4. Call the update subroutine and pass the modified multivalue data as INPUT PARAM. This update subroutine should include the OUTPUT PARAM parameter to receive the date operation status.
5. Call the SetOptimisticConcurrencyError() method to set the ROW ERROR as an OptimisticConcurrencyError.

Example

```
U2ConnectionStringBuilder l = new U2ConnectionStringBuilder();
            l.Server = "localhost";
            l.UserID = "user";
            l.Password = "pass";
            l.Database = "HS.SALES";
            l.ServerType = "universe";
            string lconnstr = l.ToString();
            U2Connection c = new U2Connection();
            U2ServerMarks lU2ServerMarks = c.U2ServerMarks;
            c.ConnectionString = lconnstr;
            c.Open();
            U2Command command = c.CreateCommand();
            command.CommandText = "CALL DATASET_TO_MV_UPDATE_SUBROUTINE (?,?)";
            // UniVerse subroutine
            command.CommandType = CommandType.StoredProcedure;
            U2Parameter p1 = new U2Parameter();
            p1.Direction = ParameterDirection.InputOutput;

            p1.Value = data;// modified multi-value data
            p1.ParameterName = "@arg1";

            U2Parameter p2 = new U2Parameter();
            p2.Direction = ParameterDirection.InputOutput;

            p2.Value = ""; // Return Value such as 0:@VM:0:@VM:0:@VM:0:@FM:"Name1":@SVM:
            "Name1NEW":@VM:0:@VM:0:@VM:0:@FM:0:@VM:0:@VM:0:@VM:0
            // non-zero indicates optimistic concurrency
            p2.ParameterName = "@arg2";


            command.Parameters.Add(p1);
            command.Parameters.Add(p2);
            p1.DataTable_To_MV_OC(ds_DataTableToMV.Tables[0]);
            command.ExecuteNonQuery();
            p2.SetOptimisticConcurrencyError(ds_DataTableToMV.Tables[0], 252);
```

Limitations of U2 Toolkit for .NET Developer

The following features are not currently supported in U2 Toolkit for .NET Developer.

Entity Data Model Designer

- The **Update Model from Database** option is not supported.
- UniData customers must manually establish associations between multivalues and multi-subvalues.

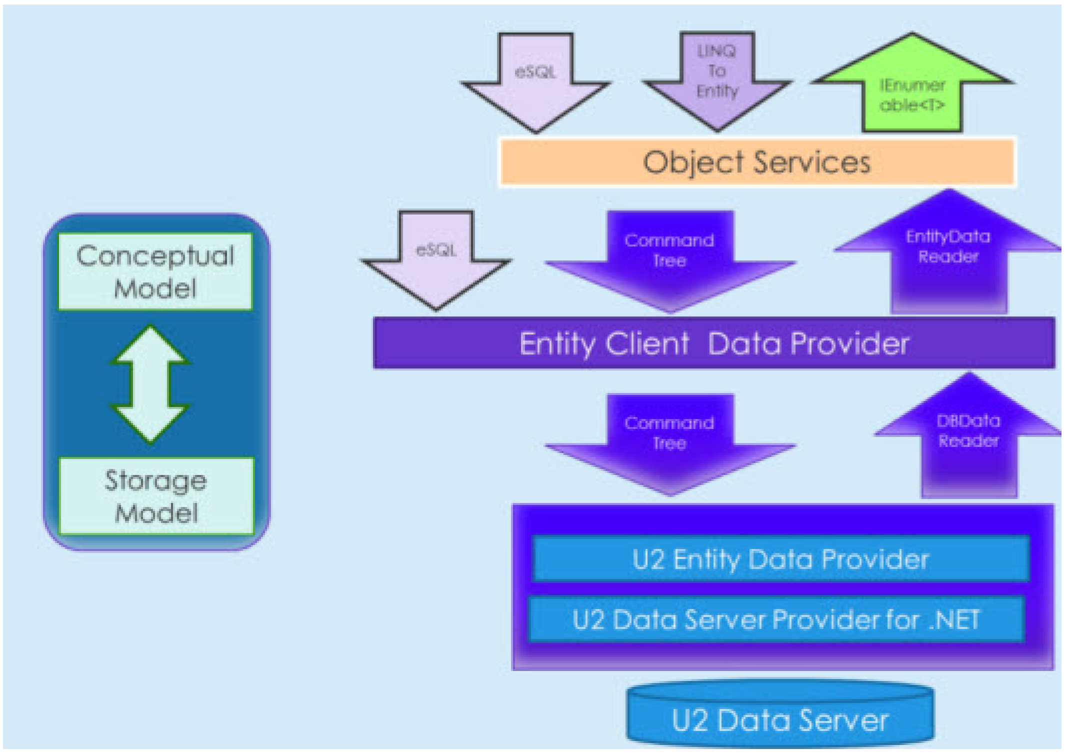
Server Explorer

The KEEP ALIVE value is not supported.

