DB2

DB2 Version 9
for Linux, UNIX, and Windows

Developing ADO.NET and OLE DB Applications

SC10-4230-00

DB2®

DB2 Version 9
for Linux, UNIX, and Windows

Developing ADO.NET and OLE DB Applications

SC10-4230-00

Contents

Chapter 1. ADO.NET development for DB2 databases

ADO.NET application development

In recent years, Microsoft has been promoting a new software development platform for Windows, known as the .NET Framework. The .NET Framework is Microsoft's replacement for Component Object Model (COM) technology. The following points highlight the key .NET Framework features:

- You can code .NET applications in over forty different programming languages. The most popular languages for .NET development are C# and Visual Basic .NET.
- The .NET Framework class library provides the building blocks with which you build .NET applications. This class library is language agnostic and provides interfaces to operating system and application services.
- Your .NET application (regardless of language) compiles into Intermediate Language (IL), a type of bytecode.
- The Common Language Runtime (CLR) is the heart of the .NET Framework, compiling the IL code on the fly, and then running it. In running the compiled IL code, the CLR activates objects, verifies their security clearance, allocates their memory, executes them, and cleans up their memory once execution is finished.

Through these features, the .NET Framework facilitates a wide variety of application implementations (for instance, Windows forms, web forms, and web services), rapid application development, and secure application deployment. COM and COM+ proved to be inadequate or cumbersome for all the aforementioned features.

The .NET Framework provides extensive data access support through ADO.NET. ADO.NET supports both connected and disconnected access. The key component of disconnected data access in ADO.NET is the DataSet class, instances of which act as a database cache that resides in your application's memory. Connected access in ADO.NET requires no additional classes.

For both connected and disconnected access, your applications use databases through what's known as a .NET data provider. Various database products include their own .NET data providers, including DB2 for Windows.

A .NET data provider features implementations of the following base classes:

- Connection: Establishes and manages a database connection.
- Command: Executes an SQL statement against a database.
- DataReader: Reads and returns result set data from a database.
- DataAdapter: Links a DataSet instance to a database. Through a DataAdapter instance, the DataSet can read and write database table data.

DB2 UDB for Windows includes three .NET data providers:

- DB2 .NET Data Provider: A high performance, managed ADO.NET data provider. This is the recommended .NET data provider for use with DB2 family databases. ADO.NET database access using the DB2 .NET Data Provider has fewer restrictions, and provides significantly better performance than the OLE DB and ODBC .NET bridge providers.

- OLE DB .NET Data Provider: A bridge provider that feeds ADO.NET requests to the IBM OLE DB Provider (by way of the COM interop module). This .NET data provider is not recommended for access to DB2 family databases. The DB2 .NET Data Provider is faster and more feature-rich.
- ODBC .NET Data Provider: A bridge provider that feeds ADO.NET requests to the IBM ODBC Driver. This .NET data provider is not recommended for access to DB2 family databases. The DB2 .NET Data Provider is faster and more feature-rich.

Supported .NET development software

To develop and deploy .NET applications that run against DB2® databases, you will need to use supported development software and operating systems.

Supported operating systems for developing and deploying .NET Framework 1.1 applications:

- Windows® 2000
- Windows XP (32-bit edition)
- Windows Server 2003 (32-bit edition)

Note: Windows 98, Windows ME, and Windows NT® are also supported, but only for running DB2 client-side applications.

Supported operating systems for developing and deploying .NET Framework 2.0 applications:

- Windows 2000, Service Pack 3
- Windows XP, Service Pack 2 (32-bit and 64-bit editions)
- Windows Server 2003 (32-bit and 64-bit editions)

Note: Windows 98 and Windows ME are also supported, but only for running DB2 client-side applications.

Supported development software for .NET Framework applications:

In addition to a DB2 client, you will need one of the following options to develop .NET Framework applications.

- Visual Studio 2003 (for .NET Framework 1.1 applications)
- Visual Studio 2005 (for .NET Framework 2.0 applications)
- .NET Framework 1.1 Software Development Kit and .NET Framework Version 1.1 Redistributable Package (for .NET Framework 1.1 applications)
- .NET Framework 2.0 Software Development Kit and .NET Framework Version 2.0 Redistributable Package (for .NET Framework 2.0 applications)

Supported deployment software for .NET Framework applications:

In addition to a DB2 client , you will need one of the following two options to deploy .NET Framework applications.

- .NET Framework Version 1.1 Redistributable Package (for .NET Framework 1.1 applications)
- .NET Framework Version 2.0 Redistributable Package (for .NET Framework 2.0 applications)

Setting up the Windows application development environment

When you install the DB2 Client on Windows operating systems, the install program updates the configuration registry with the environment variables INCLUDE, LIB, and PATH. The system-wide environment variable, DB2INSTANCE, is set by install to the default instance created, called DB2. DB2PATH is set inside a DB2 command window when the window is opened.

You can override these environment variables to set the values for the machine or the currently logged-on user. Exercise caution when changing these environment variables. Do not change the DB2PATH environment variable. DB2INSTANCE is defined as a system-level environment variable. You do not need to make use of the DB2INSTDEF DB2 registry variable which defines the default instance name to use if DB2INSTANCE is not set.

Procedure:

To override the environment variable settings, use the Windows Control Panel.

When using the variable `%DB2PATH%` in a command, put the full path in quotes, as in:

```
set LIB="%DB2PATH%%\lib";%LIB%
```

The default installation value for this variable is `\Program Files\IBM\SQLLIB`, which contains a space, so not using quotes can cause an error.

In addition, you must take the following specific steps for running DB2 applications:

- When building C or C++ programs, you must ensure that the `INCLUDE` environment variable contains `%DB2PATH%\INCLUDE` as the first directory.

  To do this, update the environment setup file for your compiler:

  Microsoft Visual C++ 6.0
  `"C:\Program Files\Microsoft Visual Studio\VC98\bin\vcvars32.bat"`

  Microsoft Visual C++ .NET
  `"C:\Program Files\Microsoft Visual Studio .NET\Common7\Tools\vsvars32.bat"`

  These files have the following commands:

  Microsoft Visual C++ 6.0

  ```
  set INCLUDE=%MSVCDir%\ATL\INCLUDE;%MSVCDir%\INCLUDE;
     %MSVCDir%\MFC\INCLUDE;%INCLUDE%
  ```

  Microsoft Visual C++ .NET

  ```
  @set INCLUDE=%MSVCDir%\ATLMFC\INCLUDE;...;
     %FrameworkSDKDir%\include;%INCLUDE%
  ```

  To use either file with DB2, first move `%INCLUDE%`, which sets the `%DB2PATH%\INCLUDE` path, from the end of the list to the beginning, as follows:

  Microsoft Visual C++ 6.0

  ```
  set INCLUDE=%INCLUDE%;%MSVCDir%\ATL\INCLUDE;
     %MSVCDir%\INCLUDE;%MSVCDir%\MFC\INCLUDE
  ```

  Microsoft Visual C++ .NET

```
@set INCLUDE=%INCLUDE%;%MSVCDir%\ATLMFC\INCLUDE;...;
   %FrameworkSDKDir%\include
```

- When building Micro Focus COBOL programs, set the `COBCPY` environment variable to point to `%DB2PATH%\INCLUDE\cobol_mf`.
- When building IBM COBOL programs, set the `SYSLIB` environment variable to point to `%DB2PATH%\INCLUDE\cobol_a`.
- Ensure the LIB environment variable points to `%DB2PATH%\lib` by using:

  ```
  set LIB="%DB2PATH%\lib";%LIB%
  ```

  Note: To enable cross-developing 64-bit applications from a 32-bit environment, see Migrating 32-bit database applications to run on 64-bit instances
- Ensure that the `DB2COMM` environment variable is set at the server of a remote database.
- Ensure that the security service has started at the server for SERVER authentication, and at the client, when using CLIENT authentication.

  Note: Because CLIENT authentication happens on the client side instead of the server side, the client application is running under the context of the user. The Win32 authentication API requires certain privileges that might or might not be held by the user. In order to make sure that the CLIENT authentication takes place successfully, the authentication requests are passed from the client application to the security server (which runs under a privileged account local system by default, and has the right to call the authentication API).

  To start the security service manually, use the `NET START DB2NTSECSERVER` command.

  Normally, the only time you would want to set the security service to start automatically is if the workstation is acting as a DB2 client connecting to a server that is configured for Client Authentication. To have the security service start automatically, do the following:

  Windows 2000 and Windows Server 2003

  1. Click the ″Start″ button.
  2. For Windows 2000, click ″Settings″ and then click ″Control Panel″.

     For Windows Server 2003, click ″Control Panel″.
  3. Click ″Administrative Tools″.
  4. Click ″Services″.
  5. In the Services window, highlight ″DB2 Security Server″.
  6. If it does not have the settings ″Started″ and ″Automatic″ listed, click ″Action″ from the top menu.
  7. Click ″Properties″.
  8. Make sure you are in the ″General″ tab.
  9. Choose ″Automatic″ from the ′Startup Type′ drop-down menu.
  10. Click ″OK″.
  11. Reboot your machine to have the settings take effect.

  Windows XP

  1. Click the ″Start″ button.
  2. Click ″Settings″.
  3. Click ″Control Panel″.
  4. Click ″Performance and Maintenance″.

5. Click ″Administrative Tools″.
6. Click ″Services″.
7. In the Services window, highlight ″DB2 Security Server″.
8. If it does not have the settings ″Started″ and ″Automatic″ listed, click ″Action″ from the top menu.
9. Click ″Properties″.
10. Make sure you are in the ″General″ tab.
11. Choose ″Automatic″ from the ′Startup Type′ drop-down menu.
12. Click ″OK″.
13. Reboot your machine to have the settings take effect.

The database manager on a Windows XP, Windows Server 2003,or a Windows 2000 environment is implemented as a service, and hence does not return errors or warnings when the service is started, though problems might have occurred. This means that when you run the `db2start` or the `NET START` command, no warnings will be returned if any communication subsystem failed to start. Therefore, the user should always examine the event logs or the DB2 Administration Notification log for any errors that might have occurred during the running of these commands.

If you will be using DB2 CLI or Java™, proceed to the appropriate task:

– Setting up the Windows CLI environment
– Setting up the Windows CLI environment

Related tasks:

- “Migrating 32-bit database applications to run on 64-bit instances” in *Migration Guide*
- “Setting up the Windows CLI environment” in *Call Level Interface Guide and Reference, Volume 1*
- “Installing the IBM DB2 Driver for JDBC and SQLJ” in *Developing Java Applications*
- “Configuring the operating system for database application development” in *Getting Started with Database Application Development*
- “Setting environment variables on Windows” in *Administration Guide: Implementation*
- “Setting the default instance when using multiple DB2 copies (Windows)” in *Administration Guide: Implementation*

Related reference:

- “Supported operating systems for database application development” in *Getting Started with Database Application Development*

DB2 integration in Visual Studio

The IBM Database Add-Ins for Visual Studio 2003 and 2005 are a collection of features that integrate seamlessly into your Visual Studio development environment so that you can work with DB2 servers and develop DB2 procedures, functions, and objects. These add-ins are designed to present a simple interface to DB2 databases. For example, instead of using SQL, the creation of database objects can be done using wizards. And for situations where you do need to write SQL code, the integrated DB2 SQL editor has the following features:

- Colored SQL text for increased readability

- Integration with the Microsoft Visual Studio IntelliSense feature, which provides for intelligent auto-completion while you are typing DB2 scripts

With IBM Database Add-Ins for Visual Studio, you can:
- Open various DB2 development and administration tools
- Create and manage DB2 projects in the Solution Explorer
- Access and manage DB2 data connections (in Visual Studio 2005 you can do this from the Server Explorer; in Visual Studio 2003, you can do this from the IBM Explorer)
- Create and modify DB2 scripts, including scripts to create stored procedures, functions, tables, views, indexes, and triggers

Following are the means by which IBM Database Add-Ins for Visual Studio can be installed on your computer.

Visual Studio 2003
The IBM Database Add-Ins for Visual Studio 2003 are included with the DB2 Client and the DB2 servers. The DB2 installation detects the presence of Visual Studio 2003, and if it is installed, the add-ins are registered. If you install Visual Studio 2003 after you install a DB2 product, run the ″Register Visual Studio Add-Ins″ utility from the DB2 instance’s start menu.

Visual Studio 2005
The IBM Database Add-Ins for Visual Studio 2005 are included as a separately installable component with the DB2 Client and the DB2 servers. Once you are finished installing your DB2 product, you will be presented with an option to install the IBM Database Add-Ins for Visual Studio 2005. If you do not have Visual Studio 2005 installed on your computer, the add-ins will not install. Once you install Visual Studio 2005, you can then install the add-ins at any time from the DB2 product’s setup menu.

Chapter 2. Programming applications for the DB2 .NET Data Provider

DB2 .NET Data Provider

The DB2 .NET Data Provider extends DB2 support for the ADO.NET interface. The DB2 .NET Data Provider delivers high-performing, secure access to DB2 data.

The DB2 .NET Data Provider allows your .NET applications to access the following database management systems:

- DB2 Database for Linux, UNIX, and Windows, Version 9
- DB2 Universal Database™ Version 8 for Windows, UNIX®, and Linux-based computers
- DB2 Universal Database Version 6 (or later) for OS/390® and z/OS®, through DB2 Connect™
- DB2 Universal Database Version 5, Release 1 (or later) for AS/400® and iSeries™, through DB2 Connect
- DB2 Universal Database Version 7.3 (or later) for VSE & VM, through DB2 Connect

To develop and run applications that use DB2 .NET Data Provider you need the .NET Framework, Version 2.0 or 1.1.

In addition to the DB2 .NET Data Provider, the IBM Database Development Add-In enables you to quickly and easily develop .NET applications for DB2 databases in Visual Studio 2005. You can also use the Add-In to create database objects such as indexes and tables, and develop server-side objects, such as stored procedures and user-defined functions.

DB2 .NET Data Provider database system requirements

The DB2 .NET Data Provider enables your .NET applications to access the following database management systems:

- DB2 Version 9 for Linux™, UNIX, and Windows
- DB2 Universal Database Version 8 for Linux, UNIX, and Windows
- DB2 Universal Database Version 6 (or later) for OS/390 and z/OS, through DB2 Connect
- DB2 Universal Database Version 5, Release 1 (or later) for AS/400 and iSeries, through DB2 Connect
- DB2 Universal Database Version 7.3 (or later) for VSE & VM, through DB2 Connect

Before using a DB2 client or server installer to install the DB2 .NET Data Provider, you must already have the .NET Framework (Version 1.1 or Version 2.0) installed on the computer. If the .NET Framework is not installed, the DB2 client or server installer will not install the DB2 .NET Data Provider.

For DB2 Universal Database for AS/400 and iSeries, the following fix is required on the server: APAR ii13348.

The .NET Framework Version 1.0 and Visual Studio .NET 2002 are not supported for use with the DB2 Version 9 DB2 .NET Data Provider.

Generic coding with the ADO.NET common base classes

The .NET Framework version 2.0 features a namespace called `System.Data.Common`, which features a set of base classes that can be shared by any .NET data provider. This facilitates a generic ADO.NET database application development approach, featuring a constant programming interface. The main classes in the DB2 .NET Data Provider for .NET Framework 2.0 are inherited from the `System.Data.Common` base classes. As a result, generic ADO.NET applications will work with DB2 databases through the DB2 .NET Data provider.

The following C# demonstrates a generic approach to establishing a database connection.

```
DbProviderFactory factory = DbProviderFactories.GetFactory("IBM.Data.DB2");
DbConnection conn = factory.CreateConnection();
DbConnectionStringBuilder sb = factory.CreateConnectionStringBuilder();

if( sb.ContainsKey( "Database" ) )
{
   sb.Remove( "database" );
   sb.Add( "database", "SAMPLE" );
}

conn.ConnectionString = sb.ConnectionString;

conn.Open();
```

The `DbProviderFactory` object is the point where any generic ADO.NET application begins. This object creates generic instances of .NET data provider objects, such as connections, data adapters, commands, and data readers, which work with a specific database product. In the case of the example above, the `"IBM.Data.DB2"` string passed into the `GetFactory` method uniquely identifies the DB2 .NET Data Provider, and results in the initialization of a `DbProviderFactory` instance that creates database provider object instances specific to the DB2 .NET Data Provider. The `DbConnection` object can connect to DB2 family databases, just as a `DB2Connection` object, which is actually inherited from `DbConnection`. Using the `DbConnectionStringBuilder` class, you can determine the connection string keywords for a data provider, and generate a custom connection string. The code in the above example checks if a keyword named `"database"` exists in the DB2 .NET Data Provider, and if so, generates a connection string to connect to the SAMPLE database.

Connecting to a database from an application using the DB2 .NET Data Provider

When using the DB2 .NET Data Provider, a database connection is established through the `DB2Connection` class. First, you must create a string that stores the connection parameters.

Examples of possible connection strings are:

```
String connectString = "Database=SAMPLE";
// When used, attempts to connect to the SAMPLE database.
```

```
String cs = "Server=srv:50000;Database=SAMPLE;UID=db2adm;PWD=ab1d;Connect Timeout=30";
// When used, attempts to connect to the SAMPLE database on the server
// 'srv' through port 50000 using 'db2adm' and 'ab1d' as the user id and
// password respectively. If the connection attempt takes more than thirty seconds,
the attempt will be terminated and an error will be generated.
```

To create the database connection, pass the `connectString` to the `DB2Connection` constructor. Then use the `DB2Connection` object's `Open` method to formally connect to the database identified in `connectString`.

Connecting to a database in C#:

```
String connectString = "Database=SAMPLE";
DB2Connection conn = new DB2Connection(connectString);
conn.Open();
return conn;
```

Connecting to a database in Visual Basic .NET:

```
Dim connectString As String = "Database=SAMPLE"
Dim conn As DB2Connection = new DB2Connection(connectString)
conn.Open()
Return conn
```

Connection pooling with the DB2 .NET Data Provider

When a connection is first opened against a DB2 database, a connection pool is created. As connections are closed, they enter the pool, ready to be used by other applications needing connections. The DB2 .NET Data Provider enables connection pooling by default. You can turn connection pooling off using the `Pooling=false` connection string keyword/value pair.

You can control the behavior of the connection pool by setting connection string keywords for the following:

- The minimum and maximum pool size (min pool size, max pool size)
- The length of time a connection can be idle before its returned to the pool (connection lifetime)
- Whether or not the current connection will be put in the connection pool when it is closed (connection reset)

Executing SQL statements from an application using the DB2 .NET Data Provider

When using the DB2 .NET Data Provider, the execution of SQL statements is done through a `DB2Command` class using its methods `ExecuteReader()` and `ExecuteNonQuery()`, and its properties `CommandText`, `CommandType` and `Transaction`. For SQL statements that produce output, the `ExecuteReader()` method should be used and its results can be retrieved from a `DB2DataReader` object. For all other SQL statements, the method `ExecuteNonQuery()` should be used. The `Transaction` property of the `DB2Command` object should be initialized to a `DB2Transaction`. A `DB2Transaction` object is responsible for rolling back and committing database transactions.

Executing an UPDATE statement in C#:

```
// assume a DB2Connection conn
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "                FROM staff " +
    "                 WHERE id >= 310) " +
                  "  WHERE id = 310";
cmd.ExecuteNonQuery();
```

Executing an UPDATE statement in Visual Basic .NET:

```
' assume a DB2Connection conn
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "                FROM staff " +
                  "                WHERE id >= 310) " +
                  "  WHERE id = 310";
cmd.ExecuteNonQuery();
```

Executing a SELECT statement in C#:

```
// assume a DB2Connection conn
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "SELECT deptnumb, location " +
                  "  FROM org " +
                  "  WHERE deptnumb < 25";
DB2DataReader reader = cmd.ExecuteReader();
```

Executing a SELECT statement in Visual Basic .NET:

```
' assume a DB2Connection conn
Dim cmd As DB2Command = conn.CreateCommand()
Dim trans As DB2Transaction = conn.BeginTransaction()
cmd.Transaction = trans
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "                FROM staff " +
                  "                WHERE id >= 310) " +
                  "  WHERE id = 310"
cmd.ExecuteNonQuery()
```

Once your application has performed a database transaction, you must either roll it back or commit it. This is done through the `Commit()` and `Rollback()` methods of a `DB2Transaction` object.

Rolling back or committing a transaction in C#:

```
// assume a DB2Transaction object conn
trans.Rollback();
...
trans.Commit();
```

Rolling back or committing a transaction in C#:

```
' assume a DB2Transaction object conn
trans.Rollback()
...
trans.Commit()
```

Reading result sets from an application using the DB2 .NET Data Provider

When using the DB2 .NET Data Provider, the reading of result sets is done through a `DB2DataReader` object. The `DB2DataReader` method, `Read()` is used to advance to the next row of result set. The methods `GetString()`, `GetInt32()`, `GetDecimal()`, and other methods for all the available data types are used to extract data from the individual columns of output. `DB2DataReader`'s `Close()` method is used to close the `DB2DataReader`, which should always be done when finished reading output.

Reading a result set in C#:

```
// assume a DB2DataReader reader
Int16 deptnum = 0;
String location="";

// Output the results of the query
while(reader.Read())
{
  deptnum = reader.GetInt16(0);
  location = reader.GetString(1);
  Console.WriteLine("    " + deptnum + " " + location);
}
reader.Close();
```

Reading a result set in Visual Basic .NET:

```
' assume a DB2DataReader reader
Dim deptnum As Int16 = 0
Dim location As String ""

' Output the results of the query
Do While (reader.Read())
  deptnum = reader.GetInt16(0)
  location = reader.GetString(1)
  Console.WriteLine("    " & deptnum & " " & location)
Loop
reader.Close();
```

Calling stored procedures from an application using the DB2 .NET Data Provider

When using the DB2 .NET Data Provider, you can call stored procedures by using a `DB2Command` object. The default value of the `CommandType` property is `CommandType.Text`. This is the appropriate value for SQL statements and can also be used to call stored procedures. However, calling stored procedures is easier when you set `CommandType` to `CommandType.StoredProcedure`. In this case, you only need to specify the stored procedure name and any parameters.

The following examples demonstrates how to invoke a stored procedure called `INOUT_PARAM`, with the `CommandType` property set to either `CommandType.StoredProcedure` or `CommandType.Text`.

Calling a stored procedure by setting the `CommandType` property of the `DB2Command` to `CommandType.StoredProcedure` in C#:

```
// assume a DB2Connection comm
DB2Transaction trans = conn.BeginTransaction();
DB2Command cmd = conn.CreateCommand();
String procName = "INOUT_PARAM";
cmd.Transaction = trans;
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = procName;

// Register input-output and output parameters for the DB2Command
...

// Call the stored procedure
Console.WriteLine("  Call stored procedure named " + procName);
cmd.ExecuteNonQuery();
```

Calling a stored procedure by setting the CommandType property of the `DB2Command` to `CommandType.Text` in C#:

```
// assume a DB2Connection comm
DB2Transaction trans = conn.BeginTransaction();
DB2Command cmd = conn.CreateCommand();
String procName = "INOUT_PARAM";
String procCall = "CALL INOUT_PARAM (?, ?, ?)";
cmd.Transaction = trans;
cmd.CommandType = CommandType.Text;
cmd.CommandText = procCall;

// Register input-output and output parameters for the DB2Command
...

// Call the stored procedure
Console.WriteLine("  Call stored procedure named " + procName);
cmd.ExecuteNonQuery();
```

Calling a stored procedure by setting the CommandType property of the `DB2Command` to `CommandType.StoredProcedure` in Visual Basic .NET:

```
' assume a DB2DataReader reader
Dim trans As DB2Transaction = conn.BeginTransaction()
Dim cmd As DB2Command = conn.CreateCommand()
Dim procName As String = "INOUT_PARAM"
cmd.Transaction = trans
cmd.CommandType = CommandType.StoredProcedure
cmd.CommandText = procName

' Register input-output and output parameters for the DB2Command
...

' Call the stored procedure
Console.WriteLine("  Call stored procedure named " & procName)
cmd.ExecuteNonQuery()
```

Calling a stored procedure by setting the CommandType property of the `DB2Command` to `CommandType.Text` in Visual Basic .NET:

```
' assume a DB2DataReader reader
Dim trans As DB2Transaction = conn.BeginTransaction()
Dim cmd As DB2Command = conn.CreateCommand()
Dim procName As String = "INOUT_PARAM"
Dim procCall As String = "CALL INOUT_PARAM (?, ?, ?)"
cmd.Transaction = trans
cmd.CommandType = CommandType.Text
cmd.CommandText = procCall

' Register input-output and output parameters for the DB2Command
...

' Call the stored procedure
Console.WriteLine("  Call stored procedure named " & procName)
cmd.ExecuteNonQuery()
```

Chapter 3. Building applications for the DB2 .NET Data Provider

Building Visual Basic .NET applications

DB2 provides a batch file, bldapp.bat, for compiling and linking DB2 Visual Basic .NET applications, located in the sqllib\samples\.NET\vb directory, along with sample programs that can be built with this file. The batch file takes one parameter, %1, for the name of the source file to be compiled (without the .vb extension).

Procedure:

To build the program, `DbAuth`, from the source file, `DbAuth.vb`, enter:

```
bldapp DbAuth
```

To ensure you have the parameters you need when you run the executable, you can specify different combinations of parameters depending on the number entered:

1. No parameters. Enter just the program name:
   ```
   DbAuth
   ```
2. One parameter. Enter the program name plus the database alias:
   ```
   DbAuth <db_alias>
   ```
3. Two parameters. Enter the program name plus user ID and password:
   ```
   DbAuth <userid> <passwd>
   ```
4. Three parameters. Enter the program name plus the database alias, user ID, and password:
   ```
   DbAuth <db_alias> <userid> <passwd>
   ```
5. Four parameters. Enter the program name plus server name, port number, user ID, and password:
   ```
   DbAuth <server> <portnum> <userid> <passwd>
   ```
6. Five parameters. Enter the program name plus database alias, server name, port number, user ID, and password:
   ```
   DbAuth <db_alias> <server> <portnum> <userid> <passwd>
   ```

To build and run the `LCTrans` sample program, you need to follow more detailed instructions given in the source file, `LCTrans.vb`.

Related tasks:

- "Building Common Language Runtime (CLR) .NET routines" on page 72

Related reference:

- "Visual Basic .NET application compile and link options" on page 15
- "Visual Basic .NET samples" in *Samples Topics*

Related samples:

- "bldapp.bat -- Builds Visual Basic .Net applications on Windows"
- "DbAuth.vb -- How to Grant, display and revoke privileges on database"
- "LCTrans.vb -- Demonstrates loosely coupled transactions"

Building C# .NET applications

DB2 provides a batch file, bldapp.bat, for compiling and linking DB2 C# .NET applications, located in the sqllib\samples\.NET\cs directory, along with sample programs that can be built with this file. The batch file takes one parameter, %1, for the name of the source file to be compiled (without the .cs extension).

Procedure:

To build the program, `DbAuth`, from the source file, `DbAuth.cs`, enter:

```
bldapp DbAuth
```

To ensure you have the parameters you need when you run the executable, you can specify different combinations of parameters depending on the number entered:

1. No parameters. Enter just the program name:
   ```
   DbAuth
   ```
2. One parameter. Enter the program name plus the database alias:
   ```
   DbAuth <db_alias>
   ```
3. Two parameters. Enter the program name plus user ID and password:
   ```
   DbAuth <userid> <passwd>
   ```
4. Three parameters. Enter the program name plus the database alias, user ID, and password:
   ```
   DbAuth <db_alias> <userid> <passwd>
   ```
5. Four parameters. Enter the program name plus server name, port number, user ID, and password:
   ```
   DbAuth <server> <portnum> <userid> <passwd>
   ```
6. Five parameters. Enter the program name plus database alias, server name, port number, user ID, and password:
   ```
   DbAuth <db_alias> <server> <portnum> <userid> <passwd>
   ```

To build and run the `LCTrans` sample program, you need to follow more detailed instructions given in the source file, `LCTrans.cs`.

Related tasks:

- "Building Common Language Runtime (CLR) .NET routines" on page 72

Related reference:

- "C# .NET application compile and link options" on page 16
- "C# samples" in *Samples Topics*

Related samples:

- "bldapp.bat -- Builds C# applications on Windows"
- "DbAuth.cs -- How to Grant, display and revoke privileges on database"
- "LCTrans.cs -- Demonstrates loosely coupled transactions (CSNET)"

Visual Basic .NET application compile and link options

The following are the compile and link options recommended by DB2 for building Visual Basic .NET applications on Windows with the Microsoft® Visual Basic .NET compiler, as demonstrated in the `bldapp.bat` batch file.

Compile and link options for bldapp

Compile and link options for standalone VB .NET applications:

`%BLDCOMP%`
Variable for the compiler. The default is `vbc`, the Microsoft Visual Basic .NET compiler.

`/r:"%DB2PATH%"\bin\%VERSION%IBM.Data.DB2.dll`
Reference the DB2 data link library for the .NET framework version you are using.

`%VERSION%`
There are two supported versions of the .NET framework for applications. DB2 has a data link library for each in separate sub-directories. For .NET Framework Version 2.0, `%VERSION%` points to the `netf20\` sub-directory; For .NET Framework Version 1.1, `%VERSION%` points to the `netf11\` sub-directory.

Compile and link options for bldapp

Compile and link options for the loosely-coupled sample program, `LCTrans`:

`%BLDCOMP%`
: Variable for the compiler. The default is `vbc`, the Microsoft Visual Basic .NET compiler.

`/out:RootCOM.dll`
: Output the `RootCOM` data link library, used by the `LCTrans` application, from the `RootCOM.vb` source file,

`/out:SubCOM.dll`
: Output the `SubCOM` data link library, used by the `LCTrans` application, from the `SubCOM.vb` source file,

`/target:library %1.cs`
: Create the data link library from the input source file (`RootCOM.vb` or `SubCOM.vb`).

`/r:System.EnterpriseServices.dll`
: Reference the Microsoft Windows System EnterpriseServices data link library.

`/r:"%DB2PATH%"\bin\%VERSION%IBM.Data.DB2.dll`
: Reference the DB2 data link library for the .NET framework version you are using.

  `%VERSION%`
  : There are two supported versions of the .NET framework for applications. DB2 has a data link library for each in separate sub-directories. For .NET Framework Version 2.0, `%VERSION%` points to the `netf20` sub-directory; For .NET Framework Version 1.1, `%VERSION%` points to the `netf11` sub-directory.

  `/r:System.Data.dll`
  : Reference the Microsoft Windows System Data data link library.

  `/r:System.dll`
  : Reference the Microsoft Windows System data link library.

  `/r:System.Xml.dll`
  : Reference the Microsoft Windows System XML data link library (for `SubCOM.vb`).

  `/r:SubCOM.dll`
  : Reference the SubCOM data link library (for `RootCOM.vb` and `LCTrans.vb`).

  `/r:RootCOM.dll`
  : Reference the RootCOM data link library (for `LCTrans.vb`).

Refer to your compiler documentation for additional compiler options.

Related tasks:

- "Building Visual Basic .NET applications" on page 13

Related samples:

- "bldapp.bat -- Builds Visual Basic .Net applications on Windows"

C# .NET application compile and link options

The following are the compile and link options recommended by DB2 for building C# applications on Windows with the Microsoft C# compiler, as demonstrated in the `bldapp.bat` batch file.

Compile and link options for bldapp
Compile and link options for standalone C# applications: **`%BLDCOMP%`** Variable for the compiler. The default is `csc`, the Microsoft C# compiler. **`/r:"%DB2PATH%"\bin\%VERSION%IBM.Data.DB2.dll`** Reference the DB2 data link library for the .NET framework version you are using. **`%VERSION%`** There are two supported versions of the .NET framework for applications. DB2 has a data link library for each in separate sub-directories. For .NET Framework Version 2.0, `%VERSION%` points to the `netf20\` sub-directory; For .NET Framework Version 1.1, `%VERSION%` points to the `netf11\` sub-directory.
Compile and link options for the loosely-coupled sample program, `LCTrans`: **`%BLDCOMP%`** Variable for the compiler. The default is `csc`, the Microsoft C# compiler. **`/out:RootCOM.dll`** Output the `RootCOM` data link library, used by the `LCTrans` application, from the `RootCOM.cs` source file, **`/out:SubCOM.dll`** Output the `SubCOM` data link library, used by the `LCTrans` application, from the `SubCOM.cs` source file, **`/target:library %1.cs`** Create the data link library from the input source file (`RootCOM.cs` or `SubCOM.cs`). **`/r:System.EnterpriseServices.dll`** Reference the Microsoft Windows System EnterpriseServices data link library. **`/r:"%DB2PATH%"\bin\%VERSION%IBM.Data.DB2.dll`** Reference the DB2 data link library for the .NET framework version you are using. **`%VERSION%`** There are two supported versions of the .NET framework for applications. DB2 has a data link library for each in separate sub-directories. For .NET Framework Version 2.0, `%VERSION%` points to the `netf20` sub-directory; For .NET Framework Version 1.1, `%VERSION%` points to the `netf11` sub-directory. **`/r:System.Data.dll`** Reference the Microsoft Windows System Data data link library. **`/r:System.dll`** Reference the Microsoft Windows System data link library. **`/r:System.Xml.dll`** Reference the Microsoft Windows System XML data link library (for `SubCOM.cs`). **`/r:SubCOM.dll`** Reference the SubCOM data link library (for `RootCOM.cs` and `LCTrans.cs`). **`/r:RootCOM.dll`** Reference the RootCOM data link library (for `LCTrans.cs`). Refer to your compiler documentation for additional compiler options.

Related tasks:

- “Building C# .NET applications” on page 14

Related samples:

- “bldapp.bat -- Builds C# applications on Windows”

Chapter 4. Developing external routines

External routines

External routines are routines that have their logic implemented in a programming language application that resides outside of the database, in the file system of the database server. The association of the routine with the external code application is asserted by the specification of the EXTERNAL clause in the CREATE statement of the routine.

You can create external procedures, external functions, and external methods. Although they are all implemented in external programming languages, each routine functional type has different features. Before deciding to implement an external routine, it is important that you first understand what external routines are, and how they are implemented and used, by reading the topic, "Overview of external routines". With that knowledge you can then learn more about external routines from the topics targeted by the related links so that you can make informed decisions about when and how to use them in your database environment.

Related concepts:
- "OLE DB user-defined table functions" in *Developing SQL and External Routines*
- "Overview of external routines" on page 21
- "Overview of routines" in *Developing SQL and External Routines*
- "Support for external routine development in .NET CLR languages" on page 58
- "Support for external routine development in C" in *Developing SQL and External Routines*
- "Support for external routine development in C++" in *Developing SQL and External Routines*
- "Supported Java routine development software" in *Developing SQL and External Routines*
- ".NET common language runtime (CLR) routines" on page 57
- "COBOL procedures" in *Developing SQL and External Routines*
- "DB2GENERAL routines" in *Developing SQL and External Routines*
- "External routine features" on page 22
- "Java routines" in *Developing SQL and External Routines*
- "OLE automation routine design" in *Developing SQL and External Routines*

Related tasks:
- "Creating C and C++ routines" in *Developing SQL and External Routines*
- "Creating external routines" on page 55
- "Developing routines" in *Developing SQL and External Routines*

Related reference:
- "Support for external procedure development in COBOL" in *Developing SQL and External Routines*
- "CREATE FUNCTION (External Table) statement" in *SQL Reference, Volume 2*
- "CREATE PROCEDURE (External) statement" in *SQL Reference, Volume 2*

Benefits of using routines

The following benefits can be gained by using routines:

Encapsulate application logic that can be invoked from an SQL interface
In an environment containing many different client applications that have common requirements, the effective use of routines can simplify code reuse, code standardization, and code maintenance. If a particular aspect of common application behavior needs to be changed in an environment where routines are used, only the affected routine that encapsulates the behavior requires modification. Without routines, application logic changes are required in each application.

Enable controlled access to other database objects
Routines can be used to control access to database objects. A user might not have permission to generally issue a particular SQL statement, such as CREATE TABLE; however the user can be given permission to invoke routines that contain one or more specific implementations of the statement, thus simplifying privilege management through encapsulation of privileges.

Improve application performance by reducing network traffic
When applications run on a client computer, each SQL statement is sent separately from the client computer to the database server computer to be executed and each result set is returned separately. This can result in high levels of network traffic. If a piece of work can be identified that requires extensive database interaction and little user interaction, it makes sense to install this piece of work on the server to minimize the quantity of network traffic and to allow the work to be done on the more powerful database servers.

Allow for faster, more efficient SQL execution
Because routines are database objects, they are more efficient at transmitting SQL requests and data than client applications. Therefore, SQL statements executed within routines can perform better than if executed in client applications. Routines that are created with the NOT FENCED clause run in the same process as the database manager, and can therefore use shared memory for communication, which can result in improved application performance.

Allow the interoperability of logic implemented in different programming languages
Because code modules might be implemented by different programmers in different programming languages, and because it is generally desirable to reuse code when possible, DB2 routines support a high degree of interoperability.

- Client applications in one programming language can invoke routines that are implemented in a different programming language. For example C client applications can invoke .NET common language runtime routines.
- Routines can invoke other routines regardless of the routine type or routine implementation. For example a Java procedure can invoke an embedded SQL scalar function.
- Routines created in a database server on one operating system can be invoked from a DB2 client running on a different operating system.

The benefits described above are just some of the many benefits of using routines. Using routines can be beneficial to a variety of users including database administrators, database architects, and database application developers. For this reason there are many useful applications of routines that you might want to explore.

There are various kinds of routines that address particular functional needs and various routine implementations. The choice of routine type and implementation can impact the degree to which the above benefits are exhibited. In general, routines are a powerful way of encapsulating logic so that you can extend your SQL, and improve the structure, maintenance, and potentially the performance of your applications.

Related concepts:

- "User-defined table functions" in *Developing SQL and External Routines*
- "External scalar functions" on page 23
- "Routine invocation" in *Developing SQL and External Routines*

Related tasks:

- "Debugging routines" in *Developing SQL and External Routines*

Overview of external routines

Overview of external routines

External routines are characterized primarily by the fact that their routine logic is implemented in programming language code and not in SQL.

Before deciding to implement an external routine, it is important that you understand what external routines are, how they are implemented, and how they can be used. The following concept topics will help you get an understanding of external routines so that you can make informed decisions about when and how to use them in your database environment:

- "External routine features" on page 22
- External routine creation
- External routine library or class management
- Supported programming languages for external routine development
- 32-bit and 64-bit support for external routines
- External routine parameter styles
- Restrictions on external routines

Once you have an understanding of external routine concepts you might want to:

- "Creating external routines" on page 55

Related concepts:

- "32-bit and 64-bit support for external routines" on page 49
- "External function and method features" on page 23
- "External routine creation" on page 41
- "External routine features" on page 22
- "External routine library and class management" on page 45
- "External routine parameter styles" on page 42

- "External routines" on page 19
- "Performance of routines with 32-bit libraries on 64-bit database servers" on page 50
- "Restrictions on external routines" on page 52
- "Supported APIs and programming languages for external routine development" on page 34
- "XML data type support in external routines" on page 51

Related tasks:
- "Creating external routines" on page 55

External routine features

External routines provide support for most of the common routine features as well as support for additional features not supported by SQL routines. The following features are unique to external routines:

Access to files, data, and applications residing outside of the database
External routines can access and manipulate data or files that reside outside of the database itself. They can also invoke applications that reside outside of the database. The data, files, or applications might, for example, reside in the database server file system or within the available network.

Variety of external routine parameter style options
The implementation of external routines in a programming language can be done using a choice of parameter styles. Although there might be a preferred parameter style for a chosen programming language, there is sometimes choice. Some parameter styles provide support for the passing of additional database and routine property information to and from the routine in a structure named *dbinfo* structure that might be useful within the routine logic.

Preservation of state between external function invocations with a scratchpad
External user-defined functions provide support for state preservation between function invocations for a set of values. This is done with a structure called a *scratchpad*. This can be useful both for functions that return aggregated values and for functions that require initial setup logic such as initialization of buffers.

Call-types identify individual external function invocations
External user-defined functions are invoked multiple times for a set of values. Each invocation is identified with a call-type value that can be referenced within the function logic. For example there are special call-types for the first invocation of a function, for data fetching calls, and for the final invocation. Call-types are useful, because specific logic can be associated with a particular call-type.

Related concepts:
- "32-bit and 64-bit support for external routines" on page 49
- "External function and method features" on page 23
- "External routines" on page 19
- "Features of SQL procedures" in *Developing SQL and External Routines*
- "Overview of external routines" on page 21
- "Restrictions on external routines" on page 52
- "SQL in external routines" in *Developing SQL and External Routines*

- “XML data type support in external routines” on page 51

External function and method features

External function and method features

External functions and external methods provide support for functions that, for a given set of input data, might be invoked multiple times and produce a set of output values.

To learn more about the features of external functions and methods, see the following topics:
- “External scalar functions”
- “External scalar function and method processing model” on page 25
- “External table functions” on page 26
- “External table function processing model” on page 27
- “Table function execution model for Java” on page 28
- “Scratchpads for external functions and methods” on page 30
- “Scratchpads on 32-bit and 64-bit operating systems” on page 33

These features are unique to external functions and methods and do not apply to SQL functions and SQL methods.

Related concepts:
- “Overview of external routines” on page 21
- “Restrictions on external routines” on page 52
- “External scalar functions” on page 23
- “External routine features” on page 22
- “External routines” on page 19
- “External scalar function and method processing model” on page 25

External scalar functions

External scalar functions are scalar functions that have their logic implemented in an external programming language.

These functions can be developed and used to extend the set of existing SQL functions and can be invoked in the same manner as DB2 built-in functions such as LENGTH and COUNT. That is, they can be referenced in SQL statements wherever an expression is valid.

The execution of external scalar function logic takes place on the DB2 database server, however unlike built-in or user-defined SQL scalar functions, the logic of external functions can access the database server filesystem, perform system calls or access a network.

External scalar functions can read SQL data, but cannot modify SQL data.

External scalar functions can be repeatedly invoked for a single reference of the function and can maintain state between these invocations by using a scratchpad, which is a memory buffer. This can be powerful if a function requires some initial, but expensive, setup logic. The setup logic can be done on a first invocation using the scratchpad to store some values that can be accessed or updated in subsequent invocations of the scalar function.

Features of external scalar functions

- Can be referenced as part of an SQL statement anywhere an expression is supported.
- The output of a scalar function can be used directly by the invoking SQL statement.
- For external scalar user-defined functions, state can be maintained between the iterative invocations of the function by using a scratchpad.
- Can provide a performance advantage when used in predicates, because they are executed at the server. If a function can be applied to a candidate row at the server, it can often eliminate the row from consideration before transmitting it to the client machine, reducing the amount of data that must be passed from server to client.

Limitations

- Cannot do transaction management within a scalar function. That is, you cannot issue a COMMIT or a ROLLBACK within a scalar function.
- Cannot return result sets.
- Scalar functions are intended to return a single scalar value per set of inputs.
- External scalar functions are not intended to be used for a single invocation. They are designed such that for a single reference to the function and a given set of inputs, that the function be invoked once per input, and return a single scalar value. On the first invocation, scalar functions can be designed to do some setup work, or store some information that can be accessed in subsequent invocations. SQL scalar functions are better suited to functionality that requires a single invocation.
- In a single partition database external scalar functions can contain SQL statements. These statements can read data from tables, but cannot modify data in tables. If the database has more than one partition then there must be no SQL statements in an external scalar function. SQL scalar functions can contain SQL statements that read or modify data.

Common uses

- Extend the set of DB2 built-in functions.
- Perform logic inside an SQL statement that SQL cannot natively perform.
- Encapsulate a scalar query that is commonly reused as a subquery in SQL statements. For example, given a postal code, search a table for the city where the postal code is found.

Supported languages

- C
- C++
- Java
- OLE
- .NET common language runtime languages

Notes:

1. There is a limited capability for creating aggregate functions. Also known as column functions, these functions receive a set of like values (a column of data) and return a single answer. A user-defined aggregate function can only be created if it is sourced upon a built-in aggregate function. For example, if a distinct type `SHOESIZE` exists that is defined with base type INTEGER, you could define a function, `AVG(SHOESIZE)`, as an aggregate function sourced on the existing built-in aggregate function, `AVG(INTEGER)`.
2. You can also create function that return a row. These are known as row functions and can only be used as a transform function for structured types. The output of a row function is a single row.

Related concepts:

- "Benefits of using routines" on page 20
- "External function and method features" on page 23
- "External scalar function and method processing model" on page 25
- "OLE DB user-defined table functions" in *Developing SQL and External Routines*
- "Scratchpads for external functions and methods" on page 30
- "External table functions" on page 26

Related tasks:

- "Invoking scalar functions or methods" in *Developing SQL and External Routines*
- "Invoking user-defined table functions" in *Developing SQL and External Routines*

Related reference:

- "CREATE FUNCTION statement" in *SQL Reference, Volume 2*

External scalar function and method processing model

The processing model for methods and scalar UDFs that are defined with the FINAL CALL specification is as follows:

FIRST call
This is a special case of the NORMAL call, identified as FIRST to enable the function to perform any initial processing. Arguments are evaluated and passed to the function. Normally, the function will return a value on this call, but it can return an error, in which case no NORMAL or FINAL call is made. If an error is returned on a FIRST call, the method or UDF must clean up before returning, because no FINAL call will be made.

NORMAL call
These are the second through second-last calls to the function, as dictated by the data and the logic of the statement. The function is expected to return a value with each NORMAL call after arguments are evaluated and passed. If NORMAL call returns an error, no further NORMAL calls are made, but the FINAL call is made.

FINAL call
This is a special call, made at end-of-statement processing (or CLOSE of a cursor), provided that the FIRST call succeeded. No argument values are passed on a FINAL call. This call is made so that the function can clean up any resources. The function does not return a value on this call, but can return an error.

For methods or scalar UDFs not defined with FINAL CALL, only NORMAL calls are made to the function, which normally returns a value for each call. If a NORMAL call returns an error, or if the statement encounters another error, no more calls are made to the function.

Note: This model describes the ordinary error processing for methods and scalar UDFs. In the event of a system failure or communication problem, a call indicated by the error processing model cannot be made. For example, for a FENCED UDF, if the `db2udf` fenced process is somehow prematurely terminated, DB2 cannot make the indicated calls.

Related concepts:
- "External function and method features" on page 23
- "External scalar functions" on page 23

External table functions

A user-defined table function delivers a table to the SQL in which it is referenced. A table UDF reference is only valid in a FROM clause of a SELECT statement. When using table functions, observe the following:
- Even though a table function delivers a table, the physical interface between DB2 and the UDF is one-row-at-a-time. There are five types of calls made to a table function: OPEN, FETCH, CLOSE, FIRST, and FINAL. The existence of FIRST and FINAL calls depends on how you define the UDF. The same *call-type* mechanism that can be used for scalar functions is used to distinguish these calls.
- Not every result column defined in the RETURNS clause of the CREATE FUNCTION statement for the table function has to be returned. The DBINFO keyword of CREATE FUNCTION, and corresponding *dbinfo* argument enable the optimization that only those columns needed for a particular table function reference need be returned.
- The individual column values returned conform in format to the values returned by scalar functions.
- The CREATE FUNCTION statement for a table function has a CARDINALITY specification. This specification enables the definer to inform the DB2 optimizer of the approximate size of the result so that the optimizer can make better decisions when the function is referenced.

  Regardless of what has been specified as the CARDINALITY of a table function, exercise caution against writing a function with infinite cardinality, that is, a function that always returns a row on a FETCH call. There are many situations where DB2 expects the end-of-table condition, as a catalyst within its query processing. Using GROUP BY or ORDER BY are examples where this is the case. DB2 cannot form the groups for aggregation until end-of-table is reached, and it cannot sort until it has all the data. So a table function that never returns the end-of-table condition (SQL-state value '02000') can cause an infinite processing loop if you use it with a GROUP BY or ORDER BY clause.