U2 Entity Data Provider for .NET

The U2 Entity Data Provider for .NET is the U2 Entity Framework provider used to connect with a U2 database. It enables developers to create .NET applications that do not rely on additional query-based languages. With this paradigm, developers can write set-based queries directly. Developers create applications by programming against the Entity Data Model (EDM), and a conceptual application model that primarily deals with entities, associations, and conceptual schemas. Applications built using the Entity Framework can work well within an application-centric conceptual model, which allows developers to create mappings between a single conceptual model and various storage schemas without having to change the application code.

The following diagram illustrates the architecture of the U2 Entity Data Provider for .NET.



For more information about the Entity Framework Provider, refer to the Microsoft MSDN documentation.

Tutorial: Developing an application using the U2 Entity Data Provider

In this example, we develop a simple application using ADO.NET. This application calls the HS.SALES account in UniVerse. It loads the information from the CUSTOMER file on to a DataGridView control, and allows you to make changes to the CUSTOMER file using the U2 CommandBuilder class to update the file.

Prerequisite

You must be running ADO.NET Entity Framework 4.1 or later to develop applications using the U2 Entity Data Provider.

Procedure

1. Open Visual Studio 2010.
2. Select the programming language with which you want to work. The examples in this document are all created using C#.
3. Select **File > New Project**. When the New Project dialog box opens, select **Windows Forms Application**.
4. In the name field, enter a name for the project. The project name in this example is **WindowsFormsApplication1**.
5. In the location field, enter the location where you want the project to reside. The location in this example is **C:\walkthrough**.
6. Click **OK**.

The Microsoft Visual Studio Form Design window has three main panes: The form designer, the Solution Explorer, and the Properties pane. You can create and edit your application in the form designer by dragging items from the Visual Studio Toolbox onto the form. The Solution Explorer provides a navigation tree view of all the files associated with your project. The property pane allows you to set the properties of the form and of the individual objects on the form.

The new project opens in the form designer.

Next

Tutorial: Adding a reference to the project

Tutorial: Adding a reference to the project

You must add a reference to the U2 Data.Client.dll.

Prerequisite

Tutorial: Developing an application using the U2 Entity Data Provider

Procedure

1. In the Solution Explorer, right-click the **References** node and select **Add Reference** from the context menu.
2. Select the **Browse** tab and navigate to (on a 64-bit Windows machine): :

   32-bit: C:\Program Files\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\bin\.NETFramework\v4.0

   64-bit: C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\bin\.NETFramework\v4.0

3. Select the U2.Data.Client.dll and click **OK**.

Next

Tutorial: Adding controls to the form

Tutorial: Adding controls to the form

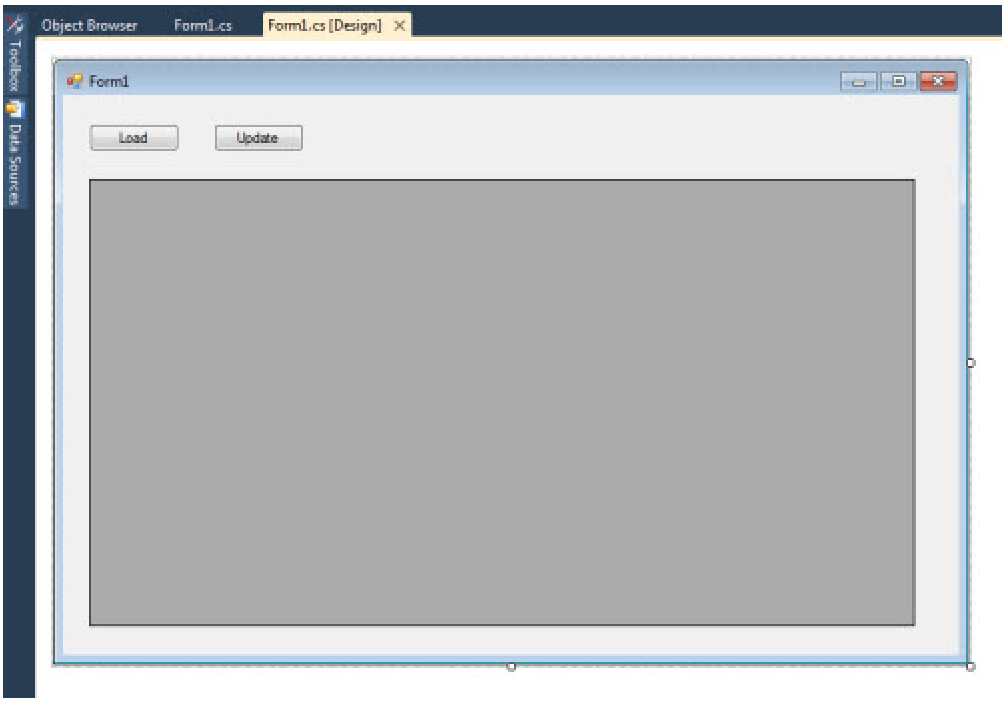
After adding a .dll reference to the project, you can add controls to the form.