Related concepts:
- "External function and method features" on page 23
- "External table function processing model" on page 27

Related reference:
- "CREATE FUNCTION statement" in *SQL Reference, Volume 2*

- “Passing arguments to C, C++, OLE, or COBOL routines” in *Developing SQL and External Routines*

External table function processing model

The processing model for table UDFs that are defined with the FINAL CALL specification is as follows:

FIRST call
: This call is made before the first OPEN call, and its purpose is to enable the function to perform any initial processing. The scratchpad is cleared prior to this call. Arguments are evaluated and passed to the function. The function does not return a row. If the function returns an error, no further calls are made to the function.

OPEN call
: This call is made to enable the function to perform special OPEN processing specific to the scan. The scratchpad (if present) is not cleared prior to the call. Arguments are evaluated and passed. The function does not return a row on an OPEN call. If the function returns an error from the OPEN call, no FETCH or CLOSE call is made, but the FINAL call will still be made at end of statement.

FETCH call
: FETCH calls continue to be made until the function returns the SQLSTATE value signifying end-of-table. It is on these calls that the UDF develops and returns a row of data. Argument values can be passed to the function, but they are pointing to the same values that were passed on OPEN. Therefore, the argument values might not be current and should not be relied upon. If you do need to maintain current values between the invocations of a table function, use a scratchpad. The function can return an error on a FETCH call, and the CLOSE call will still be made.

CLOSE call
: This call is made at the conclusion of the scan or statement, provided that the OPEN call succeeded. Any argument values will not be current. The function can return an error.

FINAL call
: The FINAL call is made at the end of the statement, provided that the FIRST call succeeded. This call is made so that the function can clean up any resources. The function does not return a value on this call, but can return an error.

For table UDFs not defined with FINAL CALL, only OPEN, FETCH, and CLOSE calls are made to the function. Before each OPEN call, the scratchpad (if present) is cleared.

The difference between table UDFs that are defined with FINAL CALL and those defined with NO FINAL CALL can be seen when examining a scenario involving a join or a subquery, where the table function access is the ″inner″ access. For example, in a statement such as:

```
SELECT x,y,z,... FROM table_1 as A,
  TABLE(table_func_1(A.col1,...)) as B
  WHERE...
```

In this case, the optimizer would open a scan of table_func_1 for each row of table_1. This is because the value of table_1’s col1, which is passed to table_func_1, is used to define the table function scan.

For NO FINAL CALL table UDFs, the OPEN, FETCH, FETCH, ..., CLOSE sequence of calls repeats for each row of table_1. Note that each OPEN call will get a clean scratchpad. Because the table function does not know at the end of each scan whether there will be more scans, it must clean up completely during CLOSE processing. This could be inefficient if there is significant one-time open processing that must be repeated.

FINAL CALL table UDFs, provide a one-time FIRST call, and a one-time FINAL call. These calls are used to amortize the expense of the initialization and termination costs across all the scans of the table function. As before, the OPEN, FETCH, FETCH, ..., CLOSE calls are made for each row of the outer table, but because the table function knows it will get a FINAL call, it does not need to clean everything up on its CLOSE call (and reallocate on subsequent OPEN). Also note that the scratchpad is not cleared between scans, largely because the table function resources will span scans.

At the expense of managing two additional call types, the table UDF can achieve greater efficiency in these join and subquery scenarios. Deciding whether to define the table function as FINAL CALL depends on how it is expected to be used.

Related concepts:
- "External function and method features" on page 23
- "Table function execution model for Java" on page 28
- "User-defined table functions" in *Developing SQL and External Routines*

Related reference:
- "CREATE FUNCTION (External Table) statement" in *SQL Reference, Volume 2*
- "CREATE FUNCTION (OLE DB External Table) statement" in *SQL Reference, Volume 2*
- "CREATE FUNCTION (SQL Scalar, Table, or Row) statement" in *SQL Reference, Volume 2*

Table function execution model for Java

For table functions written in Java and using PARAMETER STYLE DB2GENERAL, it is important to understand what happens at each point in DB2's processing of a given statement. The following table details this information for a typical table function. Covered are both the NO FINAL CALL and the FINAL CALL cases, assuming SCRATCHPAD in both cases.

Point in scan time	**NO FINAL CALL LANGUAGE JAVA SCRATCHPAD**	**FINAL CALL LANGUAGE JAVA SCRATCHPAD**
Before the first OPEN for the table function	No calls.	• Class constructor is called (means new scratchpad). UDF method is called with FIRST call. • Constructor initializes class and scratchpad variables. Method connects to Web server.

Point in scan time	NO FINAL CALL LANGUAGE JAVA SCRATCHPAD	FINAL CALL LANGUAGE JAVA SCRATCHPAD
At each OPEN of the table function	• Class constructor is called (means new scratchpad). UDF method is called with OPEN call. • Constructor initializes class and scratchpad variables. Method connect to Web server, and opens the scan for Web data.	• UDF method is opened with OPEN call. • Method opens the scan for whatever Web data it wants. (Might be able to avoid reopen after a CLOSE reposition, depending on what is saved in the scratchpad.)
At each FETCH for a new row of table function data	• UDF method is called with FETCH call. • Method fetches and returns next row of data, or EOT.	• UDF method is called with FETCH call. • Method fetches and returns new row of data, or EOT.
At each CLOSE of the table function	• UDF method is called with CLOSE call. `close()` method if it exists for class. • Method closes its Web scan and disconnects from the Web server. `close()` does not need to do anything.	• UDF method is called with CLOSE call. • Method might reposition to the top of the scan, or close the scan. It can save any state in the scratchpad, which will persist.
After the last CLOSE of the table function	No calls.	• UDF method is called with FINAL call. `close()` method is called if it exists for class. • Method disconnects from the Web server. `close()` method does not need to do anything.

Notes:

1. The term "UDF method" refers to the Java class method that implements the UDF. This is the method identified in the EXTERNAL NAME clause of the CREATE FUNCTION statement.
2. For table functions with NO SCRATCHPAD specified, the calls to the UDF method are as indicated in this table, but because the user is not asking for any continuity with a scratchpad, DB2 will cause a new object to be instantiated before each call, by calling the class constructor. It is not clear that table functions with NO SCRATCHPAD (and thus no continuity) can do useful things, but they are supported.

Related concepts:

- "DB2GENERAL routines" in *Developing SQL and External Routines*
- "External function and method features" on page 23
- "External table function processing model" on page 27
- "Java routines" in *Developing SQL and External Routines*

Related tasks:

- "Designing Java routines" in *Developing SQL and External Routines*

Related reference:

- "CREATE FUNCTION (External Table) statement" in *SQL Reference, Volume 2*

Scratchpads for external functions and methods

A *scratchpad* enables a user-defined function or method to save its state from one invocation to the next. For example, here are two situations where saving state between invocations is beneficial:

1. Functions or methods that, to be correct, depend on saving state.

   An example of such a function or method is a simple counter function that returns a '1' the first time it is called, and increments the result by one each successive call. Such a function could, in some circumstances, be used to number the rows of a SELECT result:

   ```
   SELECT counter(), a, b+c, ...
     FROM tablex
     WHERE ...
   ```

   The function needs a place to store the current value for the counter between invocations, where the value will be guaranteed to be the same for the following invocation. On each invocation, the value can then be incremented and returned as the result of the function.

   This type of routine is NOT DETERMINISTIC. Its output does not depend solely on the values of its SQL arguments.

2. Functions or methods where the performance can be improved by the ability to perform some initialization actions.

   An example of such a function or method, which might be a part of a document application, is a *match* function, which returns 'Y' if a given document contains a given string, and 'N' otherwise:

   ```
   SELECT docid, doctitle, docauthor
     FROM docs
     WHERE match('myocardial infarction', docid) = 'Y'
   ```

   This statement returns all the documents containing the particular text string value represented by the first argument. What *match* would like to do is:

   - First time only.

     Retrieve a list of all the document IDs that contain the string 'myocardial infarction' from the document application, that is maintained outside of DB2. This retrieval is a costly process, so the function would like to do it only one time, and save the list somewhere handy for subsequent calls.

   - On each call.

     Use the list of document IDs saved during the first call to see if the document ID that is passed as the second argument is contained in the list.

   This type of routine is DETERMINISTIC. Its answer only depends on its input argument values. What is shown here is a function whose performance, not correctness, depends on the ability to save information from one call to the next.

Both of these needs are met by the ability to specify a SCRATCHPAD in the CREATE statement:

```
CREATE FUNCTION counter()
  RETURNS int ... SCRATCHPAD;

CREATE FUNCTION match(varchar(200), char(15))
  RETURNS char(1) ... SCRATCHPAD 10000;
```

The SCRATCHPAD keyword tells DB2 to allocate and maintain a scratchpad for a routine. The default size for a scratchpad is 100 bytes, but you can determine the size (in bytes) for a scratchpad. The *match* example is 10000 bytes long. DB2 initializes the scratchpad to binary zeros before the first invocation. If the scratchpad is being defined for a table function, and if the table function is also defined with NO FINAL CALL (the default), DB2 refreshes the scratchpad before each OPEN call. If you specify the table function option FINAL CALL, DB2 does not examine or change the content of the scratchpad after its initialization. For scalar functions defined with scratchpads, DB2 also does not examine or change the scratchpad's content after its initialization. A pointer to the scratchpad is passed to the routine on each invocation, and DB2 preserves the routine's state information in the scratchpad.

So for the *counter* example, the last value returned could be kept in the scratchpad. And the *match* example could keep the list of documents in the scratchpad if the scratchpad is big enough, otherwise it could allocate memory for the list and keep the address of the acquired memory in the scratchpad. Scratchpads can be variable length: the length is defined in the CREATE statement for the routine.

The scratchpad only applies to the individual reference to the routine in the statement. If there are multiple references to a routine in a statement, each reference has its own scratchpad, thus scratchpads cannot be used to communicate between references. The scratchpad only applies to a single DB2 agent (an agent is a DB2 entity that performs processing of all aspects of a statement). There is no ″global scratchpad″ to coordinate the sharing of scratchpad information between the agents. This is especially important for situations where DB2 establishes multiple agents to process a statement (in either a single partition or multiple partition database). In these cases, even though there might only be a single reference to a routine in a statement, there could be multiple agents doing the work, and each would have its own scratchpad. In a multiple partition database, where a statement referencing a UDF is processing data on multiple partitions, and invoking the UDF on each partition, the scratchpad would only apply to a single partition. As a result, there is a scratchpad on each partition where the UDF is executed.

If the correct execution of a function depends on there being a single scratchpad per reference to the function, then register the function as DISALLOW PARALLEL. This will force the function to run on a single partition, thereby guaranteeing that only a single scratchpad will exist per reference to the function.

Because it is recognized that a UDF or method might require system resources, the UDF or method can be defined with the FINAL CALL keyword. This keyword tells DB2 to call the UDF or method at end-of-statement processing so that the UDF or method can release its system resources. It is vital that a routine free any resources it acquires; even a small leak can become a big leak in an environment where the statement is repetitively invoked, and a big leak can cause a DB2 crash.

Since the scratchpad is of a fixed size, the UDF or method can itself include a memory allocation and thus, can make use of the final call to free the memory. For example, the preceding *match* function cannot predict how many documents will match the given text string. So a better definition for *match* is:

```
CREATE FUNCTION match(varchar(200), char(15))
  RETURNS char(1) ... SCRATCHPAD 10000 FINAL CALL;
```

For UDFs or methods that use a scratchpad and are referenced in a subquery, DB2 might make a final call, if the UDF or method is so specified, and refresh the scratchpad between invocations of the subquery. You can protect yourself against this possibility, if your UDFs or methods are ever used in subqueries, by defining the UDF or method with FINAL CALL and using the call-type argument, or by always checking for the *binary zero* state of the scratchpad.

If you do specify FINAL CALL, note that your UDF or method receives a call of type FIRST. This could be used to acquire and initialize some persistent resource.

Following is a simple Java example of a UDF that uses a scratchpad to compute the sum of squares of entries in a column. This example takes in a column and returns a column containing the cumulative sum of squares from the top of the column to the current row entry:

```
CREATE FUNCTION SumOfSquares(INTEGER)
RETURNS INTEGER
EXTERNAL NAME 'UDFsrv!SumOfSquares'
DETERMINISTIC
NO EXTERNAL ACTION
FENCED
NOT NULL CALL
LANGUAGE JAVA
PARAMETER STYLE DB2GENERAL
NO SQL
SCRATCHPAD 10
FINAL CALL
DISALLOW PARALLEL
NO DBINFO@


// Sum Of Squares using Scratchpad UDF
public void SumOfSquares(int inColumn,
                         int outSum)
throws Exception
{
  int sum = 0;
  byte[] scratchpad = getScratchpad();

  // variables to read from SCRATCHPAD area
  ByteArrayInputStream byteArrayIn = new ByteArrayInputStream(scratchpad);
  DataInputStream dataIn = new DataInputStream(byteArrayIn);

  // variables to write into SCRATCHPAD area
  byte[] byteArrayCounter;
  int i;
  ByteArrayOutputStream byteArrayOut = new ByteArrayOutputStream(10);
  DataOutputStream dataOut = new DataOutputStream(byteArrayOut);

  switch(getCallType())
  {
    case SQLUDF_FIRST_CALL:
      // initialize data
   sum = (inColumn * inColumn);
      // save data into SCRATCHPAD area
      dataOut.writeInt(sum);
      byteArrayCounter = byteArrayOut.toByteArray();
      for(i = 0; i < byteArrayCounter.length; i++)
      {
        scratchpad[i] = byteArrayCounter[i];
      }
      setScratchpad(scratchpad);
   break;
    case SQLUDF_NORMAL_CALL:
      // read data from SCRATCHPAD area
      sum = dataIn.readInt();
      // work with data
      sum = sum + (inColumn * inColumn);
      // save data into SCRATCHPAD area
      dataOut.writeInt(sum);
      byteArrayCounter = byteArrayOut.toByteArray();
      for(i = 0; i < byteArrayCounter.length; i++)
      {
          scratchpad[i] = byteArrayCounter[i];
        }
        setScratchpad(scratchpad);
  break;
    }
    //set the output value
    set(2, sum);
  } // SumOfSquares UDF
```

Please note that there is a built-in DB2 function that performs the same task as the SumOfSquares UDF. This example was chosen to demonstrate the use of a scratchpad.

Related concepts:
- "Scratchpad as C or C++ function parameter" in *Developing SQL and External Routines*
- "Scratchpads on 32-bit and 64-bit operating systems" on page 33
- "External function and method features" on page 23
- "External scalar function and method processing model" on page 25

Related reference:
- "CREATE FUNCTION statement" in *SQL Reference, Volume 2*
- "CREATE METHOD statement" in *SQL Reference, Volume 2*
- "CREATE TYPE (Structured) statement" in *SQL Reference, Volume 2*

Related samples:
- "udfsrv.c -- Library of user-defined functions (UDFs) called by the client"
- "UDFCreate.db2 -- How to catalog the Java UDFs contained in UDFsrv.java "
- "UDFsrv.java -- Provide UDFs to be called by UDFcli.java (JDBC)"

Scratchpads on 32-bit and 64-bit operating systems

To make your UDF or method code portable between 32-bit and 64-bit operating systems, you must take care in the way you create and use scratchpads that contain 64-bit values. It is recommended that you do not declare an explicit length variable for a scratchpad structure that contains one or more 64-bit values, such as 64-bit pointers or `sqlint64` BIGINT variables.

Following is a sample structure declaration for a scratchpad:

```
struct sql_scratchpad
{
   sqlint32 length;
   char data[100];
};
```

When defining its own structure for the scratchpad, a routine has two choices:

1. Redefine the entire scratchpad `sql_scratchpad`, in which case it needs to include an explicit length field. For example:

```
   struct sql_spad
   {
     sqlint32 length;
     sqlint32 int_var;
     sqlint64 bigint_var;
   };
```

```
void SQL_API_FN routine( ..., struct sql_spad* scratchpad, ... )
{
  /* Use scratchpad */
}
```

2. Redefine just the data portion of the scratchpad `sql_scratchpad`, in which case no length field is needed.

```
struct spaddata
{
  sqlint32 int_var;
  sqlint64 bigint_var;
};
void SQL_API_FN routine( ..., struct sql_scratchpad* spad, ... )
{
  struct spaddata* scratchpad = (struct spaddata*)spad→data;
  /* Use scratchpad */
}
```

Since the application cannot change the value in the length field of the scratchpad, there is no significant benefit to coding the routine as shown in the first example. The second example is also portable between computers with different word sizes, so it is the preferred way of writing the routine.

Related concepts:
- "External function and method features" on page 23
- "Scratchpads for external functions and methods" on page 30
- "External scalar functions" on page 23
- "User-defined table functions" in *Developing SQL and External Routines*

Related tasks:
- "Invoking 32-bit routines on a 64-bit database server" in *Developing SQL and External Routines*

Supported APIs and programming languages

Supported APIs and programming languages for external routine development

You can develop DB2 external routines (procedures and functions) using the following APIs and associated programming languages:
- ADO.NET
  - .NET Common Language Runtime programming languages
- CLI
- Embedded SQL
  - C
  - C++
  - COBOL (Only supported for procedures)
- JDBC
  - Java
- OLE
  - Visual Basic
  - Visual C++
  - Any other programming language that supports this API.
- OLE DB (Only supported for table functions)

  - Any programming language that supports this API.
- SQLJ
  - Java

Related concepts:

- "Overview of external routines" on page 21
- "Support for external routine development in .NET CLR languages" on page 58
- "Support for external routine development in C" in *Developing SQL and External Routines*
- "Support for external routine development in C++" in *Developing SQL and External Routines*
- "Supported Java routine development software" in *Developing SQL and External Routines*
- "Choosing an application programming interface" in *Getting Started with Database Application Development*
- "Supported database application programming interfaces" in *Getting Started with Database Application Development*

Related reference:

- "Support for external procedure development in COBOL" in *Developing SQL and External Routines*
- "Supported programming languages and compilers for database application development" in *Getting Started with Database Application Development*

Comparison of supported APIs and programming languages for external routine development

It is important to consider the characteristics and limitations of the various supported external routine application programming interfaces (APIs) and programming languages before you start implementing external routines. This will ensure that you choose the right implementation from the start and that the routine features that you require are available.

Table 1. Comparison of external routine APIs and programming languages

API and programming language	Feature support	Performance	Security	Scalability	Limitations
SQL (includes SQL PL)	• SQL is a high level language that is easy to learn and use, which makes implementation go quickly. • SQL Procedural Language (SQL PL) elements allow for control-flow logic around SQL operations and queries. • Strong data type support.	• Very good. • SQL routines perform better than Java routines. • SQL routines perform as well as C and C++ external routines created with the NOT FENCED clause.	• Very safe. • SQL procedures run in the same memory as the database manager.	• Highly scalable.	• Cannot access the database server file system. • Cannot invoke applications that reside outside of the database.

Table 1. Comparison of external routine APIs and programming languages (continued)

API and programming language	Feature support	Performance	Security	Scalability	Limitations
Embedded SQL (includes C and C++)	• Low level, but powerful programming language.	• Very good. • C and C++ routines perform better than Java routines. • C and C++ routines created with the NOT FENCED clause perform as well as SQL routines.	• C and C++ routines are prone to programming errors. • Programmers must be proficient in C to avoid making common memory and pointer manipulation errors which make routine implementation more tedious and time consuming. • C and C++ routines should be created with the FENCED clause and the NOT THREADSAFE clause to avoid the disruption of the database manager should an exception occur in the routine at run time. These are default clauses. The use of these clauses can somewhat negatively impact performance, but ensure safe execution. See: Security of routines .	• Scalability is reduced when C and C++ routines are created with the FENCED and NOT THREADSAFE clauses. These routines are run in an isolated *db2fmp* process apart from the database manager process. One *db2fmp* process is required per concurrently executed routine.	• There are multiple supported parameter passing styles which can be confusing. Users should use parameter style SQL as much as possible.

Table 1. Comparison of external routine APIs and programming languages (continued)

API and programming language	Feature support	Performance	Security	Scalability	Limitations
Embedded SQL (COBOL)	• High-level programming language good for developing business, typically file oriented, applications. • Pervasively used in the past for production business applications, although its popularity is decreasing. • COBOL does not contain pointer support and is a linear iterative programming language.	• COBOL routines do not perform as well as routines created with any of the other external routine implementation options.	• No information at this time.	• No information at this time.	• You can create and invoke 32-bit COBOL procedures in 64-bit DB2 instances, however these routines will not perform as well as 64-bit COBOL procedures within a 64-bit DB2 instance.
JDBC (Java) and SQLJ (Java)	• High-level object-oriented programming language suitable for developing standalone applications, applets, and servlets. • Java objects and data types facilitate the establishment of database connections, execution of SQL statements, and manipulation of data.	• Java routines do not perform as well as C and C++ routines or SQL routines.	• Java routines are safer than C and C++ routines, because the control of dangerous operations is handled by the Java Virtual Machine (JVM). This increases reliability and makes it very difficult for the code of one Java routine to harm another routine running in the same process.	• Good scalability • Java routines created with the FENCED THREADSAFE clause (the default) scale well. All fenced Java routines will share a few JVMs. More than one JVM might be in use on the system if the Java heap of a particular db2fmp process is approaching exhaustion.	• To avoid potentially dangerous operations, Java Native Interface (JNI) calls from Java routines are not permitted.

Table 1. Comparison of external routine APIs and programming languages (continued)

API and programming language	Feature support	Performance	Security	Scalability	Limitations
.NET common language runtime supported languages (includes C#, Visual Basic, and others)	• Part of the Microsoft .NET model of managed code. • Source code is compiled into intermediate language (IL) byte code that can be interpreted by the Microsoft .NET Framework common language runtime. • CLR assemblies can be built up from sub-assemblies that were compiled from different .NET programming language source code, which allows users to re-use and integrate code modules written in various languages.	• CLR routines can only be created with the FENCED NOT THREADSAFE clause so as to minimize the possibility of database manager interruption at runtime. This can somewhat negatively impact performance	• CLR routines can only be created with the FENCED NOT THREADSAFE clause. They are therefore safe because they will be run outside of the database manager in a separate db2fmp process.	• No information available.	• Refer to the topic, ″Restrictions on .NET CLR routines″.

Table 1. Comparison of external routine APIs and programming languages (continued)

API and programming language	Feature support	Performance	Security	Scalability	Limitations
• OLE	• OLE routines can be implemented in Visual C++, Visual Basic, and other languages supported by OLE.	• The speed of OLE automated routines depends on the language used to implement them. In general they are slower than non-OLE C/C++ routines. • OLE routines can only run in FENCED NOT THREADSAFE mode, and therefore OLE automated routines do not scale well.	• No information available.	• No information available.	• No information available.

Table 1. Comparison of external routine APIs and programming languages (continued)

API and programming language	Feature support	Performance	Security	Scalability	Limitations
• OLE DB	• OLE DB can be used to create user-defined table functions. • OLE DB functions connect to external OLE DB data sources.	• Performance of OLE DB functions depends on the OLE DB provider, however in general OLE DB functions perform better than logically equivalent Java functions, but slower than logically equivalent C, C++, or SQL functions. However some predicates from the query where the function is invoked might be evaluated at the OLE DB provider, therefore reducing the number of rows that DB2 has to process which can frequently result in improved performance.	• No information available.	• No information available.	• OLE DB can only be used to create user-defined table functions.

Related concepts:

- "Supported database application programming interfaces" in *Getting Started with Database Application Development*
- "Support for external routine development in .NET CLR languages" on page 58
- "Support for external routine development in C" in *Developing SQL and External Routines*
- "Support for external routine development in C++" in *Developing SQL and External Routines*
- "Supported Java routine development software" in *Developing SQL and External Routines*
- "Supported APIs and programming languages for external routine development" on page 34

Related reference:

- "Supported programming languages and compilers for database application development" in *Getting Started with Database Application Development*
- "Support for external procedure development in COBOL" in *Developing SQL and External Routines*

External routine creation

External routines are created in a similar way as routines with other implementations. However there are a few additional steps required because the routine implementation requires the coding, compilation, and deployment of source code.

There are two parts to an external routine:
- The CREATE statement that defines the routine.
- The external library or class that implements the routine-body

Upon the successful execution of a CREATE statement that defines a routine, the routine is created within the database. The statement must at a minimum define the name of the routine, the routine parameter signature that will be used in the routine implementation, and the location of the external library or class built from the routine implementation source code.

External routine implementation must be coded in one of the supported programming languages and then built into a library or class file that must be installed in the file system of the database server.

An external routine cannot be successfully invoked until it has been created in the database and the library or class associated with the routine has been put in the location specified by the EXTERNAL clause.

The development of external routines generally consists of the following tasks:
- Determining what functional type of routine to implement.
- Choosing one of the supported external routine programming languages for the routine implementation.
- Designing the routine.
- Connecting to a database and creating the routine in the database.
  – This is done by executing one of the CREATE PROCEDURE, CREATE FUNCTION, or CREATE METHOD statements or by using a graphical tool that automates this step.
  – This task, also known as defining or registering a routine, can occur at any time before you invoke the routine, except in the following circumstances:
    - For Java routines that reference an external JAR file or files, the external code and JAR files must be coded and compiled before the routine is created in the database using the routine type specific CREATE statement.
    - Routines that execute SQL statements and refer to themselves directly must be created in the database by issuing the CREATE statement before the external code associated with the routine is precompiled and bound. This also applies to situations where there is a cycle of references, for example, Routine A references Routine B, which references Routine A.
- Coding the routine logic such that it corresponds to the routine definition.
- Building the routine and generating a library or class file.

  - For embedded SQL routines this includes: precompiling , compiling, and link the code as well as binding the routine package to the target database.
  - For non-embedded SQL routines this includes: compiling and linking the code.
- Deploying the library or class file to the database server in the location specified in the routine definition.
- Granting the EXECUTE privilege on the routine to the routine invoker or invokers (if they are not the routine definer).
- Invoking, testing, and debugging the routine.

The steps required to create an external routine can all be done using the DB2 Command Line Processor or a DB2 Command Window. Tools can be of assistance in automating some or all of these steps.

Related concepts:
- "External routine features" on page 22
- "External routines" on page 19
- "Overview of external routines" on page 21

Related tasks:
- "Creating external routines" on page 55

External routine parameter styles

External routine implementations must conform to a particular convention for the exchange of routine parameter values. These conventions are known as *parameter styles*. An external routine parameter style is specified when the routine is created by specifying the PARAMETER STYLE clause. Parameter styles characterize the specification and order in which parameter values will be passed to the external routine implementation. They also specify what if any additional values will be passed to the external routine implementation. For example, some parameter styles specify that for each routine parameter value that an additional separate null-indicator value be passed to the routine implementation to provide information about the parameters nullability which cannot otherwise be easily determined with a native programming language data type.

The table below provides a list of the available parameter styles, the routine implementations that support each parameter style, the functional routine types that support each parameter style, and a description of the parameter style:

Table 2. Parameter styles

Parameter style	Supported language	Supported routine type	Description
SQL [1]	• C/C++ • OLE • .NET common language runtime languages • COBOL [2]	• UDFs • stored procedures • methods	In addition to the parameters passed during invocation, the following arguments are passed to the routine in the following order: • A null indicator for each parameter or result declared in the CREATE statement. • The SQLSTATE to be returned to DB2. • The qualified name of the routine. • The specific name of the routine. • The SQL diagnostic string to be returned to DB2. Depending on options specified in the CREATE statement and the routine type, the following arguments can be passed to the routine in the following order: • A buffer for the scratchpad. • The call type of the routine. • The dbinfo structure (contains information about the database).
DB2SQL [1]	• C/C++ • OLE • .NET common language runtime languages • COBOL	• stored procedures	In addition to the parameters passed during invocation, the following arguments are passed to the stored procedure in the following order: • A vector containing a null indicator for each parameter on the CALL statement. • The SQLSTATE to be returned to DB2. • The qualified name of the stored procedure. • The specific name of the stored procedure. • The SQL diagnostic string to be returned to DB2. If the DBINFO clause is specified in the CREATE PROCEDURE statement, a dbinfo structure (it contains information about the database) is passed to the stored procedure.
JAVA	• Java	• UDFs • stored procedures	PARAMETER STYLE JAVA routines use a parameter passing convention that conforms to the Java language and SQLJ Routines specification. For stored procedures, INOUT and OUT parameters will be passed as single entry arrays to facilitate the returning of values. In addition to the IN, OUT, and INOUT parameters, Java method signatures for stored procedures include a parameter of type ResultSet[] for each result set specified in the DYNAMIC RESULT SETS clause of the CREATE PROCEDURE statement. For PARAMETER STYLE JAVA UDFs and methods, no additional arguments to those specified in the routine invocation are passed. PARAMETER STYLE JAVA routines do not support the DBINFO or PROGRAM TYPE clauses. For UDFs, PARAMETER STYLE JAVA can only be specified when there are no structured data types specified as parameters and no structured type, CLOB, DBCLOB, or BLOB data types specified as return types (SQLSTATE 429B8). Also, PARAMETER STYLE JAVA UDFs do not support table functions, call types, or scratchpads.
DB2GENERAL	• Java	• UDFs • stored procedures • methods	This type of routine will use a parameter passing convention that is defined for use with Java methods. Unless you are developing table UDFs, UDFs with scratchpads, or need access to the dbinfo structure, it is recommended that you use PARAMETER STYLE JAVA. For PARAMETER STYLE DB2GENERAL routines, no additional arguments to those specified in the routine invocation are passed.

Table 2. Parameter styles (continued)

Parameter style	Supported language	Supported routine type	Description
GENERAL	• C/C++ • .NET common language runtime languages • COBOL	• stored procedures	A PARAMETER STYLE GENERAL stored procedure receives parameters from the CALL statement in the invoking application or routine. If the DBINFO clause is specified in the CREATE PROCEDURE statement, a dbinfo structure (it contains information about the database) is passed to the stored procedure. GENERAL is the equivalent of SIMPLE stored procedures for DB2 Universal Database for z/OS and OS/390.
GENERAL WITH NULLS	• C/C++ • .NET common language runtime languages • COBOL	• stored procedures	A PARAMETER STYLE GENERAL WITH NULLS stored procedure receives parameters from the CALL statement in the invoking application or routine. Also included is a vector containing a null indicator for each parameter on the CALL statement. If the DBINFO clause is specified in the CREATE PROCEDURE statement, a dbinfo structure (it contains information about the database) is passed to the stored procedure. GENERAL WITH NULLS is the equivalent of SIMPLE WITH NULLS stored procedures for DB2 Universal Database for z/OS and OS/390.

Note:

1. For UDFs and methods, PARAMETER STYLE SQL is equivalent to PARAMETER STYLE DB2SQL.
2. COBOL can only be used to develop stored procedures.
3. .NET common language runtime methods are not supported.

Related concepts:

- "DB2GENERAL routines" in *Developing SQL and External Routines*
- "Java routines" in *Developing SQL and External Routines*
- "Overview of external routines" on page 21
- "Parameters in .NET CLR routines" in *Developing SQL and External Routines*
- "Parameters in C and C++ routines" in *Developing SQL and External Routines*
- "Parameters in Java routines" in *Developing SQL and External Routines*
- "Parameters in SQL procedures" in *Developing SQL and External Routines*

Related tasks:

- "Distinct types as UDF or method parameters" in *Developing SQL and External Routines*
- "Passing structured type parameters to external routines" in *Developing SQL and External Routines*

Related reference:

- "CREATE FUNCTION statement" in *SQL Reference, Volume 2*
- "CREATE METHOD statement" in *SQL Reference, Volume 2*
- "CREATE PROCEDURE statement" in *SQL Reference, Volume 2*
- "CREATE TYPE (Structured) statement" in *SQL Reference, Volume 2*
- "Passing arguments to C, C++, OLE, or COBOL routines" in *Developing SQL and External Routines*

Routine library or class management

External routine library and class management

To successfully develop and invoke external routines, external routine library and class files must be deployed and managed properly.

External routine library and class file management can be minimal if care is taken when external routines are first created and library and class files deployed.

The main external routine management considerations are the following:
- Deployment of external routine library and class files
- Security of external routine libary and class files
- Resolution of external routine libraries and classes
- Modifications to external routine library and class files
- Backup and restore of external routine library and class files

System administrators, database administrators and database application developers should all take responsibility to ensure that external routine library and class files are secure and correctly preserved during routine development and database administration tasks.

Related concepts:
- "Backup and restore of external routine library and class files" on page 48
- "Deployment of external routine libraries and classes" on page 45
- "Overview of external routines" on page 21
- "Resolution of external routine libraries and classes" on page 47
- "Restrictions on external routines" on page 52
- "Security of external routine library or class files" on page 46
- "Modifications to external routine library and class files" on page 48

Related reference:
- "ALTER FUNCTION statement" in *SQL Reference, Volume 2*
- "ALTER METHOD statement" in *SQL Reference, Volume 2*
- "ALTER PROCEDURE statement" in *SQL Reference, Volume 2*
- "CREATE FUNCTION statement" in *SQL Reference, Volume 2*
- "CREATE METHOD statement" in *SQL Reference, Volume 2*
- "CREATE PROCEDURE statement" in *SQL Reference, Volume 2*

Deployment of external routine libraries and classes

Deployment of external routine libraries and classes refers to the copying of external routine libraries and classes to the database server once they have been built from source code.

External routine library, class, or assembly files must be copied into the DB2 *function directory* or a sub-directory of this directory on the database server. This is the recommended external routine deployment location. To find out more about the function directory, see the description of the EXTERNAL clause for either of the following SQL statements: CREATE PROCEDURE or CREATE FUNCTION.

You can copy the external routine class, library, or assembly to other directory locations on the server, depending on the API and programming language used to implement the routine, however this is generally discouraged. If this is done, to successfully invoke the routine you must take particular note of the fully qualified path name and ensure that this value is used with the EXTERNAL NAME clause.

Library and class files can be copied to the database server file system using most generally available file transfer tools. Java routines can be copied from a computer where a DB2 client is installed to a DB2 database server using special system-defined procedures designed specifically for this purpose. See the topics on Java routines for more details.

When executing the appropriate SQL language CREATE statement for the routine type: CREATE PROCEDURE or CREATE FUNCTION, be sure to specify the appropriate clauses, paying particular attention to the EXTERNAL NAME clause.

- Specify the `LANGUAGE` clause with the appropriate value for the chosen API or programming language. Examples include: CLR, C, JAVA.
- Specify the `PARAMETER STYLE` clause with the name of the supported parameter style that was implemented in the routine code.
- Specify the `EXTERNAL` clause with the name of the library, class, or assembly file to be associated with the routine using one of the following values:
  - the fully qualified path name of the routine library, class, or assembly file.
  - the relative path name of the routine library, class, or assembly file relative to the function directory.

By default DB2 will look for the library, class, or assembly file by name in the function directory unless a fully qualified or relative path name for it is specified in the EXTERNAL clause.

Related concepts:

- "External routine library management and performance" on page 49
- "Modifications to external routine library and class files" on page 48
- "Resolution of external routine libraries and classes" on page 47
- "Security of external routine library or class files" on page 46
- "Backup and restore of external routine library and class files" on page 48
- "External routine library and class management" on page 45

Security of external routine library or class files

External routine libraries are stored in the file system on the database server and are not backed up or protected in any way by the DB2 database manager. For routines to continue to successfully be invoked, it is imperative that the library associated with the routine continue to exist in the location specified in the EXTERNAL clause of the CREATE statement used to create the routine. Do not move or delete routine libraries after creating routines; doing so will cause routine invocations to fail.

To prevent routine libraries from being accidentally or intentionally deleted or replaced, you must restrict access to the directories on the database server that contain routine libraries and restrict access to the routine library files. This can be done by using operating system commands to set directory and file permissions.

Related concepts:

- "Backup and restore of external routine library and class files" on page 48
- "External routine library and class management" on page 45
- "External routine library management and performance" on page 49
- "Modifications to external routine library and class files" on page 48

Resolution of external routine libraries and classes

DB2 external routine library resolution is performed at the DB2 instance level. This means that in DB2 instances containing multiple DB2 databases, external routines can be created in one database that use external routine libraries already being used for a routine in another database.

Instance level external routine resolution supports code re-use by allowing multiple routine definitions to be associated with a single library. When external routine libraries are not re-used in this way, and instead copies of the external routine library exist in the file system of the database server, library name conflicts can happen. This can specifically happen when there are multiple databases in a single instance and the routines in each database are associated with their own copies of libraries and classes of routine bodies. A conflict arises when the name of a library or class used by a routine in one database is identical to the name of a library or class used by a routine in another database (in the same instance).

To minimize the likelihood of this happening, it is recommended that a single copy of a routine library be stored in the instance level function directory (sqllib/function directory) and that the EXTERNAL clause of all of the routine definitions in each of the databases reference the unique library.

If two functionally different routine libraries must be created with the same name, it is important to take additional steps to minimize the likelihood of library name conflicts.

For C, C++, COBOL, and ADO.NET routines:

Library name conflicts can be minimized or resolved by:

1. Storing the libraries with routine bodies in separate directories for each database.
2. Creating the routines with an EXTERNAL NAME clause value that specifies the full path of the given library (instead of a relative path).

For Java routines:

Class name conflicts cannot be resolved by moving the class files in question into different directories, because the CLASSPATH environment variable is instance-wide. The first class encountered in the CLASSPATH is the one that is used. Therefore, if you have two different Java routines that reference a class with the same name, one of the routines will use the incorrect class. There are two possible solutions: either rename the affected classes, or create a separate instance for each database.

Related concepts:

- "Backup and restore of external routine library and class files" on page 48
- "External routine library and class management" on page 45
- "External routine library management and performance" on page 49
- "Modifications to external routine library and class files" on page 48
- "Security of external routine library or class files" on page 46

Modifications to external routine library and class files

Modifications to an existing external routine's logic might be necessary after an external routine has been deployed and it is in use in a production database system environment. Modifications to existing routines can be made, but it is important that they be done carefully so as to define a clear takeover point in time for the updates and to minimize the risk of interrupting any concurrent invocations of the routine.

If an external routine library requires an update, do not recompile and relink the routine to the same target file (for example, `sqllib/function/foo.a`) that the current routine is using while the database manager is running. If a current routine invocation is accessing a cached version of the routine process and the underlying library is replaced, this can cause the routine invocation to fail. If it is necessary to change the body of a routine without stopping and restarting DB2, complete the following steps:

1. Create the new external routine library with a different library or class file name.
2. If it is an embedded SQL routine, bind the routine package to the database using the BIND command.
3. Use the ALTER ROUTINE statement to change the routine definition so that the EXTERNAL NAME clause references the updated routine library or class. If the routine body to be updated is used by routines cataloged in multiple databases, the actions prescribed in this section must be completed for each affected database.
4. For updating Java routines that are built into JAR files, you must issue a CALL SQLJ.REFRESH_CLASSES() statement to force DB2 to load the new classes. If you do not issue the CALL SQLJ.REFRESH_CLASSES() statement after you update Java routine classes, DB2 continues to use the previous versions of the classes. DB2 refreshes the classes when a COMMIT or ROLLBACK occurs.

Once the routine definition has been updated, all subsequent invocations of the routine will load and run the new external routine library or class.

Related concepts:

- "Backup and restore of external routine library and class files" on page 48
- "External routine library and class management" on page 45
- "External routine library management and performance" on page 49
- "Security of external routine library or class files" on page 46

Backup and restore of external routine library and class files

External routine libraries are not backed up with other database objects when a database backup is performed. They are similarly not restored when a database is restored.

If the purpose of a database backup and restore is to re-deploy a database, then external routine library files must be copied from the original database server file system to the target database server file system in such a way as to preserve the relative path names of the external routine libraries.

Related concepts:

- "External routine library and class management" on page 45
- "External routine library management and performance" on page 49

- “Modifications to external routine library and class files” on page 48
- “Security of external routine library or class files” on page 46

External routine library management and performance

External routine library management can impact routine performance, because the DB2 database manager dynamically caches external routine libraries in an effort to improve performance in accordance with routine usage. For optimal external routine performance consider the following:

- Keep the number of routines in each library as small as possible. It is better to have numerous small external routine libraries than a few large ones.
- Group together within source code the routine functions of routines that are commonly invoked together. When the code is compiled into an external routine library the entry points of commonly invoked routines will be closer together which allows the database manager to provide better caching support. The improved caching support is due to the efficiency that can be gained by loading a single external routine libary once and then invoking multiple external routine functions within that library.

  For external routines implemented in the C or C++ programming language, the cost of loading a library is paid only once for libraries that are consistently in use by C routines. After a routine is invoked once, all subsequent invocations from the same thread in the process, do not need to re-load the routine’s library.

Related concepts:

- “Backup and restore of external routine library and class files” on page 48
- “External routine library and class management” on page 45
- “Modifications to external routine library and class files” on page 48

Related tasks:

- “Replacing an AIX shared library” in *Developing SQL and External Routines*

32-bit and 64-bit support for external routines

Support for 32-bit and 64-bit external routines is determined by the specification of one of the following two clauses in the CREATE statement for the routines: FENCED clause or NOT FENCED clause.

The routine-body of an external routine is written in a programming language and compiled into a library or class file that is loaded and run when the routine is invoked. The specification of the FENCED or NOT FENCED clause determines whether the external routine runs in a fenced environment distinct from the database manager or in the same addressing space as the database manager which can yield better performance through the use of shared memory instead of TCPIP for communications. By default routines are always created as fenced regardless of the other clauses selected.

The following table illustrates DB2’s support for running fenced and unfenced 32-bit and 64-bit routines on 32-bit and 64-bit database servers that are running the same operating system.

Table 3. Support for 32-bit and 64-bit external routines

Bit-width of routine	32-bit server	64-bit server
32-bit fenced procedure or UDF	Supported	Supported

Table 3. Support for 32-bit and 64-bit external routines (continued)

Bit-width of routine	32-bit server	64-bit server
64-bit fenced procedure or UDF	Not supported (4)	Supported
32-bit unfenced procedure or UDF	Supported	Supported (2)
64-bit unfenced procedure or UDF	Not supported (4)	Supported

The footnotes in the table above correspond to:

- (1) Running a 32-bit routine on a 64-bit server is not as fast as running a 64-bit routine on a 64-bit server.
- (2) 32-bit routines must be created as FENCED and NOT THREADSAFE to work on a 64-bit server.
- (3) It is not possible to invoke 32-bit routines on Linux IA 64-bit database servers.
- (4) 64-bit applications and routines cannot be run in 32-bit addressing spaces.

The important thing to note in the table is that 32-bit unfenced procedures cannot run on a 64-bit DB2 server. If you must deploy 32-bit unfenced routines to 64-bit platforms, remove the NOT FENCED clause from the CREATE statements for these routines before you catalog them.

Related concepts:

- "External routine library and class management" on page 45
- "External routines" on page 19
- "Overview of external routines" on page 21
- "External routine creation" on page 41
- "External routine features" on page 22
- "External routine implementation" in *Developing SQL and External Routines*

Related tasks:

- "Creating external routines" on page 55

Performance of routines with 32-bit libraries on 64-bit database servers

It is possible to invoke routines with 32-bit routine libraries on 64-bit DB2 database servers. However, this does not perform as well as invoking a 64-bit routine on a 64-bit server. The reason for the performance degradation is that before each attempt to execute a 32-bit routine on a 64-bit server, an attempt is first made to invoke it as a 64-bit library. If this fails, the library is then invoked as a 32-bit library. A failed attempt to invoke a 32-bit library as a 64-bit library produces an error message (SQLCODE -444) in the db2diag.log.

Java classes are bit-width independent. Only Java virtual machines (JVMs) are classified as 32-bit or 64-bit. DB2 only supports the use of JVMs that are the same bit width as the instance in which they are used. In other words, in a 32-bit DB2 instance only a 32-bit JVM can be used, and in a 64-bit DB2 instance only a 64-bit JVM can be used. This ensures proper functioning of Java routines and the best possible performance.

Related concepts:

- "Overview of external routines" on page 21

XML data type support in external routines

External procedures and functions written in the following programming languages support parameters and variables of data type XML:
- C
- C++
- COBOL
- Java
- .NET CLR languages

External OLE and OLEDB routines do not support parameters of data type XML.

XML data type values are represented in external routine code in the same way as CLOB data types.

When declaring external routine parameters of data type XML, the CREATE PROCEDURE and CREATE FUNCTION statements that will be used to create the routines in the database must specify that the XML data type is to be stored as a CLOB data type. The size of the CLOB value should be close to the size of the XML document represented by the XML parameter.

The CREATE PROCEDURE statement below shows a CREATE PROCEDURE statement for an external procedure implemented in the C programming language with an XML parameter named `parm1`:

```
CREATE PROCEDURE myproc(IN parm1 XML AS CLOB(2M), IN parm2 VARCHAR(32000))
LANGUAGE C
FENCED
PARAMETER STYLE SQL
EXTERNAL NAME 'mylib!myproc';
```

Similar considerations apply when creating external UDFs, as shown in the example below:

```
CREATE FUNCTION myfunc (IN parm1 XML AS CLOB(2M))
RETURNS SMALLINT
LANGUAGE C
PARAMETER STYLE SQL
DETERMINISTIC
NOT FENCED
NULL CALL
NO SQL
NO EXTERNAL ACTION
EXTERNAL NAME 'mylib1!myfunc'
```

XML values are serialized before they are passed to external routines.

Within external routine code, XML parameter and variable values are accessed, set, and modified in the same way as in database applications.

Related concepts:
- "Overview of external routines" on page 21
- "Encoding considerations for passing XML data in routine parameters" in *XML Guide*

- "Parameters and variables of data type XML in SQL functions" in *Developing SQL and External Routines*
- "XML and XQuery support in SQL procedures" in *Developing SQL and External Routines*

Related tasks:

- "Specifying a column data type" in *Developing SQL and External Routines*

Related reference:

- "Supported SQL data types for the DB2 .NET Data Provider" on page 60
- "Supported SQL data types in DB2GENERAL routines" in *Developing SQL and External Routines*
- "Supported SQL data types in Java routines" in *Developing SQL and External Routines*

Restrictions on external routines

The following restrictions apply to external routines and should be considered when developing or debugging external routines.

Restrictions that apply to all external routines::

- New threads cannot be created in external routines.
- Connection level APIs cannot be called from within external functions or external methods.
- Receiving inputs from the keyboard and displaying outputs to standard output is not possible from external routines. Do not use standard input-output streams. For example:
  - In external Java routine code, do not issue the `System.out.println()` methods.
  - In external C or C++ routine code, do not issue `printf()`.
  - In external COBOL routine code, do not issue `display`

  Although external routines cannot display data to standard output, they can include code that writes data to a file on the database server file system.

  For fenced routines that run in UNIX environments, the target directory where the file is to be created, or the file itself, must have the appropriate permissions such that the owner of the `sqllib/adm/.fenced` file can create it or write to it. For not fenced routines, the instance owner must have create, read, and write permissions for the directory in which the file is opened.

  Note: DB2 does not attempt to synchronize any external input or output performed by a routine with DB2's own transactions. So, for example, if a UDF writes to a file during a transaction, and that transaction is later backed out for some reason, no attempt is made to discover or undo the writes to the file.
- Connection-related statements or commands cannot be executed in external routines. This restriction applies to the following statements: including:
  - BACKUP
  - CONNECT
  - CONNECT TO
  - CONNECT RESET
  - CREATE DATABASE

  - DROP DATABASE
  - FORWARD RECOVERY
  - RESTORE
- Operating system function usage within routines is not recommended. The use of these functions is not restricted except in the following cases:
  - **User-defined signal handlers must not be installed for external routines. Failure to adhere to this restriction can result in unexpectedexternal routine run-time failures, database abends, or other problems. Installing signal handlers can also interfere with operation of the JVM for Java routines.**
  - System calls that terminate a process can abnormally terminate one of DB2's processes and result in database system or database application failure.