Prerequisite

Tutorial: Adding a reference to the project

Procedure

1. From the Visual Studio Toolbox, drag two buttons onto the form. In the properties window, change the button properties as follows:
    - Change the Text property of Button1 to **Load**
    - Change the Text property of Button2 to **Update**
2. From the Visual Studio Toolbox, drag a **DataGridView** control onto the form. Your form should look similar to the following:



Next

Tutorial: Creating a new Entity Data Model from a U2 data source

Tutorial: Creating a new Entity Data Model from a U2 data source

After adding controls to the form, you are ready to create a new Entity Data Model from a U2 data source.

Prerequisite

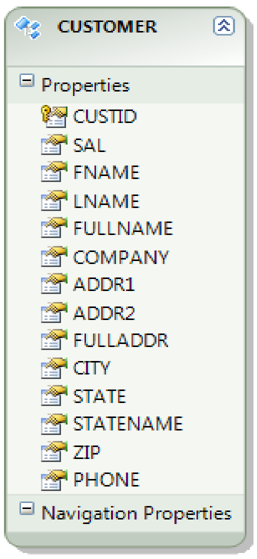
Tutorial: Adding controls to the form

Procedure

1. In Visual Studio create a new **Windows Forms Application** project and go to **Data > Show Data Sources**.
2. In the **Data Sources** window, click **Add New Data Source**.
3. Choose the **Data Source Type** from which the application will get the data and then click **Next**. For this example, select **Database**.
4. Choose a **Database Model** which determines the types of data objects your application will use and then click **Next**. For this example, select **Entity Data Model**.
5. Choose the contents of the Entity Data Model. Select **Generate from Database** to generate the Entity Data Model from your U2 database and then click **Next**. Classes are generated from the model automatically.
6. Choose the **Data Connection** the application will use to connect to the database. Select **New Connection** to establish a data connection to your local U2 database account and then click **Next**.
7. Choose the **Database Objects** to include in your model. For this example, select **CUSTOMER** from the **Tables** list then click **Finish**.

Result

The U2 database files are now mapped to the Entity Data Model and are located in the project file, which is denoted by the **.edmx** extension. The Customer.edmx should look similar to the following:



CUSTOMER

Properties

CUSTID
SAL
FNAME
LNAME
FULLNAME
COMPANY
ADDR1
ADDR2
FULLADDR
CITY
STATE
STATENAME
ZIP
PHONE

Navigation Properties

Next

Tutorial: Creating event handlers for the button controls

Tutorial: Creating event handlers for the button controls

After creating the Entity Data Model, you must add some event handlers to the form.

Prerequisite

Tutorial: Creating a new Entity Data Model from a U2 data source

Tip

This example uses the Entity data model (CUSTOMER.EDMX) file. Refer to the C# EntityFramework.sln sample code for details about this file. The sample code is located by default in the following location:

On 32-bit machines: C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples

On 64-bit machines: C:\Program Files\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\samples

Procedure

After adding the Entity Data Model data source, create an event handler for each button click. This requires you to add a small amount of code to the form.

1. Double-click **Load** to create an event handler for the button. The code editor opens.
2. Add the following code to the **Button1** click event:

```
{
public partial class Form1 : Form
{

    CustomerEntities m_ctx = new CustomerEntities();
    public Form1()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        var q = from c in m_ctx.CUSTOMERs
                select c;

        this.dataGridView1.DataSource = q;
```

3. Add the following code to the **Button2** click event:

```
 private void button2_Click(object sender, EventArgs e)
    {
        var q = from c in m_ctx.CUSTOMERs
                select c;
        CUSTOMER cust = q.FirstOrDefault();
        cust.FNAME = "Diana2";
        m_ctx.SaveChanges();

    }
```

Result

The Button1 event handler creates a connection the to the HS.SALES database and then loads the information from the CUSTOMER account into the data grid.