    Other system calls can also cause problems if they interfere with the normal operation of the DB2; database manager. For example, a function that attempts to unload a library containing a user-defined function from memory could cause severe problems. Be careful in coding and testing external routines containing system calls.
- External routines must not contain commands that would terminate the current process. An external routine must always return control to the DB2 database manager without terminating the current process.
- External routine libraries, classes, or assemblies must not be updated while the database is active except in special cases. If an update is required while the DB2 database manager is active, and stopping and starting the instance is not an option, create the new library, class, or assembly for the routine with a different. Then, use the ALTER statement to change the external routine's EXTERNAL NAME clause value so that it references the name of the new library, class, or assembly file.
- Environment variable `DB2CKPTR` is not available in external routines. All other environment variables with names beginning with 'DB2' are captured at the time the database manager is started and are available for use in external routines.
- Some environment variables with names that do not start with 'DB2' are not available to external routines that are fenced. For example, the `LIBPATH` environment variable is not available for use. However these variables are available to external routines that are not fenced.
- Environment variable values that were set after the DB2 database manager is started are not available to external routines.
- Use of protected resources, resources that can only be accessed by one process at a time, within external routines should be limited. If used, try to reduce the likelihood of deadlocks when two external routines try to access the protected resource. If two or more external routines deadlock while attempting to access the protected resource, the DB2 database manager will not be able to detect or resolve the situation. This will result in hung external routine processes.
- Memory for external routine parameters should not be explicitly allocated on the DB2 database server. The DB2 database manager automatically allocates storage based upon the parameter declaration in the CREATE statement for the routine. Do not alter any storage pointers for parameters in external routines. Attempting to change a pointer with a locally created storage pointer can result in memory leaks, data corruption, or abends.
- Do not use static or global data in external routines. DB2 cannot guarantee that the memory used by static or global variables will be untouched between external routine invocations. For UDFs and methods, you can use scratchpads to store values for use between invocations.

- All SQL parameter values are buffered. This means that a copy of the value is made and passed to the external routine. If there are changes made to the input parameters of an external routine, these changes will have no effect on SQL values or processing. However, if an external routine writes more data to an input or output parameter than is specified by the CREATE statement, memory corruption has occurred, and the routine can abend.

Restrictions that apply to external procedures only:

- When returning result sets from nested stored procedures, you can open a cursor with the same name on multiple nesting levels. However, pre-version 8 applications will only be able to access the first result set that was opened. This restriction does not apply to cursors that are opened with a different package level.

Restrictions that apply to external functions only:

- External functions cannot return result sets. All cursors opened within an external function must be closed by the time the final-call invocation of the function completes.
- Dynamic allocations of memory in an external routine should be freed before the external routine returns. Failure to do so will result in a memory leak and the continuous growth in memory consumption of a DB2 process that could result in the database system running out of memory.

  For external user-defined functions and external methods, scratchpads can be used to allocate dynamic memory required for multiple function invocations. When scratchpads are used in this way, specify the FINAL CALL attribute in the CREATE FUNCTION or CREATE METHOD statement statement. This ensures that allocated memory is freed before the routine returns.

Related concepts:

- "Overview of external routines" on page 21
- "SQL data type handling in C and C++ routines" in *Developing SQL and External Routines*

Related reference:

- "Supported SQL data types in OLE automation" in *Developing SQL and External Routines*
- "Supported SQL data types in OLE DB" in *Developing SQL and External Routines*
- "Data type mappings between DB2 and OLE DB" on page 127
- "Supported SQL data types in C and C++ embedded SQL applications" in *Developing Embedded SQL Applications*
- "ALTER FUNCTION statement" in *SQL Reference, Volume 2*
- "ALTER METHOD statement" in *SQL Reference, Volume 2*
- "ALTER PROCEDURE statement" in *SQL Reference, Volume 2*
- "CALL statement" in *SQL Reference, Volume 2*
- "CREATE FUNCTION statement" in *SQL Reference, Volume 2*
- "CREATE METHOD statement" in *SQL Reference, Volume 2*
- "CREATE PROCEDURE statement" in *SQL Reference, Volume 2*

Creating external routines

External routines including procedures and functions are created in a similar way as routines with other implementations, however there are a few more steps required, because the routine implementation requires the coding, compilation, and deployment of source code.

You would choose to implement an external routine if:

- You want to encapsulate complex logic into a routine that accesses the database or that performs an action outside of the database.
- You require the encapsulated logic to be invoked from any of: multiple applications, the CLP, another routine (procedure, function (UDF), or method), or a trigger.
- You are most comfortable coding this logic in a programming language rather than using SQL and SQL PL statements.
- You require the routine logic to perform operations external to the database such as writing or reading to a file on the database server, the running of another application, or logic that cannot be represented with SQL and SQL PL statements.

Prerequisites:

- Knowledge of external routine implementation. To learn about external routines in general, see the topic:
  - "External routines" on page 19
  - "External routine creation" on page 41
- The DB2 Client must be installed.
- The database server must be running an operating system that supports the chosen implementation programming language compilers and development software.
- The required compilers and runtime support for the chosen programming language must be installed on the database server
- Authority to execute the CREATE PROCEDURE, CREATE FUNCTION, or CREATE METHOD statement.

Restrictions:

For a list of restrictions associated with external routines see:

- "Restrictions on external routines" on page 52

Procedure:

1. Code the routine logic in the chosen programming language.
   - For general information about external routines, routine features, and routine feature implementation, see the topics referenced in the Prerequisites section.
   - Use or import any required header files required to support the execution of SQL statemets.
   - Declare variables and parameters correctly using programming language data types that map to DB2 SQL data types.
2. Parameters must be declared in accordance with the format required by the parameter style for the chosen programming language. For more on parameters and prototype declarations see:
   - "External routine parameter styles" on page 42

3. Build your code into a library or class file.
4. Copy the library or class file into the DB2 *function directory* on the database server. It is recommended that you store assemblies or libraries associated with DB2 routines in the function directory. To find out more about the function directory, see the EXTERNAL clause of either of the following statements: CREATE PROCEDURE or CREATE FUNCTION.

   You can copy the assembly to another directory on the server if you wish, but to successfully invoke the routine you must note the fully qualified path name of your assembly as you will require it for the next step.
5. Execute either dynamically or statically the appropriate SQL language CREATE statement for the routine type: CREATE PROCEDURE or CREATE FUNCTION.
   - Specify the `LANGUAGE` clause with the appropriate value for the chosen API or programming language. Examples include: CLR, C, JAVA.
   - Specify the `PARAMETER STYLE` clause with the name of the supported parameter style that was implemented in the routine code.
   - Specify the `EXTERNAL` clause with the name of the library, class, or assembly file to be associated with the routine using one of the following values:
     – the fully qualified path name of the routine library, class, or assembly file .
     – the relative path name of the routine library, class, or assembly file relative to the function directory.

     By default DB2 will look for the library, class, or assembly file by name in the function directory unless a fully qualified or relative path name for it is specified in the EXTERNAL clause.
   - Specify DYNAMIC RESULT SETS with a numeric value if your routine is a procedure and it will return one or more result sets to the caller.
   - Specify any other clauses required to characterize the routine.

To invoke your external routine, see Routine invocation

Related concepts:
- "External routines" on page 19
- "External routine creation" on page 41
- "Restrictions on external routines" on page 52
- "External routine parameter styles" on page 42
- "Routine invocation" in *Developing SQL and External Routines*
- "Overview of external routines" on page 21

Chapter 5. .NET common language runtime (CLR) routines

.NET common language runtime (CLR) routines

In DB2, a common language runtime (CLR) routine is an external routine created by executing a CREATE PROCEDURE or CREATE FUNCTION statement that references a .NET assembly as its external code body.

The following terms are important in the context of CLR routines:

.NET Framework
A Microsoft application development environment comprised of the CLR and .NET Framework class library designed to provide a consistent programming environment for developing and integrating code pieces.

Common language runtime (CLR)
The runtime interpreter for all .NET Framework applications.

intermediate language (IL)
Type of compiled byte-code interpreted by the .NET Framework CLR. Source code from all .NET compatible languages compiles to IL byte-code.

assembly
A file that contains IL byte-code. This can either be a library or an executable.

You can implement CLR routines in any language that can be compiled into an IL assembly. These languages include, but are not limited to: Managed C++, C#, Visual Basic, and J#.

Before developing a CLR routine, it is important to both understand the basics of routines and the unique features and characteristics specific to CLR routines. To learn more about routines and CLR routines see:

- "Benefits of using routines" on page 20
- "Supported SQL data types for the DB2 .NET Data Provider" on page 60
- "Parameters in .NET CLR routines" on page 62
- "Returning result sets from .NET CLR procedures" on page 65
- "Restrictions on .NET CLR routines" on page 67
- "Errors related to .NET CLR routines" on page 81

Developing a CLR routine is easy. For step-by-step instructions on how to develop a CLR routine and complete examples see:

- "Creating .NET CLR routines from DB2 Command Windows" on page 69
- "Examples of C# .NET CLR procedures" on page 94
- "Examples of C# .NET CLR functions" on page 116

Related concepts:

- "Authorizations and binding of routines that contain SQL" in *Developing SQL and External Routines*
- "Benefits of using routines" on page 20
- "External routine parameter styles" on page 42

- “External routines” on page 19
- “SQL in external routines” in *Developing SQL and External Routines*

Related tasks:
- “Building Common Language Runtime (CLR) .NET routines” on page 72
- “Building .NET CLR routine code” on page 72
- “Creating .NET CLR routines” on page 68
- “Debugging .NET CLR routines” on page 79
- “Examples of .NET CLR routines” on page 84

Related samples:
- “SpCreate.db2 -- Creates the external procedures implemented in spserver.cs”
- “SpServer.cs -- C# external code implementation of procedures created in spcat.db2”
- “SpCreate.db2 -- Creates the external procedures implemented in spserver.vb”
- “SpServer.vb -- VB.NET implementation of procedures created in SpCat.db2”

Supported .NET CLR routine development software

Support for external routine development in .NET CLR languages

To develop external routines in .NET CLR languages and successfully run them, you will need to use supported operating systems, versions of DB2 database servers and clients, and development software.

Supported operating systems for developing .NET CLR routines with .NET Framework 1.1 or .NET Framework 2.0:
- Windows 2000
- Windows XP (32-bit edition)
- Windows Server 2003 (32-bit edition)

Supported DB2 database servers and clients for .NET CLR routine development:

The following minimum DB2 database servers and clients must be installed:
- DB2 server: Minimum supported version is DB2 Version 8.2.
- DB2 client: Minimum supported version is DB2 Version 7.2.

Required development software for .NET CLR routines:

One of the following two software products must be installed on the same computer as the DB2 database server:
- Microsoft .NET Framework, Version 1.1
- Microsoft .NET Framework, Version 2.0

The Microsoft .NET Framework is independently available or as part of one of the following development kits:
- Microsoft .NET Framework Version 1.1 Software Development Kit
- Microsoft .NET Framework Version 2.0 Software Development Kit

.NET CLR external routines can be implemented in any language that can be compiled into an IL assembly by the Microsoft .NET Framework. These languages include, but are not limited to: Managed C++, C#, Visual Basic, and J#.

Related concepts:
- "Supported .NET development software" on page 2
- ".NET common language runtime (CLR) routines" on page 57
- "Tools for developing .NET CLR routines" on page 59

Related tasks:
- "Building Common Language Runtime (CLR) .NET routines" on page 72

Related reference:
- "Supported programming languages and compilers for database application development" in *Getting Started with Database Application Development*

Tools for developing .NET CLR routines

Tools can make the task of developing .NET CLR routines that interact with DB2 database faster and easier.

.NET CLR routines can be developed in Microsoft Visual Studio .NET using graphical tools available in:
- IBM® DB2 Development Add-In for Microsoft Visual Studio .NET 1.2

The following command line interfaces, provided with DB2, are also available for developing .NET CLR routines on DB2 database servers:
- DB2 Command Line Processor (DB2 CLP)
- DB2 Command Windows

Related concepts:
- "DB2 integration in Visual Studio" on page 5
- "Support for external routine development in .NET CLR languages" on page 58
- "Tools for developing routines" in *Developing SQL and External Routines*

Designing .NET CLR routines

Designing .NET CLR routines

When designing .NET CLR routines, you should take into account both general external routine design considerations and .NET CLR specific design considerations.

Prerequisites:

Knowledge and experience with .NET application development and general knowledge of external routines. The following topics can provide you with some of the required prerequisite information.

For more information on the features and uses of external routines see:
- External routines

For more information on the characteristics of .NET CLR routines, see:
- .NET CLR routines

Procedure:

With the prerequisite knowledge, designing embedded SQL routines consists mainly of learning about the unique features and characteristics of .NET CLR routines:
- Include assemblies that provide support for SQL statement execution in .NET CLR routines (IBM.Data.DB2)
- Supported SQL data types in .NET CLR routines
- Parameters to .NET CLR routines
- Returning result sets from .NET CLR routines
- Security and execution control mode settings for .NET CLR routines
- Restrictions on .NET CLR routines
- Returning result sets from .NET CLR procedures

After having learned about the .NET CLR characteristics, you might want to: ″Create .NET CLR routines″.

Related concepts:
- ".NET common language runtime (CLR) routines" on page 57
- "Parameters in .NET CLR routines" on page 62
- "Security and execution modes for CLR routines" on page 66

Related tasks:
- "Returning result sets from .NET CLR procedures" on page 65

Related reference:
- "Restrictions on .NET CLR routines" on page 67
- "Supported SQL data types for the DB2 .NET Data Provider" on page 60

Supported SQL data types for the DB2 .NET Data Provider

The following table lists the mappings between the `DB2Type` data types in the DB2 .NET Data Provider, the DB2 data type, and the corresponding .NET Framework data type:

Table 4. Mapping DB2 Data Types to .NET data types

DB2Type Enum	DB2 Data Type	.NET Data Type
SmallInt	SMALLINT	Int16
Integer	INTEGER	Int32
BigInt	BIGINT	Int64
Real	REAL	Single
Real370(2)	REAL	Single
Double	DOUBLE PRECISION	Double
Float	FLOAT	Double
Decimal	DECIMAL	Decimal
Numeric	DECIMAL	Decimal

Table 4. Mapping DB2 Data Types to .NET data types (continued)

DB2Type Enum	DB2 Data Type	.NET Data Type
Date	DATE	DateTime
Time	TIME	TimeSpan
Timestamp	TIMESTAMP	DateTime
Char	CHAR	String
VarChar	VARCHAR	String
LongVarChar(1)	LONG VARCHAR	String
Binary	CHAR FOR BIT DATA	Byte[]
VarBinary	VARCHAR FOR BIT DATA	Byte[]
LongVarBinary(1)	LONG VARCHAR FOR BIT DATA	Byte[]
Graphic	GRAPHIC	String
VarGraphic	VARGRAPHIC	String
LongVarGraphic(1)	LONG GRAPHIC	String
Clob	CLOB	String
Blob	BLOB	Byte[]
DbClob	DBCLOB(N)	String
Xml(3)	XML	IBM.Data.DB2Types.DB2Xml

Notes:

1. These data types are not supported in DB2 .NET common language runtime routines. They are only supported in client applications.
2. A DB2Parameter.Value property of the type DB2Type.Xml can accept variables of the following types: String, Byte[], DB2Xml, and XmlReader.
3. The Real370 enumeration can only be used for parameters in SQL statements that are executed against DB2 UDB for OS/390 and z/OS databases.

Note: The `dbinfo` structure is passed into CLR functions and procedures as a parameter. The scratchpad and call type for CLR UDFs are also passed into CLR routines as parameters. For information about the appropriate CLR data types for these parameters, see the related topic:

- Parameters in CLR routines

Related concepts:

- ".NET common language runtime (CLR) routines" on page 57
- "Designing .NET CLR routines" on page 59
- "External routine parameter styles" on page 42
- "Parameters in .NET CLR routines" on page 62

Related tasks:

- "Creating .NET CLR routines from DB2 Command Windows" on page 69
- "Examples of C# .NET CLR functions" on page 116
- "Examples of C# .NET CLR procedures" on page 94
- "Passing structured type parameters to external routines" in *Developing SQL and External Routines*

Related samples:

- "SpCreate.db2 -- Creates the external procedures implemented in spserver.cs"
- "SpServer.cs -- C# external code implementation of procedures created in spcat.db2"
- "SpCreate.db2 -- Creates the external procedures implemented in spserver.vb"
- "SpServer.vb -- VB.NET implementation of procedures created in SpCat.db2"

Parameters in .NET CLR routines

Parameter declaration in .NET CLR routines must conform to the requirements of one of the supported parameter styles, and must respect the parameter keyword requirements of the particular .NET language used for the routine. If the routine is to use a scratchpad, the `dbinfo` structure, or to have a PROGRAM TYPE MAIN parameter interface, there are additional details to consider. This topic addresses all CLR parameter considerations.

Supported parameter styles for CLR routines:

The parameter style of the routine must be specified at routine creation time in the EXTERNAL clause of the CREATE statement for the routine. The parameter style must be accurately reflected in the implementation of the external CLR routine code. The following DB2 parameter styles are supported for CLR routines:

- `SQL` (Supported for procedures and functions)
- `GENERAL` (Supported for procedures only)
- `GENERAL WITH NULLS` (Supported for procedures only)
- `DB2SQL` (Supported for procedures and functions)

For more information about these parameter styles see the topic:

- Parameter styles for external routines

CLR routine parameter null indicators:

If the parameter style chosen for a CLR routine requires that null indicators be specified for the parameters, the null indicators are to be passed into the CLR routine as System.Int16 type values, or in a System.Int16[] value when the parameter style calls for a vector of null indicators.

When the parameter style dictates that the null indicators be passed into the routine as distinct parameters, as is required for parameter style SQL, one System.Int16 null indicator is required for each parameter.

In .NET languages distinct parameters must be prefaced with a keywors to indicate if the parameter is passed by value or by reference. The same keyword that is used for a routine parameter must be used for the associated null indicator parameter. The keywords used to indicate whether an argument is passed by value or by reference are discussed in more detail below.

For more information about parameter style SQL and other supported parameter styles, see the topic:

- Parameter styles for external routines

Passing CLR routine parameters by value or by reference:

.NET language routines that compile into intermediate language (IL) byte-code require that parameters be prefaced with keywords that indicate the particular properties of the parameter such as whether the parameter is passed by value, by reference, is an input only, or an output only parameter.

Parameter keywords are .NET language specific. For example to pass a parameter by reference in `C#`, the parameter keyword is `ref`, whereas in `Visual Basic`, a by reference parameter is indicated by the `byRef` keyword. The keywords must be used to indicate the SQL parameter usage (`IN`, `OUT`, `INOUT`) that was specified in the CREATE statement for the routine.

The following rules apply when applying parameter keywords to .NET language routine parameters in DB2 routines:

- `IN` type parameters must be declared *without* a parameter keyword in C#, and must be declared with the `byVal` keyword in `Visual Basic`.
- `INOUT` type parameters must be declared with the language specific keyword that indicates that the parameter is passed by reference. In `C#` the appropriate keyword is `ref`. In `Visual Basic`, the appropriate keyword is `byRef`.
- `OUT` type parameters must be declared with the language specific keyword that indicates that the parameter is an output only parameter. In `C#`, use the `out` keyword. In Visual Basic, the parameter must be declared with the `byRef` keyword. Output only parameters must always be assigned a value before the routine returns to the caller. If the routine does not assign a value to an output only parameter, an error will be raised when the .NET routine is compiled.

Here is what a C#, parameter style SQL procedure prototype looks like for a routine that returns a single output parameter `language`.

```
public static void Counter  (out String language,
                             out Int16  languageNullInd,
                             ref String sqlState,
                                 String funcName,
                                 String funcSpecName,
                             ref String sqlMsgString,
                                 Byte[] scratchPad,
                                 Int32  callType);
```

It is clear that the parameter style SQL is implemented because of the extra null indicator parameter, `languageNullInd` associated with the output parameter `language`, the parameters for passing the SQLSTATE, the routine name, the routine specific name, and optional user-defined SQL error message. Parameter keywords have been specified for the parameters as follows:

- In C# no parameter keyword is required for input only parameters.
- In C# the 'out' keyword indicates that the variable is an output parameter only, and that its value has not been initialized by the caller.
- In C# the 'ref' keyword indicates that the parameter was initialized by the caller, and that the routine can optionally modify this value.

See the .NET language specific documentation regarding parameter passing to learn about the parameter keywords in that language.

Note:

DB2 controls allocation of memory for all parameters and maintains CLR references to all parameters passed into or out of a routine.

No parameter marker is required for procedure result sets:

No parameter markers is required in the procedure declaration of a procedure for a result set that will be returned to the caller. Any cursor statement that is not closed from inside of a CLR stored procedure will be passed back to its caller as a result set.

For more on result sets in CLR routines, see:
- "Returning result sets from .NET CLR procedures" on page 65

Dbinfo structure as CLR parameter:

The `dbinfo` structure used for passing additional database information parameters to and from a routine is supported for CLR routines through the use of an IL `dbinfo` class. This class contains all of the elements found in the `C` language `sqludf_dbinfo` structure except for the length fields associated with the strings. The length of each string can be found using the .NET language `Length` property of the particular string.

To access the `dbinfo` class, simply include the `IBM.Data.DB2` assembly in the file that contains your routine, and add a parameter of type `sqludf_dbinfo` to your routine's signature, in the position specified by the parameter style used.

UDF scratchpad as CLR parameter:

If a scratchpad is requested for a user defined function, it is passed into the routine as a `System.Byte[]` parameter of the specified size.

CLR UDF call type or final call parameter:

For user-defined functions that have requested a final call parameter or for table functions, the call type parameter is passed into the routine as a `System.Int32` data type.

PROGRAM TYPE MAIN supported for CLR procedures:

Program type MAIN is supported for .NET CLR procedures. Procedures defined as using Program Type MAIN must have the following signature:

```
void functionname(Int32 NumParams, Object[] Params)
```

Related concepts:
- "Designing .NET CLR routines" on page 59
- "External routine parameter styles" on page 42
- "Parameter handling in PROGRAM TYPE MAIN or PROGRAM TYPE SUB procedures" in *Developing SQL and External Routines*
- "Procedure parameter modes" in *Developing SQL and External Routines*
- "Scratchpads for external functions and methods" on page 30

Related tasks:
- "Examples of C# .NET CLR functions" on page 116
- "Examples of C# .NET CLR procedures" on page 94
- "Returning result sets from .NET CLR procedures" on page 65

- “Passing structured type parameters to external routines” in *Developing SQL and External Routines*

Related reference:

- “Supported SQL data types for the DB2 .NET Data Provider” on page 60

Returning result sets from .NET CLR procedures

You can develop CLR procedures that return result sets to a calling routine or application. Result sets cannot be returned from CLR functions (UDFs).

The .NET representation of a result set is a `DB2DataReader` object which can be returned from one of the various execute calls of a `DB2Command` object. Any `DB2DataReader` object whose `Close()` method has not explicitly been called prior to the return of the procedure, can be returned. The order in which result sets are returned to the caller is the same as the order in which the `DB2DataReader` objects were instantiated. No additional parameters are required in the function definition in order to return a result set.

Prerequisites:

An understanding of how to create CLR routines will help you to follow the steps in the procedure below for returning results from a CLR procedure.

- “Creating .NET CLR routines from DB2 Command Windows” on page 69

Procedure:

To return a result set from a CLR procedure:

1. In the CREATE PROCEDURE statement for the CLR routine you must specify along with any other appropriate clauses, the DYNAMIC RESULT SETS clause with a value equal to the number of result sets that are to be returned by the procedure.
2. No parameter marker is required in the procedure declaration for a result set that is to be returned to the caller.
3. In the .NET language implementation of your CLR routine, create a `DB2Connection` object, a `DB2Command` object, and a `DB2Transaction` object. A `DB2Transaction` object is responsible for rolling back and committing database transactions.
4. Initialize the Transaction property of the `DB2Command` object to the `DB2Transaction` object.
5. Assign a string query to the `DB2Command` object’s `CommandText` property that defines the result set that you want to return.
6. Instantiate a `DB2DataReader`, and assign to it, the result of the invocation of the `DB2Command` object method `ExecuteReader`. The result set of the query will be contained in the `DB2DataReader` object.
7. Do not execute the `Close()` method of the `DB2DataReader` object at any point prior to the procedure’s return to the caller. The still open `DB2DataReader` object will be returned as a result set to the caller.

   When more than one `DB2DataReader` is left open upon the return of a procedure, the `DB2DataReaders` are returned to the caller in the order of their creation. Only the number of result sets specified in the CREATE PROCEDURE statement will be returned to the caller.

8. Compile your .NET CLR language procedure and install the assembly in the location specified by the EXTERNAL clause in the CREATE PROCEDURE statement. Execute the CREATE PROCEDURE statement for the CLR procedure, if you have not already done so.
9. Once the CLR procedure assembly has been installed in the appropriate location and the CREATE PROCEDURE statement has successfully been executed, you can invoke the procedure with the CALL statement to see the result sets return to the caller.

   For information on calling procedures and other types of routines, see the topic:

   •

Related concepts:
- ".NET common language runtime (CLR) routines" on page 57
- "Designing .NET CLR routines" on page 59
- "Procedure parameter modes" in *Developing SQL and External Routines*

Related tasks:
- "Creating .NET CLR routines from DB2 Command Windows" on page 69

Security and execution modes for CLR routines

As a database administrator or application developer, you might want to protect the assemblies associated with your DB2 external routines from unwelcome tampering to restrict the actions of routines at run time. DB2 .NET common language runtime (CLR) routines support the specification of an execution control mode that identifies what types of actions a routine will be allowed to perform at run time. At run time, DB2 can detect if the routine attempts to perform actions beyond the scope of its specified execution control mode, which can be helpful when determining whether an assembly has been compromised.

To set the execution control mode of a CLR routine, specify the optional EXECUTION CONTROL clause in the CREATE statement for the routine. Valid modes are:
- SAFE
- FILEREAD
- FILEWRITE
- NETWORK
- UNSAFE

To modify the execution control mode in an existing CLR routine, execute the ALTER PROCEDURE or ALTER FUNCTION statement.

If the EXECUTION CONTROL clause is not specified for a CLR routine, by default the CLR routine is run using the most restrictive execution control mode: SAFE. Routines that are created with this execution control mode can only access resources that are controlled by the database manager. Less restrictive execution control modes allow a routine to access files (FILEREAD or FILEWRITE) or perform network operations such as accessing a webbpage (NETWORK). The execution control mode UNSAFE specifies that no restrictions are to be placed on the behavior of the routine. Routines defined with UNSAFE execution control mode can execute binary code.

These modes represent a hierarchy of allowable actions, and a higher-level mode includes the actions that are allowed below it in the hierarchy. For example, execution control mode NETWORK allows a routine to access web pages on the internet, read and write to files, and access resources that are controlled by the database manager. It is recommended to use the most restrictive execution control mode possible, and to avoid using the UNSAFE mode.

If DB2 detects at run time that a CLR routine is attempting an action outside of the scope of its execution control mode, DB2 will return error (SQLSTATE 38501).

The EXECUTION CONTROL clause can only be specified for LANGUAGE CLR routines. The scope of applicability of the EXECUTION CONTROL clause is limited to the .NET CLR routine itself, and does not extend to any other routines that it might call.

Refer to the syntax of the CREATE statement for the appropriate routine type for a full description of the supported execution control modes.

Related concepts:

- "Designing .NET CLR routines" on page 59

Restrictions on .NET CLR routines

The general implementation restrictions that apply to all external routines or particular routine classes (procedure or UDF) also apply to CLR routines. There are some restrictions that are particular to CLR routines. These restrictions are listed here.

The CREATE METHOD statement with LANGUAGE CLR clause is not supported:

You cannot create external methods for DB2 structured types that reference a CLR assembly. The use of a CREATE METHOD statement that specifies the LANGUAGE clause with value CLR is not supported.

CLR procedures cannot be implemented as NOT FENCED procedures:

CLR procedures cannot be run as unfenced procedures. The CREATE PROCEDURE statement for a CLR procedure can not specify the NOT FENCED clause.

EXECUTION CONTROL clause restricts the logic contained in the routine:

The EXECUTION CONTROL clause and associated value determine what types of logic and operations can be executed in a .NET CLR routine. By default the EXECUTOIN CONTROL clause value is set to SAFE. For routine logic that reads files, writes to files, or that accesses the internet, a non-default and less restrictive value for the EXECUTION CONTROL clause must be specified.

Maximum decimal precision is 29, maximum decimal scale is 28 in a CLR routine:

The DECIMAL data type in DB2 is represented with a precision of 31 digits and a scale of 28 digits. The .NET CLR System.Decimal data type is limited to a precision of 29 digits and a scale of 28 digits. Therefore, DB2 external CLR routines must not assign a value to a System.Decimal data type that has a value greater than `(2^96)-1`, which is the highest value that can be represented using a 29 digit precision and 28 digit scale. DB2 will raise a runtime error (SQLSTATE 22003, SQLCODE -413) if such an assignment occurs. At the time of execution of the CREATE statement for the routine, if a DECIMAL data type parameter is defined with a scale greater than 28, DB2 will raise an error (SQLSTATE 42613, SQLCODE -628).

If you require your routine to manipulate decimal values with the maximum precision and scale supported by DB2, you can implement your external routine in a different programming language such as Java.

Data types not supported in CLR routines:

The following DB2 SQL data types are not supported in CLR routines:
- LONG VARCHAR
- LONG VARCHAR FOR BIT DATA
- LONG GRAPHIC
- DATALINK
- ROWID

Running a 32-bit CLR routine on a 64-bit instance:

CLR routines cannot be run on 64- bit instances, because the .NET Framework cannot be installed on 64-bit operating systems at this time.

.NET CLR not supported for implementing security plug-ins:

The .NET CLR is not supported for compiling and linking source code for security plug-in libraries.

Related concepts:
- ".NET common language runtime (CLR) routines" on page 57
- "Designing .NET CLR routines" on page 59
- "Parameters in .NET CLR routines" on page 62

Related tasks:
- "Creating .NET CLR routines from DB2 Command Windows" on page 69
- "Returning result sets from .NET CLR procedures" on page 65

Creating .NET CLR routines

Creating .NET CLR routines

Creating .NET CLR routines consists of:
- Executing a CREATE statement that defines the routine in a DB2 database server
- Developing the routine implementation that corresponds to the routine definition

The ways in which you can create .NET CLR routines follow:
- Using the graphical tools provided with the DB2 Database Development Add-In for Visual Studio .NET 1.2

- Using the DB2 Command Window

In general it is easiest to create .NET CLR routines using the DB2 Database Development Add-In for Visual Studio .NET 1.2. If this is not available for use, the DB2 Command Window provides similar support through a command line interface.

Prerequisites:

- Review the .NET CLR Routine Overview.
- Ensure that you have access to a DB2 Version 9 server, including instances and databases.
- Ensure that the operating system is at a version level that is supported by DB2 database products.
- Ensure that the Microsoft .NET development software is at a version level that is supported for .NET CLR routine development.
- Authority to execute the CREATE PROCEDURE or CREATE FUNCTION statement.

Restrictions:

For a list of restrictions associated with CLR routines see:

- "Restrictions on .NET CLR routines" on page 67

Procedure:

.NET CLR routines from one of the following interfaces:

- Visual Studio .NET when the IBM DB2 Development Add-In for Microsoft Visual Studio .NET 1.2 is also installed. When the Add-In is installed, graphical tools integrated into Visual Studio .NET are available for creating .NET CLR routines that work in DB2 database servers.
- DB2 Command Windows

To create .NET CLR routines from DB2 Command Windows, see:

- Creating .NET CLR routines from DB2 Command Windows

Related concepts:

- "Support for external routine development in .NET CLR languages" on page 58
- ".NET common language runtime (CLR) routines" on page 57

Related tasks:

- "Creating .NET CLR routines from DB2 Command Windows" on page 69

Related reference:

- "Restrictions on .NET CLR routines" on page 67

Creating .NET CLR routines from DB2 Command Windows

Procedures and functions that reference an intermediate language assembly are created in the same way as any external routine is created. You would choose to implement an external routine in a .NET language if:

- You want to encapsulate complex logic into a routine that accesses the database or that performs an action outside of the database.

- You require the encapsulated logic to be invoked from any of: multiple applications, the CLP, another routine (procedure, function (UDF), or method), or a trigger.
- You are most comfortable coding this logic in a .NET language.

Prerequisites:

- Knowledge of CLR routine implementation. To learn about CLR routines in general and about CLR features, see:
  - ".NET common language runtime (CLR) routines" on page 57
- The database server must be running a Windows operating system that supports the Microsoft .NET Framework.
- The .NET Framework, version 1.1 or 2.0, must be installed on the server. The .NET Framework is independently available or as part of the Microsoft .NET Framework 1.1 Software Development Kit or .NET Framewor 2.0 Software Development Kit.
- The following versions of DB2 must be installed:
  - Server: DB2 8.2 or a later release.
  - Client: Any client that can attach to a DB2 8.2 instance will be able to invoke a CLR routine. It is recommended that you install DB2 Version 7.2 or a later release on the client.
- Authority to execute the CREATE statement for the external routine. For the privileges required to execute the CREATE PROCEDURE statement or CREATE FUNCTION statement, see the details of the appropriate statement.

Restrictions:

For a list of restrictions associated with CLR routines see:

- "Restrictions on .NET CLR routines" on page 67

Procedure:

1. Code the routine logic in any CLR supported language.
   - For general information about .NET CLR routines and .NET CLR routine features see the topics referenced in the Prerequisites section
   - Use or import the `IBM.Data.DB2` assembly if your routine will execute SQL.
   - Declare host variables and parameters correctly using data types that map to DB2 SQL data types. For a data type mapping between DB2 and .NET datatypes:
     - "Supported SQL data types for the DB2 .NET Data Provider" on page 60
   - Parameters and parameter null indicators must be declared using one of DB2's supported parameter styles and according to the parameter requirements for .NET CLR routines. As well, scratchpads for UDFs, and the DBINFO class are passed into CLR routines as parameters. For more on parameters and prototype declarations see:
     - "Parameters in .NET CLR routines" on page 62
   - If the routine is a procedure and you want to return a result set to the caller of the routine, you do not require any parameters for the result set. For more on returning result sets from CLR routines:
     - "Returning result sets from .NET CLR procedures" on page 65
   - Set a routine return value if required. CLR scalar functions require that a return value is set before returning. CLR table functions require that a return

code is specified as an output parameter for each invocation of the table function. CLR procedures do not return with a return value.

2. Build your code into an intermediate language (IL) assembly to be executed by the CLR. For information on how to build CLR .NET routines that access DB2, see the related link:
   - Building common language runtime routines
3. Copy the assembly into the DB2 *function directory* on the database server. It is recommended that you store assemblies or libraries associated with DB2 routines in the function directory. To find out more about the function directory, see the EXTERNAL clause of either of the following statements: CREATE PROCEDURE or CREATE FUNCTION.

   You can copy the assembly to another directory on the server if you want, but to successfully invoke the routine you must note the fully qualified path name of your assembly as you will require it for the next step.
4. Execute either dynamically or statically the appropriate SQL language CREATE statement for the routine type: CREATE PROCEDURE or CREATE FUNCTION.
   - Specify the `LANGUAGE` clause with value: `CLR`.
   - Specify the `PARAMETER STYLE` clause with the name of the supported parameter style that was implemented in the routine code.
   - Specify the `EXTERNAL` clause with the name of the assembly to be associated with the routine using one of the following values:
     – the fully qualified path name of the routine assembly.
     – the relative path name of the routine assembly relative to the function directory.

     By default DB2 will look for the assembly by name in the function directory unless a fully qualified or relative path name for the library is specified in the EXTERNAL clause.

     When the CREATE statement is executed, if the assembly specified in the EXTERNAL clause is not found by DB2 you will receive an error (SQLCODE -20282) with reason code 1.
   - Specify the DYNAMIC RESULT SETS clause with an integer value equivalent to the maximum number of result sets that might be returned by the routine.
   - You can not specify the NOT FENCED clause for CLR procedures. By default CLR procedures are executed as FENCED procedures.

To invoke your CLR routine, see the topic: ″Routine invocation″.

Related concepts:
- “.NET common language runtime (CLR) routines” on page 57
- “Parameters in .NET CLR routines” on page 62
- “External routine parameter styles” on page 42
- “Scratchpads for external functions and methods” on page 30
- “SQL in external routines” in *Developing SQL and External Routines*

Related tasks:
- “Returning result sets from .NET CLR procedures” on page 65
- “Building Common Language Runtime (CLR) .NET routines” on page 72
- “Creating .NET CLR routines” on page 68
- “Debugging routines” in *Developing SQL and External Routines*

Related reference:
- "Restrictions on .NET CLR routines" on page 67
- "Supported SQL data types for the DB2 .NET Data Provider" on page 60

Related samples:
- "SpCreate.db2 -- Creates the external procedures implemented in spserver.cs"
- "SpServer.cs -- C# external code implementation of procedures created in spcat.db2"
- "SpCreate.db2 -- Creates the external procedures implemented in spserver.vb"
- "SpServer.vb -- VB.NET implementation of procedures created in SpCat.db2"

Building .NET CLR routine code

Building .NET CLR routine code

Once .NET CLR routine implementation code has been written, it must be built before the routine assembly can be deployed and the routine invoked. The steps required to build .NET CLR routines are similar to those required to build any external routine however there are some differences.

There are three ways to build .NET CLR routines:
- Using the graphical tools provided with the DB2 Database Development Add-In for Visual Studio .NET 1.2
- Using DB2 sample batch files
- Entering commands from a DB2 Command Window

The DB2 sample build scripts and batch files for routines are designed for building DB2 sample routines (procedures and user-defined functions) as well as user created routines for a particular operating system using the default supported compilers.

There is a separate set of DB2 sample build scripts and batch files for C# and Visual Basic. In general it is easiest to build .NET CLR routines using the graphical tools or the build scripts which can easily be modified if required, however it is often helpful to know how to build routines from DB2 Command Windows as well.

Related concepts:
- ".NET common language runtime (CLR) routines" on page 57
- "Building .NET common language runtime (CLR) routine code using sample build scripts" on page 75

Related tasks:
- "Building .NET common language runtime (CLR) routine code from DB2 Command Windows" on page 77

Building Common Language Runtime (CLR) .NET routines

DB2 client and server products provide batch files for compiling and linking DB2 .NET programs. These are located in the `sqllib\samples\.NET\cs` and `sqllib\samples\.NET\vb` directories, along with sample programs that can be built with these files.

The batch file `bldrtn.bat` contains the commands to build CLR routines (stored procedures and user-defined functions). The batch file builds a .NET assembly DLL on the server. It takes two parameters, represented inside the batch file by the variables `%1` and `%2`.

The first parameter, `%1`, specifies the name of your source file. The batch file uses the source file name for the assembly DLL name. The second parameter, `%2`, specifies the name of the database to which you want to connect. Since the assembly DLL must be built on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, the source file name, is required. Database name is optional. If no database name is supplied, the program uses the default `sample` database.

Prerequisites:

The database server must be running a Windows operating system installed with Microsoft .NET Framework Version 1.1 or .NET Framework 2.0. The .NET Framework is independently available or as part of the Microsoft .NET Framework Software Development Kit.

The following versions of DB2 must be installed:

Server:
DB2 8.2 or later

Client:
DB2 7.2 or later

Authority must be granted to execute the CREATE statement for the routine. For the privileges required to execute the CREATE statement, see the CREATE statement for the routine type: CREATE PROCEDURE or CREATE FUNCTION.

Procedure:

The following examples show you how to build routine assembly DLLs with stored procedures and user-defined functions.

Stored procedure assembly DLL

To build the `SpServer` assembly DLL from the VB .NET source file, `SpServer.vb`, or the C# source file, `SpServer.cs`:

1. Enter the batch file name and program name (without the extension):
   ```
   bldrtn SpServer
   ```
   If connecting to a different database than the default `sample` database, also enter the database name:
   ```
   bldrtn SpServer database
   ```
   The batch file copies the assembly DLL, `SpServer.dll`, to the `sqllib\function` directory.
2. Next, catalog the routines by running the `spcat` script on the server:
   ```
   SpCat
   ```

This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling `SpDrop.db2`, then catalogs them by calling `SpCreate.db2`, and finally disconnects from the database. You can also call the `SpDrop.db2` and `SpCreate.db2` scripts individually.

3. Then, unless this is the first time the assembly DLL was built, stop and restart the database to allow the new version of the assembly DLL to be recognized. If necessary, set the file mode for the assembly DLL so the DB2 instance can access it.

Once you build the assembly DLL, `SpServer`, you can build the client application `SpClient` that calls it.

You can build `SpClient` by using the batch file, `bldapp.bat`.

To ensure you have the parameters you need when you run the executable, you can specify different combinations of parameters, instead of accepting the defaults, depending on the number of parameters entered:

1. No parameters. Enter just the program name (for calling locally on the server instance):
   ```
   SpClient
   ```
2. One parameter. Enter the program name plus the database alias (for calling a different database than the `sample` database locally on the server instance):
   ```
   SpClient <db_alias>
   ```
3. Three parameters. Enter the program name plus the database alias, user ID, and password (for calling from a remote client):
   ```
   SpClient <db_alias> <userid> <passwd>
   ```
4. Five parameters. Enter the program name plus database alias, server name, port number, user ID, and password (for calling from a remote client):
   ```
   SpClient <db_alias> <server> <portnum> <userid> <passwd>
   ```

The client application accesses the assembly DLL, `SpServer`, and executes a number of routines on the server database. The output is returned to the client application.

User-defined function assembly DLL

To build the user-defined function assembly DLL `UDFsrv` from the VB .NET source file `UDFsrv.vb`, or the C# source file, `UDFsrv.cs`, enter:

```
bldrtn UDFsrv
```

If connecting to a different database than the default `sample` database, also enter the database name:

```
bldrtn UDFsrv database
```

The batch file copies the user-defined function assembly DLL, `UDFsrv.dll`, to the `sqllib\function` directory.

Once you build `UDFsrv`, you can build the client application, `udfcli`, that calls it.

You can build `UDFcli` by using the batch file, `bldapp.bat`.

To ensure you have the parameters you need when you run the executable, you can specify different combinations of parameters, instead of accepting the defaults, depending on the number of parameters entered:

1. No parameters. Enter just the program name (for calling locally on the server instance):

   ```
   UDFcli
   ```
2. One parameter. Enter the program name plus the database alias (for calling a different database than the `sample` database locally on the server instance):

   ```
   UDFcli <db_alias>
   ```
3. Three parameters. Enter the program name plus the database alias, user ID, and password (for calling from a remote client):

   ```
   UDFcli <db_alias> <userid> <passwd>
   ```
4. Five parameters. Enter the program name plus database alias, server name, port number, user ID, and password (for calling from a remote client):

   ```
   UDFcli <db_alias> <server> <portnum> <userid> <passwd>
   ```

The calling application calls the `ScalarUDF` function from the `udfsrv` assembly DLL.

Related concepts:
- ".NET common language runtime (CLR) routines" on page 57

Related samples:
- "bldrtn.bat -- Builds C# routines (stored procedures and UDFs)"
- "SpServer.cs -- C# external code implementation of procedures created in spcat.db2"
- "SpClient.cs -- Call different types of stored procedures from SpServer.java"
- "UDFcli.cs -- Client application that calls the user-defined functions "
- "UDFsrv.cs -- User-defined scalar functions called by udfcli.cs"
- "bldrtn.bat -- Builds Visual Basic .NET routines (stored procedures and UDFs)"
- "SpServer.vb -- VB.NET implementation of procedures created in SpCat.db2"
- "SpClient.vb -- Call different types of stored procedures from SpServer.java"
- "UDFcli.vb -- Client application that calls the user-defined functions "
- "UDFsrv.vb -- User-defined scalar functions called by udfcli.vb "

Building .NET common language runtime (CLR) routine code using sample build scripts

Building .NET common langauge runtime (CLR) routine source code is a sub-task of creating .NET CLR routines. This task can be done quickly and easily using DB2 sample batch files. The sample build scripts can be used for source code with or without SQL statements. The build scripts take care of the compilation, linking, and deploymen tof the built assembly to the `function directory`.

As alternatives, you can simplify the task of building .NET CLR routine code by doing so in Visual Studio .NET or you do the steps in the DB2 sample build scripts manually. Refer to:
- Building .NET common language runtime (CLR) routines in Visual Studio .NET
- Building .NET common language runtime (CLR) routines using DB2 Command Windows

The programming language specific sample build scripts for building C# and Visual Basic .NET CLR routines are named `bldrtn`. They are located in DB2 directories along with sample programs that can be built with them as follows:

- For C: `sqllib/samples/cs/`
- For C++: `sqllib/samples/vb/`

The `bldrtn` scripts can be used to build source code files containing both procedures and user-defined functions. The script does the following:

- Establishes a connection with a user-specified database
- Compiles and links the source code to generate an assembly with a .DLL file suffix
- Copies the assembly to the DB2 function directory on the database server

The `bldrtn` scripts accept two arguments:

- The name of a source code file without any file suffix
- The name of a database to which a connection will be established

The database parameter is optional. If no database name is supplied, the program uses the default sample database. Since routines must be built on the same instance where the database resides, no arguments are required for a user ID and password.

Prerequisites:

- The requied .NET CLR routine operating system and development software prerequisements must be satisfied. See: ″Support for .NET CLR routine development″.
- Source code file containing one or more routine implementations.
- The name of the database within the current DB2 instance in which the routines are to be created.

Procedure:

To build a source code file that contains one or more routine code implementations, follow the steps below.

1. Open a DB2 Command Window.
2. Copy your source code file into the same directory as the`bldrtn`script file.
3. If the routines will be created in the sample database, enter the build script name followed by the name of the source code file without the .cs or .vb file extension:
   ```
   bldrtn <file-name>
   ```
   If the routines will be created in another database, enter the build script name, the source code file name without any file extension, and the database name:
   ```
   bldrtn <file-name> <database-name>
   ```
   The script compiles and links the source code and produces an assemblbly. The script then copies the assembly to the function directory on the database server
4. If this is not the first time that the source code file containing the routine implementations was built, stop and restart the database to ensure the new version of the shared library is used by DB2. You can do this by entering `db2stop` followed by `db2start` on the command line.

Once you have successfully built the routine shared library and deployed it to the function directory on the database server, you should complete the steps associated with the task of creating C and C++ routines.

Creating .NET CLR routines includes a step for executing the CREATE statement for each routine that was implemented in the source code file. After routine creation is completed you can invoke your routines.

Building .NET common language runtime (CLR) routine code from DB2 Command Windows

Building .NET CLR routine source code is a sub-task of creating .NET CLR routines. This task can be done manually from DB2 Command Windows. The same procedure can be followed regardless of whether there are SQL statements within the routine code or not. The task steps include compilation of source code written in a .NET CLR supported programming language into an assembly with a .DLL file suffix.

As alternatives, you can simplify the task of building .NET CLR routine code by doing so in Visual Studio .NET or by using DB2 sample build scripts. Refer to:

- Building .NET common language runtime (CLR) routines in Visual Studio .NET
- Building .NET common language runtime (CLR) routines using sample build scripts

Prerequisites:

- Required operating system and .NET CLR routine development software prerequisites have been satisfied. See: ″Support for .NET CLR routine development″.
- Source code written in a supported .NET CLR programming language containing one or more .NET CLR routine implementations.
- The name of the database within the current DB2 instance in which the routines are to be created.
- The operating specific compile and link options required for building .NET CLR routines.

Procedure:

To build a source code file that contains one or more .NET CLR routine code implementations, follow the steps below. An example follows that demonstrates each of the steps:

1. Open a DB2 Command Window.
2. Navigate to the directory that contains your source code file.
3. Establish a connection with the database in which the routines will be created.
4. Compile the source code file.
5. Link the source code file to generate a shared library. This requires the use of some DB2 specific compile and link options.
6. Copy the assembly file with the .DLL file suffix to the DB2 function directory on the database server.
7. If this is not the first time that the source code file containing the routine implementations was built, stop and restart the database to ensure the new version of the shared library is used by DB2. You can do this by issuing the db2stop command followed by the db2start command.

Once you have successfully built and deployed the routine library, you should complete the steps associated with the task of creating .NET CLR routines. Creating .NET CLR routines includes a step for executing the CREATE statement for each routine that was implemented in the source code file. This step must also be completed before you will be able to invoke the routines.

Example:

The following example demonstrates the re-building of a .NET CLR source code file. Steps are shown for both a Visual Basic code file named `myVBfile.vb` containing routine implementations as well as for a C# code file named `myCSfile.cs`. The routines are being built on a Windows 2000 operating system using Microsoft .NET Framework 1.1 to generate a 64-bit assembly.