The Button2 event handler uses the U2 CommandBuilder class to generate an update statement. It checks to see whether any changes have been made to the record(s). If it detects any changes, it first checks to see how many changes have been made and then repopulates the DataGrid using the generated Update statement.

Next

Tutorial: Building the application

Tutorial: Building the application

After adding all of the references, controls, and event handlers to the form, you are ready to build the application.

Prerequisite

Tutorial: Creating event handlers for the button controls

Procedure

1. Select **Build > Build Solution** from the Visual Studio toolbar.
2. Run the application. To do this, click **Debug** on the Visual Studio toolbar, and then select the **Start Without Debugging** option. This opens a working version of the application you just created.
3. Click **Load**. The customer information populates in the GridView.
4. In the grid, locate the name **Diana** in the FNAME field of customer 2. Click in the cell to edit the name, and change it to **Diana2**. Click **Update**.

Result

The change is saved to your account.

Tutorial: Developing an application to call a subroutine using a Native Access connection

This example demonstrates a simple application using UO.NET. The application calls the "*GETXMLSUB" subroutine against the HS.SALES account in UniVerse. It runs the "LIST CUSTOMER" command to return the CUSTOMER XML data.

Procedure

1. Open Visual Studio 2010.
2. Select the programming language with which you want to work. The examples in this document are all created using C#.
3. Select **File > New Project**. When the New Project dialog box opens, select **Console Application**.
4. In the name field, enter a name for the project. The project name in this example is **ConsoleApplication1**.
5. In the location field, enter the location where you want the project to reside. The location in this example is **C:\walkthrough**.
6. Click **OK**. The project opens in the Visual Studio code editor.

Next

Tutorial: Adding a reference to the project

Tutorial: Adding a reference to the project

You must add a reference to the U2 Data.Client.dll.

Prerequisite

Tutorial: Developing an application to call a subroutine using a Native Access connection

Procedure

1. In the Solution Explorer, right-click the **References** node and select **Add Reference** from the context menu.
2. Select the **Browse** tab and navigate to (on a 64-bit Windows machine): :

   32-bit: C:\Program Files\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\bin\.NETFramework\v4.5

   64-bit: C:\Program Files (x86)\Rocket Software\U2 Toolkit for .NET\U2 Database Provider\bin\.NETFramework\v4.5

3. Select the **U2.Data.Client.dll** and click **OK**.
4. Add the following *using* statements to the form:

   using statements

   ```
   using System.Data;
   using U2.Data.Client.dll
   ```

Next

Tutorial: Adding controls to the form

Tutorial: Adding the subroutine calling code

After adding a reference to the project, you must add the code to the form that will call the subroutine.

Prerequisite

Tutorial: Adding a reference to the project

Procedure

1. Add the following code to the application.

Subroutine calling code

```
    try
        {
U2ConnectionStringBuilder conn_str = new U2ConnectionStringBuilder();
       conn_str.UserID = "administrator";
       conn_str.Password = "password";
       conn_str.Server = "localhost";
       conn_str.ServerType = "universe";
       conn_str.Database = "HS.SALES";
       conn_str.AccessMode = "Native";
       conn_str.RpcServiceType = "uvcs";
       string s = conn_str.ToString();
       U2Connection con = new U2Connection();
       con.ConnectionString = s;
       con.Open();
       Console.WriteLine("Connected.........................");
       U2Command command = con.CreateCommand();
       command.CommandText = "*GETXMLSUB"; // UniVerse subroutine
       command.Parameters.Clear();
       command.CommandType = CommandType.StoredProcedure;
       U2Parameter p1 = new U2Parameter();
       p1.Direction = ParameterDirection.InputOutput;
       p1.Value = "LIST CUSTOMER";
       p1.ParameterName = "@arg1";
       U2Parameter p2 = new U2Parameter();
       p2.Direction = ParameterDirection.InputOutput;
       p2.Value = "";
       p2.ParameterName = "@arg2";
       U2Parameter p3 = new U2Parameter();
       p3.Direction = ParameterDirection.InputOutput;
       p3.Value = "";
       p3.ParameterName = "@arg3";
       U2Parameter p4 = new U2Parameter();
       p4.Direction = ParameterDirection.InputOutput;
       p4.Value = "";
       p4.ParameterName = "@arg4";
       U2Parameter p5 = new U2Parameter();
       p5.Direction = ParameterDirection.InputOutput;
```

```
        p5.Value = "";
        p5.ParameterName = "@arg5";
        U2Parameter p6 = new U2Parameter();
        p6.Direction = ParameterDirection.InputOutput;
        p6.Value = "";
        p6.ParameterName = "@arg6";
        command.Parameters.Add(p1);
        command.Parameters.Add(p2);
        command.Parameters.Add(p3);
        command.Parameters.Add(p4);
        command.Parameters.Add(p5);
        command.Parameters.Add(p6);
        command.ExecuteNonQuery();
        string s1 = command.Parameters[0].Value.ToString();//command
        string s2 = command.Parameters[1].Value.ToString();// command option
        string s3 = command.Parameters[2].Value.ToString(); // xml
        string s4 = command.Parameters[3].Value.ToString(); //xsd
        string s5 = command.Parameters[4].Value.ToString(); // msg #
        string s6 = command.Parameters[5].Value.ToString(); // msg description
        Console.WriteLine("Subroutine xml Output:" + s3 + Environment.NewLine);
        Console.WriteLine("Subroutine xsd Output:" + s4 + Environment.NewLine);
        con.Close();
        Console.Read();
        }
        catch (Exception e) {
            Console.WriteLine(e.Message);
        }
        finally {
            Console.WriteLine("Enter to exit:");
            string line = Console.ReadLine();
        }
```