1. Open a DB2 Command Window.
2. Navigate to the directory that contains your source code file.
3. Establish a connection with the database in which the routines will be created.
   ```
   db2 connect to <database-name>
   ```
4. Compile the source code file using the recommended compile and link options (where $DB2PATH is the install path of the DB2 instance. Replace this value before running the command):
   ```
   C# example
   ===================
   csc /out:myCSfile.dll /target:library
       /reference:$DB2PATH%\bin\netf11\IBM.Data.DB2.dll myCSfile.cs

   Visual Basic example
   ====================
   vbc /target:library /libpath:$DB2PATH\bin\netf11
       /reference:$DB2PATH\bin\netf11\IBM.Data.DB2.dll
       /reference:System.dll
       /reference:System.Data.dll myVBfile.vb
   ```
   The compiler will generate output if there are any errors. This step generates an export file named myfile.exp.
5. Copy the shared library to the DB2 function directory on the database server.
   ```
   C# example
   ====================
   rm -f ~HOME/sqllib/function/myCSfile.DLL
   cp myCSfile $HOME/sqllib/function/myCSfile.DLL

   Visual Basic example
   ====================
   rm -f ~HOME/sqllib/function/myVBfile.DLL
   cp myVBfile $HOME/sqllib/function/myVBfile.DLL
   ```
   This step ensures that the routine library is in the default directory where DB2 looks for routine libraries. Refer to the topic on creating .NET CLR routines for more on deploying routine libraries.
6. Stop and restart the database as this is a re-building of a previously built routine source code file.
   ```
   db2stop
   db2start
   ```

Building .NET CLR routines is generally most easily done using the operating specific sample build scripts which also can be used as a reference for how to build routines from the command line.

CLR .NET routine compile and link options

The following are the compile and link options recommended by DB2 for building Common Language Runtime (CLR) .NET routines on Windows with either the Microsoft Visual Basic .NET compiler or the Microsoft C# compiler, as demonstrated in the `samples\.NET\cs\bldrtn.bat` and `samples\.NET\vb\bldrtn.bat` batch files.

Compile and link options for bldrtn	
Compile and link options using the Microsoft C# compiler:	
`csc`	The Microsoft C# compiler.
`/out:%1.dll /target:library`	Output the data link library as a stored procedure assembly dll.
`/debug`	Use the debugger.
`/lib: "%DB2PATH%"\bin\netf11\`	Use the library path for .NET Framework Version 1.1.
`/reference:IBM.Data.DB2.dll`	Use the DB2 data link library for .NET Framework Version 1.1.
Refer to your compiler documentation for additional compiler options.	
Compile and link options using the Microsoft Visual Basic .NET compiler:	
`vbc`	The Microsoft Visual Basic .NET compiler.
`/out:%1.dll /target:library`	Output the data link library as a stored procedure assembly dll.
`/debug`	Use the debugger.
`/libpath:"%DB2PATH%"\bin\netf11\`	Use the library path for .NET Framework Version 1.1.
`/reference:IBM.Data.DB2.dll`	Use the DB2 data link library for .NET Framework Version 1.1.
`/reference:System.dll`	Reference the Microsoft Windows System data link library.
`/reference:System.Data.dll`	Reference the Microsoft Windows System Data data link library.
Refer to your compiler documentation for additional compiler options.	

Related tasks:
- "Building Common Language Runtime (CLR) .NET routines" on page 72

Related samples:
- "bldrtn.bat -- Builds C# routines (stored procedures and UDFs)"
- "bldrtn.bat -- Builds Visual Basic .NET routines (stored procedures and UDFs)"

Debugging .NET CLR routines

Debugging .NET CLR routines

Debugging .NET CLR routines might be required if you fail to be able to create a routine, invoke a routine, or if upon invocation a routine does not behave or perform as expected.

Procedure:

Consider the following when debugging .NET CLR routines:

- Verify that a supported operating system for .NET CLR routine development is being used.
- Verify that both a supported DB2 database server and DB2 client for .NET CLR routine development are being used.
- Verify that supported Microsoft .NET Framework development software is being used.
- If routine creation failed:
  - Verify that the user has the required authority and privileges to execute the CREATE PROCEDURE or CREATE FUNCTION statement.
- If routine invocation failed:
  - Verify that the user has authority to execute the routine. If an error (SQLCODE -551, SQLSTATE 42501), this is likely because the invoker does not have the EXECUTE privilege on the routine. This privilege can be granted by a user with SYSADM authorization, DBADM authorization, or by the definer of the routine.
  - Verify that the routine parameter signature used in the CREATE statement for the routine matches the routine parameter signature in the routine implementation.
  - Verify that the data types used in the routine implementation are compatible with the data types specified in the routine parameter signature in the CREATE statement.
  - Verify that in the routine implementation that the .NET CLR language specific keywords used to indicate the method by which the parameter must be passed (by value or by reference) are valid.
  - Verify that the value specified in the EXTERNAL clause in the CREATE PROCEDURE or CREATE FUNCTION statement matches the location where the .NET CLR assembly that contains the routine implementation is located on the file system of the computer where the DB2 database server is installed.
  - If the routine is a function, verify that all of the applicable call types have been programmed correctly in the routine implementation. This is particularly important if the routine was defined with the FINAL CALL clause.
- If the routine is not behaving as expected:
  - Modify your routine such that it outputs diagnostic information to a file located in a globally accessible directory. Output of diagnostic information to the screen is not possible from .NET CLR routines. Do not direct output to files in directories used by DB2 database managers or DB2 databases.
  - Debug your routine locally by writing a simple .NET application that invokes the routine entry point directly. For information on how to use debugging features in Microsoft Visual Studio .NET, consult the Microsoft Visual Studio .NET compiler documentation.

For more information on common errors related to .NET CLR routine creation and invocation, see:

-

Related concepts:

- ".NET common language runtime (CLR) routines" on page 57
- "Restrictions on external routines" on page 52

Related tasks:

- "Debugging .NET CLR routines from the command line" in *Developing SQL and External Routines*

Related reference:

- "Errors related to .NET CLR routines" on page 81
- "Restrictions on .NET CLR routines" on page 67

Errors related to .NET CLR routines

Although external routines share a generally common implementation, there are some DB2 errors that might arise that are specific to CLR routines. This reference lists the most commonly encountered .NET CLR related errors listed by their SQLCODE or behavior along with some debugging suggestions. DB2 errors related to routines can be classified as follows:

Routine creation time errors
Errors that arise when the CREATE statement for the routine is executed.

Routine runtime errors
Errors that arise during the routine invocation or execution.

Regardless of when a DB2 routine related error is raised by DB2, the error message text details the cause of the error and the action that the user should take to resolve the problem. Additional routine error scenario information can be found in the `db2diag.log` diagnostic log file.

CLR routine creation time errors:

SQLCODE -451, SQLSTATE 42815
This error is raised upon an attempt to execute a CREATE TYPE statement that includes an external method declaration specifying the LANGUAGE clause with value CLR. You can not create DB2 external methods for structured types that reference a CLR assembly at this time. Change the LANGUAGE clause so that it specifies a supported language for the method and implement the method in that alternate language.

SQLCODE -449, SQLSTATE 42878
The CREATE statement for the CLR routine contains an invalidly formatted library or function identification in the EXTERNAL NAME clause. For language CLR, the EXTERNAL clause value must specifically take the form: `'<a>:<b>!<c>'` as follows:

- `<a>` is the CLR assembly file in which the class is located.
- `<b>` is the class in which the method to invoke resides.
- `<c>` is the method to invoke.

No leading or trailing blank characters are permitted between the single quotation marks, object identifiers, and the separating characters (for example, `' <a> ! <b> '` is invalid). Path and file names, however, can contain blanks if the platform permits. For all file names, the file can be specified using either the short form of the name (example: `math.dll` or the fully qualified path name (example: `d:\udfs\math.dll`. If the short form of the file name is used, if the platform is UNIX or if the routine is a LANGUAGE CLR routine, then the file must reside in the function directory. If the platform is Windows and the routine is not a LANGUAGE

CLR routine then the file must reside in the system PATH. File extensions (examples: `.a` (on UNIX), `.dll` (on Windows)) should always be included in the file name.

CLR routine runtime errors:

SQLCODE -20282, SQLSTATE 42724, reason code 1
The external assembly specified by the EXTERNAL clause in the CREATE statement for the routine was not found.

- Check that the EXTERNAL clause specifies the correct routine assembly name and that the assembly is located in the specified location. If the EXTERNAL clause does not specify a fully qualified path name to the desired assembly, DB2 presumes that the path name provided is a relative path name to the assembly, relative to the DB2 function directory.

SQLCODE -20282, SQLSTATE 42724, reason code 2
An assembly was found in the location specified by the EXTERNAL clause in the CREATE statement for the routine, but no class was found within the assembly to match the class specified in the EXTERNAL clause.

- Check that the assembly name specified in the EXTERNAL clause is the correct assembly for the routine and that it exists in the specified location.
- Check that the class name specified in the EXTERNAL clause is the correct class name and that it exists in the specified assembly.

SQLCODE -20282, SQLSTATE 42724, reason code 3
An assembly was found in the location specified by the EXTERNAL clause in the CREATE statement for the routine, that had a correctly matching class definition, but the routine method signature does not match the routine signature specified in the CREATE statement for the routine.

- Check that the assembly name specified in the EXTERNAL clause is the correct assembly for the routine and that it exists in the specified location.
- Check that the class name specified in the EXTERNAL clause is the correct class name and that it exists in the specified assembly.
- Check that the parameter style implementation matches the parameter style specified in the CREATE statement for the routine.
- Check that the order of the parameter implementation matches the parameter declaration order in the CREATE statement for the routine and that it respects the extra parameter requirements for the parameter style.
- Check that the SQL parameter data types are correctly mapped to CLR .NET supported data types.

SQLCODE -4301, SQLSTATE 58004, reason code 5 or 6
An error occurred while attempting to start or communicate with a .NET interpreter. DB2 was unable to load a dependent .NET library [reason code 5] or a call to the .NET interpreter failed [reason code 6].

- Ensure that the DB2 instance is configured correctly to run a .NET procedure or function (`mscoree.dll` must be present in the system `PATH`). Ensure that `db2clr.dll` is present in the `sqllib/bin` directory, and that `IBM.Data.DB2` is installed in the global assembly cache. If these are not present, ensure that the `.NET Framework` version 1.1, or a later version, is installed on the database server, and that the database server is running DB2 version 8.2 or a later release.

SQLCODE -4302, SQLSTATE 38501

An unhandled exception occurred while executing, preparing to execute, or subsequent to executing the routine. This could be the result of a routine logic programming error that was unhandled or could be the result of an internal processing error. For errors of this type, the .NET stack traceback that indicates where the unhandled exception occured will be written to the db2diag.log.

This error can also occur if the routine attempted an action that is beyond the scope of allowed actions for the specified execution mode for the routine. In this case, an entry will be made in the db2diag.log specifically indicating that the exception occured due to an execution control violation. The exception stack traceback that indicates where the violation occured will also be included.

Determine if the assembly of the routine has been compromised or recently modifed. If the routine has been validly modified, this problem can be occuring because the EXECUTION CONTROL mode for the routine is no longer set to a mode that is appropriate for the changed logic. If you are certain that the assembly has not been wrongfully tampered with, you can modify the routine's execution mode with the ALTER PROCEDURE or ALTER FUNCTION statement as appropriate. Refer to the following topic for more information:

- "Security and execution modes for CLR routines" on page 66

Related concepts:
- ".NET common language runtime (CLR) routines" on page 57
- "Authorizations and binding of routines that contain SQL" in *Developing SQL and External Routines*
- "External routine library and class management" on page 45
- "SQL in external routines" in *Developing SQL and External Routines*

Related tasks:
- "Creating .NET CLR routines from DB2 Command Windows" on page 69
- "Debugging .NET CLR routines" on page 79

Migrating .NET CLR routines to DB2 9.1

Migrating .NET CLR routines

After migrating a DB2 instance and databases to DB2 Version 9, you must migrate your .NET CLR routines that you created prior to Version 9 to ensure that they continue to function successfully and perform as expected.

Prerequisites:
- Review the migration essentials for routines to identify key changes that might apply to your .NET CLR routines.
- Ensure that you have access to a DB2 Version 9 server, including instances and databases. The DB2 server can be part of a testing environment.
- Ensure that the operating system is at a version level that is supported by DB2 database products.
- Ensure that a supported version of the .NET Framework software is installed on the DB2 server.

- Perform the pre-migration tasks for routines.

Procedure:

To migrate your .NET CLR routines to DB2 Version 9:

1. If you identified changes in DB2 Version 9 that impact your routines, edit your routine code and modify:
   - SQL statement syntax
   - SQL statements using SQL Administrative views and routines, and catalog views
2. Connect to the DB2 Version 9 database in which you defined the .NET CLR.
3. Rebuild your .NET CLR routine source code using the compile and link options specified in `bldrtn.bat`, the DB2 sample script for building .NET CLR routines.
4. Deploy the routine assembly to the DB2 server in the same location specified by the EXTERNAL clause in the routine definition.
5. Test your .NET CLR routines. The routines should function successfully, with no differences in between DB2 UDB Version 8 and DB2 Version 9 behavior.

After migrating your .NET CLR routines, perform the recommended post-migration tasks for routines.

Related concepts:
- "Migration essentials for routines" in *Migration Guide*
- "Supported .NET development software" on page 2
- "Pre-migration tasks for database applications and routines" in *Migration Guide*
- "DB2 .NET Data Provider database system requirements" on page 7
- "Designing .NET CLR routines" on page 59
- "ODBC .NET Data Provider" on page 167
- "DB2 .NET Data Provider" on page 7

Related tasks:
- "Building .NET CLR routine code" on page 72

Related reference:
- "CLR .NET routine compile and link options" on page 79

Examples of .NET CLR routines

Examples of .NET CLR routines

When developing .NET CLR routines, it is helpful to refer to examples to get a sense of what the CREATE statement and the .NET CLR routine code should look like. The following topics contain examples of .NET CLR procedures and functions (including both scalar and table functions):

.NET CLR procedures
- Examples of Visual Basic .NET CLR procedures
- Examples of C# .NET CLR procedures

.NET CLR functions
- Examples of Visual Basic .NET CLR functions

- Examples of C# .NET CLR functions

Related concepts:
- ".NET common language runtime (CLR) routines" on page 57

Related tasks:
- "Building .NET CLR routine code" on page 72
- "Creating .NET CLR routines" on page 68
- "Examples of C# .NET CLR functions" on page 116
- "Examples of C# .NET CLR procedures" on page 94
- "Examples of Visual Basic .NET CLR functions" on page 110
- "Examples of Visual Basic .NET CLR procedures" on page 85

Related reference:
- "Supported SQL statements" in *SQL Reference, Volume 2*

Examples of Visual Basic .NET CLR procedures

Once the basics of procedures, also called stored procedures, and the essentials of .NET common language runtime routines are understood, you can start using CLR procedures in your applications.

This topic contains examples of CLR procedures implemented in `Visual Basic`; that illustrate the supported parameter styles, passing parameters, including the `dbinfo` structure, how to return a result set and more. For examples of CLR UDFs in `Visual Basic`:
- "Examples of Visual Basic .NET CLR functions" on page 110

Prerequisites:

Before working with the CLR procedure examples you might want to read the following concept topics:
- ".NET common language runtime (CLR) routines" on page 57
- "Creating .NET CLR routines from DB2 Command Windows" on page 69
- "Benefits of using routines" on page 20
- Building common language runtime (CLR) .NET routines

The examples below make use of a table named `EMPLOYEE` that is contained in the `SAMPLE` database.

Procedure:

Use the following examples as references when making your own Visual Basic CLR procedures:
- "The Visual Basic external code file" on page 86
- "Example 1: Visual Basic parameter style GENERAL procedure" on page 86
- "Example 2: Visual Basic parameter style GENERAL WITH NULLS procedure" on page 87
- "Example 3: Visual Basic parameter style SQL procedure" on page 88
- "Example 4: Visual Basic procedure returning a result set" on page 90
- "Example 5: Visual Basic procedure accessing the dbinfo structure" on page 91

- "Example 6: Visual Basic procedure in PROGRAM TYPE MAIN style " on page 92

The Visual Basic external code file:

The examples show a variety of Visual Basic procedure implementations. Each example consists of two parts: the CREATE PROCEDURE statement and the external Visual Basic code implementation of the procedure from which the associated assembly can be built.

The Visual Basic source file that contains the procedure implementations of the following examples is named `gwenVbProc.vb` and has the following format:

Table 5. Visual Basic external code file format

```
using System;
using System.IO;
using IBM.Data.DB2;

Namespace bizLogic

   Class empOps
              ...
     ' Visual Basic procedures
              ...
   End Class
End Namespace
```

The file inclusions are indicated at the top of the file. The `IBM.Data.DB2` inclusion is required if any of the procedures in the file contain SQL. There is a namespace declaration in this file and a class `empOps` that contains the procedures. The use of namespaces is optional. If a namespace is used, the namespace must appear in the assembly path name provided in the EXTERNAL clause of the CREATE PROCEDURE statement.

It is important to note the name of the file, the namespace, and the name of the class, that contains a given procedure implementation. These names are important, because the EXTERNAL clause of the CREATE PROCEDURE statement for each procedure must specify this information so that DB2 can locate the assembly and class of the CLR procedure.

Example 1: Visual Basic parameter style GENERAL procedure:

This example shows the following:
- CREATE PROCEDURE statement for a parameter style GENERAL procedure
- Visual Basic code for a parameter style GENERAL procedure

This procedure takes an employee ID and a current bonus amount as input. It retrieves the employee's name and salary. If the current bonus amount is zero, a new bonus is calculated, based on the employee salary, and returned along with the employee's full name. If the employee is not found, an empty string is returned.

Table 6. Code to create a Visual Basic parameter style GENERAL procedure

```
CREATE PROCEDURE SetEmpBonusGEN(IN empId CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC setEmpBonusGEN
LANGUAGE CLR
PARAMETER STYLE GENERAL
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusGEN'
```

```
  Public Shared Sub SetEmpBonusGEN(ByVal empId As String, _
                                   ByRef bonus As Decimal, _
                                   ByRef empName As String)

     Dim salary As Decimal
     Dim myCommand As DB2Command
     Dim myReader As DB2DataReader

     salary = 0

     myCommand = DB2Context.GetCommand()
     myCommand.CommandText = _
               "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
             + "FROM EMPLOYEE " _
             + "WHERE EMPNO = '" + empId + "'"
     myReader = myCommand.ExecuteReader()

     If myReader.Read()  ' If employee record is found
        ' Get the employee's full name and salary
        empName = myReader.GetString(0) + " " _
                + myReader.GetString(1) + ". " _
                + myReader.GetString(2)

        salary = myReader.GetDecimal(3)

        If bonus = 0
           If salary > 75000
              bonus = salary * 0.025
           Else
              bonus = salary * 0.05
           End If
        End If
     Else  ' Employee not found
        empName = ""  ' Set output parameter
     End If

     myReader.Close()

  End Sub
```

Example 2: Visual Basic parameter style GENERAL WITH NULLS procedure: This example shows the following:

- CREATE PROCEDURE statement for a parameter style GENERAL WITH NULLS procedure
- Visual Basic code for a parameter style GENERAL WITH NULLS procedure

This procedure takes an employee ID and a current bonus amount as input. If the input parameter is not null, it retrieves the employee's name and salary. If the current bonus amount is zero, a new bonus based on salary is calculated and returned along with the employee's full name. If the employee data is not found, a NULL string and integer is returned.

Table 7. Code to create a Visual Basic parameter style GENERAL WITH NULLS procedure

```
CREATE PROCEDURE SetEmpBonusGENNULL(IN empId CHAR(6),
                                    INOUT bonus Decimal(9,2),
                                    OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusGENNULL
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusGENNULL'
```

```
  Public Shared Sub SetEmpBonusGENNULL(ByVal empId As String, _
                                       ByRef bonus As Decimal, _
                                       ByRef empName As String, _
                                       byVal nullInds As Int16())

     Dim salary As Decimal
     Dim myCommand As DB2Command
     Dim myReader As DB2DataReader

     salary = 0

     If nullInds(0) = -1   ' Check if the input is null
        nullInds(1) = -1   ' Return a NULL bonus value
        empName = ""       ' Set output parameter
        nullInds(2) = -1   ' Return a NULL empName value
        Return
     Else
        myCommand = DB2Context.GetCommand()
        myCommand.CommandText = _
                   "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
                 + "FROM EMPLOYEE " _
                 + "WHERE EMPNO = '" + empId + "'"

        myReader = myCommand.ExecuteReader()

        If myReader.Read()  ' If employee record is found
           ' Get the employee's full name and salary
           empName = myReader.GetString(0) + " " _
                   + myReader.GetString(1) + ". " _
                   + myReader.GetString(2)

           salary = myReader.GetDecimal(3)

           If bonus = 0
              If salary > 75000
                 bonus = Salary * 0.025
                 nullInds(1) = 0 'Return a non-NULL value
              Else
                 bonus = salary * 0.05
                 nullInds(1) = 0 ' Return a non-NULL value
              End If
           Else  'Employee not found
              empName = ""       ' Set output parameter
              nullInds(2) = -1   ' Return a NULL value
           End If
        End If

        myReader.Close()

     End If

  End Sub
```

Example 3: Visual Basic parameter style SQL procedure:

This example shows the following:

- CREATE PROCEDURE statement for a parameter style SQL procedure
- Visual Basic code for a parameter style SQL procedure

This procedure takes an employee ID and a current bonus amount as input. It retrieves the employee's name and salary. If the current bonus amount is zero, a new bonus based on salary is calculated and returned along with the employee's full name. If the employee is not found, an empty string is returned.

Table 8. Code to create a Visual Basic procedure in parameter style SQL with parameters

```
CREATE PROCEDURE SetEmpBonusSQL(IN empId CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusSQL
LANGUAGE CLR
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusSQL'
```

Table 8. Code to create a Visual Basic procedure in parameter style SQL with parameters (continued)

```
    Public Shared Sub SetEmpBonusSQL(byVal empId As String, _
                                     byRef bonus As Decimal, _
                                     byRef empName As String, _
                                     byVal empIdNullInd As Int16, _
                                     byRef bonusNullInd As Int16, _
                                     byRef empNameNullInd As Int16, _
                                     byRef sqlState As String, _
                                     byVal funcName As String, _
                                     byVal specName As String, _
                                     byRef sqlMessageText As String)

       ' Declare local host variables
       Dim salary As Decimal
       Dim myCommand As DB2Command
       Dim myReader As DB2DataReader

       salary = 0

       If empIdNullInd = -1   ' Check if the input is null
          bonusNullInd = -1   ' Return a NULL Bonus value
          empName = ""
          empNameNullInd = -1 ' Return a NULL empName value
       Else
         myCommand = DB2Context.GetCommand()
         myCommand.CommandText = _
                    "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
                 + "FROM EMPLOYEE " _
                 + " WHERE EMPNO = '" + empId + "'"

          myReader = myCommand.ExecuteReader()

          If myReader.Read()  ' If employee record is found
             '  Get the employee's full name and salary
             empName = myReader.GetString(0) + " "
                     + myReader.GetString(1) _
                     + ". " +  myReader.GetString(2)
             empNameNullInd = 0
             salary = myReader.GetDecimal(3)

             If bonus = 0
                If salary > 75000
                   bonus = salary * 0.025
                   bonusNullInd = 0  ' Return a non-NULL value
                Else
                   bonus = salary * 0.05
                   bonusNullInd = 0  ' Return a non-NULL value
                End If
             End If
          Else  ' Employee not found
             empName = ""            ' Set output parameter
             empNameNullInd = -1     ' Return a NULL value
          End If

          myReader.Close()
       End If

    End Sub
```

Example 4: Visual Basic parameter style GENERAL procedure returning a result set:
This example shows the following:

- CREATE PROCEDURE statement for an external Visual Basic procedure returning a result set

- Visual Basic code for a parameter style GENERAL procedure that returns a result set

This procedure accepts the name of a table as a parameter. It returns a result set containing all the rows of the table specified by the input parameter. This is done by leaving a `DB2DataReader` for a given query result set open when the procedure returns. Specifically, if `reader.Close()` is not executed, the result set will be returned.