2. Update the login information in the sample code with your login credentials.
3. Click **Build > Build Solution** to run the program.

Result

The CUSTOMER file is output as both an XML file and an XSD file.

Working with Entity Framework 6

Beginning with Entity Framework 6.x, the Entity Framework is shipped independently of the .NET Framework. Visual Studio projects require Visual Studio 2012 or later to work with .NET Framework 4.5 or 4.6. Entity Framework 6.1 and later also requires the Manage NuGet Packages tool to install additional packages to the project. For more information about managing NuGet packages, refer to the Microsoft website at https://docs.microsoft.com/en-us/.

When using the U2 Toolkit Entity Framework driver, Visual Studio projects won't recognize a U2 driver by default. The U2 Data Client Entity information must be added manually to Visual Studio projects via the app.config configuration file in the U2 Toolkit application. The following examples highlight these change in the app.config file.

Example

In this example, the app.config file has been modified to use the U2 .NET Framework 4.5 driver to work with Entity Framework 6.1.

U2 .NET Framework 4.5 driver with Entity Framework 6.1

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="entityFramework"
    type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
     EntityFramework, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"
      requirePermission="false"/>
  </configSections>
  <connectionStrings>
    <add name="CustomerContext" connectionString="Database=HS.SALES;User
ID=user;Password=pass;Server=localhost;Pooling=false;ServerType=universe;
    ConnectTimeout=360;PersistSecurityInfo=true"
    providerName="U2.Data.Client.4.5"/>
  </connectionStrings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2"/>
  </startup>
  <entityFramework>
    <providers>
      <provider invariantName="U2.Data.Client.4.5"
       type="U2.Data.Client.Entity.U2ProviderServices, U2.Data.Client.Entity,
      Version=2.2.2.0, Culture=neutral, PublicKeyToken=883335d992998a08"/>
    </providers>
  </entityFramework>
</configuration>
```

Example

In this example, the app.config file has been modified to use the U2 .NET Framework 4.6 driver to work with Entity Framework 6.1.

U2 .NET Framework 4.6 driver with Entity Framework 6.1

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="entityFramework"
    type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false"/>
  </configSections>
  <connectionStrings>
    <add name="CustomerContext" connectionString="Database=HS.SALES;User
ID=user;Password=pass;Server=localhost;Pooling=false;ServerType=universe;
ConnectTimeout=360;PersistSecurityInfo=true"
providerName="U2.Data.Client.4.6"/>
  </connectionStrings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6"/>
  </startup>
  <entityFramework>
    <providers>
      <provider invariantName="U2.Data.Client.4.6"
type="U2.Data.Client.Entity.U2ProviderServices, U2.Data.Client.Entity,
Version=2.2.2.0, Culture=neutral, PublicKeyToken=7f1dc11a3fe611eb"/>
    </providers>
  </entityFramework>
</configuration>
```

Restrictions

- Visual Studio 2012 or 2013 applications must use the .NET Framework 4.5 compiler target option with Entity Framework 6.1 to create new U2 Entity data.
- Visual Studio 2015 or 2017 applications must use the .NET Framework 4.6 compiler target option with Entity Framework 6.1 to create new U2 Entity data.

Limitations of the U2 Entity Data Provider

The following features are not supported in the U2 Entity Data Provider.

LINQ to Entity

- Group
- Skip
- Functions such as TRIM, RIGHT, DIFFERENCE, etc.
- Except
- Intersect
- Concat*
- Union with Distinct
- OrderBy with FK Collection
- Instead of Concat, use Union