Table 9. Code to create a Visual Basic procedure that returns a result set

```
  CREATE PROCEDURE ReturnResultSet(IN tableName VARCHAR(20))
  SPECIFIC ReturnResultSet
  DYNAMIC RESULT SETS 1
  LANGUAGE CLR
  PARAMETER STYLE GENERAL
  FENCED
  PROGRAM TYPE SUB
  EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!ReturnResultSet'
```

```
Public Shared Sub ReturnResultSet(byVal tableName As String)

      Dim myCommand As DB2Command
      Dim myReader As DB2DataReader

      myCommand = DB2Context.GetCommand()

      ' Set the SQL statement to be executed and execute it.
      myCommand.CommandText = "SELECT * FROM " + tableName
      myReader = myCommand.ExecuteReader()

      ' The DB2DataReader contains the result of the query.
      ' This result set can be returned with the procedure,
      ' by simply NOT closing the DB2DataReader.
      ' Specifically, do NOT execute reader.Close()

    End Sub
```

Example 5: Visual Basic parameter style SQL procedure accessing the dbinfo structure:
This example shows the following:
- CREATE PROCEDURE statement for a procedure accessing the `dbinfo` structure
- Visual Basic code for a parameter style SQL procedure that accesses the `dbinfo` structure

To access the `dbinfo` structure, the DBINFO clause must be specified in the CREATE PROCEDURE statement. No parameter is required for the `dbinfo` structure in the CREATE PROCEDURE statement however a parameter must be created for it, in the external routine code. This procedure returns only the value of the current database name from the `dbname` field in the `dbinfo` structure.

Table 10. Code to create a Visual Basic procedure that accesses the dbinfo structure

```
  CREATE PROCEDURE ReturnDbName(OUT dbName VARCHAR(20))
  SPECIFIC ReturnDbName
  LANGUAGE CLR
  PARAMETER STYLE SQL
  DBINFO
  FENCED
  PROGRAM TYPE SUB
  EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!ReturnDbName'
```

Table 10. Code to create a Visual Basic procedure that accesses the dbinfo structure (continued)

```
  Public Shared Sub ReturnDbName(byRef dbName As String, _
                                 byRef dbNameNullInd As Int16, _
                                 byRef sqlState As String, _
                                 byVal funcName As String, _
                                 byVal specName As String, _
                                 byRef sqlMessageText As String, _
                                 byVal dbinfo As sqludf_dbinfo)

    ' Retrieve the current database name from the
    ' dbinfo structure and return it.
    dbName = dbinfo.dbname
    dbNameNullInd = 0  ' Return a non-null value

    ' If you want to return a user-defined error in
    ' the SQLCA you can specify a 5 digit user-defined
    ' SQLSTATE and an error message string text.
    ' For example:
    '
    ' sqlState = "ABCDE"
    ' msg_token = "A user-defined error has occured"
    '
    ' These will be returned by DB2 in the SQLCA.  It
    ' will appear in the format of a regular DB2 sqlState
    ' error.
  End Sub
```

Example 6: Visual Basic procedure with PROGRAM TYPE MAIN style:
This example shows the following:

- CREATE PROCEDURE statement for a procedure using a main program style
- Visual Basic parameter style GENERAL WITH NULLS code in using a MAIN program style

To implement a routine in a main program style, the PROGRAM TYPE clause must be specified in the CREATE PROCEDURE statement with the value MAIN. Parameters are specified in the CREATE PROCEDURE statement however in the code implementation, parameters are passed into the routine in an `argc` integer parameter and an `argv` array of parameters.

Table 11. Code to create a Visual Basic procedure in program type MAIN style

```
CREATE PROCEDURE MainStyle(IN empId CHAR(6),
                           INOUT bonus Decimal(9,2),
                           OUT empName VARCHAR(60))
SPECIFIC mainStyle
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
FENCED
PROGRAM TYPE MAIN
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!Main'
```

Table 11. Code to create a Visual Basic procedure in program type MAIN style (continued)

```
   Public Shared Sub Main( byVal argc As Int32, _
                           byVal argv As Object())

      Dim myCommand As DB2Command
      Dim myReader As DB2DataReader
      Dim empId As String
      Dim bonus As Decimal
      Dim salary As Decimal
      Dim nullInds As Int16()

      empId = argv(0)  ' argv[0] (IN)    nullInd = argv[3]
      bonus = argv(1)  ' argv[1] (INOUT) nullInd = argv[4]
                       ' argv[2] (OUT)   nullInd = argv[5]
      salary = 0
      nullInds = argv(3)

      If nullInds(0) = -1     ' Check if the empId input is null
         nullInds(1) = -1     ' Return a NULL Bonus value
         argv(1) = ""         ' Set output parameter empName
         nullInds(2) = -1     ' Return a NULL empName value
         Return
      Else
         ' If the employee exists and the current bonus is 0,
         ' calculate a new employee bonus based on the employee's
         ' salary.  Return the employee name and the new bonus
         myCommand = DB2Context.GetCommand()
         myCommand.CommandText = _
                    "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
                  + " FROM EMPLOYEE " _
                  + " WHERE EMPNO = '" + empId + "'"

         myReader = myCommand.ExecuteReader()


         If myReader.Read()  ' If employee record is found
            ' Get the employee's full name and salary
            argv(2) = myReader.GetString(0) + " " _
                    + myReader.GetString(1) + ". " _
                    + myReader.GetString(2)
            nullInds(2) = 0
            salary = myReader.GetDecimal(3)

            If bonus = 0
               If salary > 75000
                  argv(1) = salary * 0.025
                  nullInds(1) = 0  ' Return a non-NULL value
               Else
                  argv(1) = Salary * 0.05
                  nullInds(1) = 0  ' Return a non-NULL value
               End If
            End If
         Else  ' Employee not found
            argv(2) = ""      ' Set output parameter
            nullInds(2) = -1  ' Return a NULL value
         End If

         myReader.Close()
      End If

   End Sub
```

Related concepts:

- ".NET common language runtime (CLR) routines" on page 57
- "Benefits of using routines" on page 20

Related tasks:

- “Examples of Visual Basic .NET CLR functions” on page 110
- “Creating .NET CLR routines from DB2 Command Windows” on page 69
- “Building Common Language Runtime (CLR) .NET routines” on page 72
- “Examples of C# .NET CLR procedures” on page 94
- “Examples of C# .NET CLR functions” on page 116

Examples of C# .NET CLR procedures

Once the basics of procedures, also called stored procedures, and the essentials of .NET common language runtime routines are understood, you can start using CLR procedures in your applications.

This topic contains examples of CLR procedures implemented in C# that illustrate the supported parameter styles, passing parameters, including the `dbinfo` structure, how to return a result set and more. For examples of CLR UDFs in C#:

- “Examples of C# .NET CLR functions” on page 116

Prerequisites:

Before working with the CLR procedure examples you might want to read the following concept topics:

- “.NET common language runtime (CLR) routines” on page 57
- “Creating .NET CLR routines from DB2 Command Windows” on page 69
- “Benefits of using routines” on page 20
- Building common language runtime (CLR) .NET routines

The examples below make use of a table named `EMPLOYEE` that is contained in the `SAMPLE` database.

Procedure:

Use the following examples as references when making your own C# CLR procedures:

- “The C# external code file”
- “Example 1: C# parameter style GENERAL procedure” on page 95
- “Example 2: C# parameter style GENERAL WITH NULLS procedure” on page 96
- “Example 3: C# parameter style SQL procedure” on page 98
- “Example 4: C# procedure returning a result set” on page 101
- “Example 5: C# procedure accessing the dbinfo structure” on page 101
- “Example 6: C# procedure in PROGRAM TYPE MAIN style ” on page 102

The C# external code file:

The examples show a variety of C# procedure implementations. Each example consists of two parts: the CREATE PROCEDURE statement and the external C# code implementation of the procedure from which the associated assembly can be built.

The C# source file that contains the procedure implementations of the following examples is named gwenProc.cs and has the following format:

Table 12. C# external code file format

```
using System;
using System.IO;
using IBM.Data.DB2;

namespace bizLogic
{
   class empOps
   {          ...
     // C# procedures
              ...
   }
}
```

The file inclusions are indicated at the top of the file. The IBM.Data.DB2 inclusion is required if any of the procedures in the file contain SQL. There is a namespace declaration in this file and a class empOps that contains the procedures. The use of namespaces is optional. If a namespace is used, the namespace must appear in the assembly path name provided in the EXTERNAL clause of the CREATE PROCEDURE statement.

It is important to note the name of the file, the namespace, and the name of the class, that contains a given procedure implementation. These names are important, because the EXTERNAL clause of the CREATE PROCEDURE statement for each procedure must specify this information so that DB2 can locate the assembly and class of the CLR procedure.

Example 1: C# parameter style GENERAL procedure:

This example shows the following:

- CREATE PROCEDURE statement for a parameter style GENERAL procedure
- C# code for a parameter style GENERAL procedure

This procedure takes an employee ID and a current bonus amount as input. It retrieves the employee's name and salary. If the current bonus amount is zero, a new bonus is calculated, based on the employee's salary, and returned along with the employee's full name. If the employee is not found, an empty string is returned.

Table 13. Code to create a C# parameter style GENERAL procedure

```
CREATE PROCEDURE setEmpBonusGEN(IN empID CHAR(6), INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusGEN
LANGUAGE CLR
PARAMETER STYLE GENERAL
MODIFIES SQL DATA
EXECUTION CONTROL SAFE
FENCED
THREADSAFE
DYNAMIC RESULT SETS 0
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusGEN' ;
```

Table 13. Code to create a C# parameter style GENERAL procedure (continued)

```
public static void SetEmpBonusGEN(    String empID,
                                  ref Decimal bonus,
                                  out String empName)
{
   // Declare local variables
   Decimal salary = 0;

   DB2Command myCommand = DB2Context.GetCommand();
   myCommand.CommandText =
                    "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY "
                  + "FROM EMPLOYEE "
                  + "WHERE EMPNO = '" + empID + '";

   DB2DataReader reader = myCommand.ExecuteReader();

   if (reader.Read())  // If employee record is found
   {
      // Get the employee's full name and salary
      empName = reader.GetString(0) + " " +
                reader.GetString(1) + ". " +
                reader.GetString(2);

      salary = reader.GetDecimal(3);

      if (bonus == 0)
      {
         if (salary > 75000)
         {
            bonus = salary * (Decimal)0.025;
         }
         else
         {
            bonus = salary * (Decimal)0.05;
         }
      }
   }
   else  // Employee not found
   {
      empName = "";  // Set output parameter
   }

   reader.Close();
}
```

Example 2: C# parameter style GENERAL WITH NULLS procedure:
This example shows the following:

- CREATE PROCEDURE statement for a parameter style GENERAL WITH NULLS procedure
- C# code for a parameter style GENERAL WITH NULLS procedure

This procedure takes an employee ID and a current bonus amount as input. If the input parameter is not null, it retrieves the employee's name and salary. If the current bonus amount is zero, a new bonus based on salary is calculated and returned along with the employee's full name. If the employee data is not found, a NULL string and integer is returned.

Table 14. Code to create a C# parameter style GENERAL WITH NULLS procedure

```
CREATE PROCEDURE SetEmpbonusGENNULL(IN empID CHAR(6),
                                    INOUT bonus Decimal(9,2),
                                    OUT empName VARCHAR(60))
SPECIFIC SetEmpbonusGENNULL
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
DYNAMIC RESULT SETS 0
MODIFIES SQL DATA
EXECUTION CONTROL SAFE
FENCED
THREADSAFE
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusGENNULL'
;
```

Table 14. Code to create a C# parameter style GENERAL WITH NULLS procedure (continued)

```
   public static void SetEmpBonusGENNULL(    String empID,
                                         ref Decimal bonus,
                                         out String empName,
                                             Int16[] NullInds)
   {
      Decimal salary = 0;
      if (NullInds[0] == -1) // Check if the input is null
      {
        NullInds[1] = -1;    // Return a NULL bonus value
        empName = "";        // Set output value
        NullInds[2] = -1;    // Return a NULL empName value
      }
      else
      {
         DB2Command myCommand = DB2Context.GetCommand();
        myCommand.CommandText =
                         "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY "
                       + "FROM EMPLOYEE "
                       + "WHERE EMPNO = '" + empID + "'";
         DB2DataReader reader = myCommand.ExecuteReader();

         if (reader.Read())  // If employee record is found
         {
            // Get the employee's full name and salary
            empName = reader.GetString(0) + " "
            +
                      reader.GetString(1) + ". " +
                      reader.GetString(2);
            salary = reader.GetDecimal(3);

            if (bonus == 0)
            {
              if (salary > 75000)
              {
                 bonus = salary * (Decimal)0.025;
                 NullInds[1] = 0; // Return a non-NULL value
              }
              else
              {
                 bonus = salary * (Decimal)0.05;
                 NullInds[1] = 0; // Return a non-NULL value
              }
            }
         }
         else  // Employee not found
         {
            empName = "*sdq;;         // Set output parameter
            NullInds[2] = -1;     // Return a NULL value
         }

         reader.Close();
      }
   }
```

Example 3: C# parameter style SQL procedure:
This example shows the following:

- CREATE PROCEDURE statement for a parameter style SQL procedure
- C# code for a parameter style SQL procedure

This procedure takes an employee ID and a current bonus amount as input. It retrieves the employee's name and salary. If the current bonus amount is zero, a new bonus based on salary is calculated and returned along with the employee's full name. If the employee is not found, an empty string is returned.

Table 15. Code to create a C# procedure in parameter style SQL with parameters

```
CREATE PROCEDURE SetEmpbonusSQL(IN empID CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpbonusSQL
LANGUAGE CLR
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
MODIFIES SQL DATA
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusSQL' ;
```

Table 15. Code to create a C# procedure in parameter style SQL with parameters (continued)

```
public static void SetEmpBonusSQL(    String empID,
                                  ref Decimal bonus,
                                  out String empName,
                                      Int16  empIDNullInd,
                                  ref Int16  bonusNullInd,
                                  out Int16  empNameNullInd,
                                  ref string sqlStateate,
                                      string funcName,
                                      string specName,
                                  ref string sqlMessageText)
{
   // Declare local host variables
   Decimal salary eq; 0;

   if (empIDNullInd == -1) // Check if the input is null
   {
      bonusNullInd = -1;   // Return a NULL bonus value
      empName = "";
      empNameNullInd = -1; // Return a NULL empName value
   }
   else
      DB2Command myCommand = DB2Context.GetCommand();
      myCommand.CommandText =
                       "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY
                       "
                     + "FROM EMPLOYEE "
                     + "WHERE EMPNO = '" + empID + "'";

      DB2DataReader reader = myCommand.ExecuteReader();

      if (reader.Read())  // If employee record is found
      {
         // Get the employee's full name and salary
         empName = reader.GetString(0) + " "
         +
         reader.GetString(1) + ". " +
         reader.GetString(2);
         empNameNullInd = 0;
         salary = reader.GetDecimal(3);

         if (bonus == 0)
         {
            if (salary > 75000)
            {
               bonus = salary * (Decimal)0.025;
               bonusNullInd = 0;  // Return a non-NULL value
            }
            else
            {
               bonus = salary * (Decimal)0.05;
               bonusNullInd = 0;  // Return a non-NULL value
            }
         }
      }
      else  // Employee not found
      }
         empName = "";         // Set output parameter
         empNameNullInd = -1;  // Return a NULL value
      }

      reader.Close();
   }
}
```

Example 4: C# parameter style GENERAL procedure returning a result set: This example shows the following:

- CREATE PROCEDURE statement for an external C# procedure returning a result set
- C# code for a parameter style GENERAL procedure that returns a result set

This procedure accepts the name of a table as a parameter. It returns a result set containing all the rows of the table specified by the input parameter. This is done by leaving a `DB2DataReader` for a given query result set open when the procedure returns. Specifically, if `reader.Close()` is not executed, the result set will be returned.

Table 16. Code to create a C# procedure that returns a result set

```
CREATE PROCEDURE ReturnResultSet(IN tableName
VARCHAR(20))
SPECIFIC ReturnResultSet
DYNAMIC RESULT SETS 1
LANGUAGE CLR
PARAMETER STYLE GENERAL
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME
'gwenProc.dll:bizLogic.empOps!ReturnResultSet' ;
```

```
 public static void ReturnResultSet(string tableName)
 {
    DB2Command myCommand = DB2Context.GetCommand();

    // Set the SQL statement to be executed and execute it.
    myCommand.CommandText = "SELECT * FROM " + tableName;
    DB2DataReader reader = myCommand.ExecuteReader();

    // The DB2DataReader contains the result of the query.
    // This result set can be returned with the procedure,
    // by simply NOT closing the DB2DataReader.
    // Specifically, do NOT execute reader.Close();
 }
```

Example 5: C# parameter style SQL procedure accessing the dbinfo structure: This example shows the following:

- CREATE PROCEDURE statement for a procedure accessing the `dbinfo` structure
- C# code for a parameter style SQL procedure that accesses the `dbinfo` structure

To access the `dbinfo` structure, the DBINFO clause must be specified in the CREATE PROCEDURE statement. No parameter is required for the `dbinfo` structure in the CREATE PROCEDURE statement however a parameter must be created for it, in the external routine code. This procedure returns only the value of the current database name from the `dbname` field in the `dbinfo` structure.

Table 17. Code to create a C# procedure that accesses the dbinfo structure

```
CREATE PROCEDURE ReturnDbName(OUT dbName VARCHAR(20))
SPECIFIC ReturnDbName
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE SQL
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
DBINFO
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!ReturnDbName'
;
```

```
 public static void ReturnDbName(out string dbName,
                                out Int16  dbNameNullInd,
                                ref string sqlStateate,
                                    string funcName,
                                    string specName,
                                ref string sqlMessageText,
                                    sqludf_dbinfo dbinfo)
{
   // Retrieve the current database name from the
   // dbinfo structure and return it.
   // ** Note! ** dbinfo field names are case sensitive
   dbName = dbinfo.dbname;
   dbNameNullInd = 0;  // Return a non-null value;

   // If you want to return a user-defined error in
   // the SQLCA you can specify a 5 digit user-defined
   // sqlStateate and an error message string text.
   // For example:
   //
   //   sqlStateate = "ABCDE";
   //   sqlMessageText = "A user-defined error has occured"
   //
   //  DB2 returns the above values to the client in the
   //  SQLCA structure.  The values are used to generate a
   //  standard DB2 sqlStateate error.
}
```

Example 6: C# procedure with PROGRAM TYPE MAIN style:
This example shows the following:

- CREATE PROCEDURE statement for a procedure using a main program style
- C# parameter style GENERAL WITH NULLS code in using a MAIN program style

To implement a routine in a main program style, the PROGRAM TYPE clause must be specified in the CREATE PROCEDURE statement with the value MAIN. Parameters are specified in the CREATE PROCEDURE statement however in the code implementation, parameters are passed into the routine in an `argc` integer parameter and an `argv` array of parameters.

Table 18. Code to create a C# procedure in program type MAIN style

```
CREATE PROCEDURE MainStyle( IN empID CHAR(6),
                            INOUT bonus Decimal(9,2),
                            OUT empName VARCHAR(60))
SPECIFIC MainStyle
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
MODIFIES SQL DATA
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
PROGRAM TYPE MAIN
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!main' ;
```

Table 18. Code to create a C# procedure in program type MAIN style (continued)

```
public static void main(Int32 argc, Object[]
argv)
{
  String empID = (String)argv[0];   // argv[0] has nullInd:argv[3]
  Decimal bonus = (Decimal)argv[1]; // argv[1] has nullInd:argv[4]
                                    // argv[2] has nullInd:argv[5]
  Decimal salary = 0;     Int16[] NullInds =
  (Int16[])argv[3];

  if ((NullInds[0]) == (Int16)(-1)) // Check if empID is null
  {
     NullInds[1] = (Int16)(-1);  // Return a NULL bonus value
     argv[1] = (String)"";       // Set output parameter empName
     NullInds[2] = (Int16)(-1);  // Return a NULL empName value
     Return;
  }
  else
     DB2Command myCommand = DB2Context.GetCommand();
     myCommand.CommandText =
                       "SELECT FIRSTNME, MIDINIT, LASTNAME, salary "
                     + "FROM EMPLOYEE "
                     + "WHERE EMPNO = '" + empID + "'";

     DB2DataReader reader = myCommand.ExecuteReader();

     if (reader.Read())  // If employee record is found
     {
       // Get the employee's full name and salary
       argv[2] = (String) (reader.GetString(0) + " " +
                          reader.GetString(1) + ".
                          " +
                          reader.GetString(2));
       NullInds[2] = (Int16)0;
       salary = reader.GetDecimal(3);

       if (bonus == 0)
       {
          if (salary > 75000)
          {
             argv[1] = (Decimal)(salary * (Decimal)0.025);
             NullInds[1] = (Int16)(0);  // Return a non-NULL value
          }
          else
          {
             argv[1] = (Decimal)(salary * (Decimal)0.05);
             NullInds[1] = (Int16)(0);  // Return a non-NULL value
          }
       }
     }
     else  // Employee not found
     {
        argv[2] = (String)("");         // Set output parameter
        NullInds[2] = (Int16)(-1);      // Return a NULL value
     }

     reader.Close();
   }
}
```

Related concepts:

- ".NET common language runtime (CLR) routines" on page 57
- "Benefits of using routines" on page 20

Related tasks:

- "Examples of C# .NET CLR functions" on page 116
- "Creating .NET CLR routines from DB2 Command Windows" on page 69
- "Building Common Language Runtime (CLR) .NET routines" on page 72
- "Examples of .NET CLR routines" on page 84

Related samples:

- "SpCreate.db2 -- Creates the external procedures implemented in spserver.cs"
- "SpServer.cs -- C# external code implementation of procedures created in spcat.db2"
- "SpCreate.db2 -- Creates the external procedures implemented in spserver.vb"
- "SpServer.vb -- VB.NET implementation of procedures created in SpCat.db2"

Example: XML and XQuery support in C# .NET CLR procedure

Once the basics of procedures, the essentials of .NET common language runtime routines, XQuery and XML are understood, you can start creating and using CLR procedures with XML features.

The example below demonstrates a C# .NET CLR procedure with parameters of type XML as well as how to update and query XML data.

Prerequisites:

Before working with the CLR procedure example you might want to read the following concept topics:

- Common language runtime (CLR) routines
- Creating CLR routines
- Benefits of using routines
- Building common language runtime (CLR) .NET routines

The examples below makes use of a table named `xmltable` that is defined as follows:

```
CREATE TABLE xmltable
(
   num INTEGER,
   xdata XML
)

INSERT INTO xmltable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                            <type>car</type>
                            <make>Pontiac</make>
                            <model>Sunfire</model>
                            </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                            <type>car</type>
                            <make>Mazda</make>
                            <model>Miata</model>
                            </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                            <type>person</type>
                            <name>Mary</name>
                            <town>Vancouver</town>
                            <street>Waterside</street>
                            </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                             <type>person</type>
                             <name>Mark</name>
                             <town>Edmonton</town>
                             <street>Oak</street>
                             </doc>' PRESERVE WHITESPACE)),
(5, XMLPARSE(DOCUMENT '<doc>
                             <type>animal</type>
                             <name>dog</name>
                             </doc>' PRESERVE WHITESPACE)),
(6, NULL),
(7, XMLPARSE(DOCUMENT '<doc>
                             <type>car</type>
                             <make>Ford</make>
                             <model>Taurus</model>
                             </doc>' PRESERVE WHITESPACE)),
(8, XMLPARSE(DOCUMENT '<doc>
                             <type>person</type>
                             <name>Kim</name>
                             <town>Toronto</town>
                             <street>Elm</street>
                             </doc>' PRESERVE WHITESPACE)),
(9, XMLPARSE(DOCUMENT '<doc>
                             <type>person</type>
                             <name>Bob</name>
                             <town>Toronto</town>
                             <street>Oak</street>
                             </doc>' PRESERVE WHITESPACE)),
(10, XMLPARSE(DOCUMENT '<doc>
                              <type>animal</type>
                              <name>bird</name>
                              </doc>' PRESERVE WHITESPACE))@
```

Procedure:

Use the following examples as references when making your own C# CLR procedures:

- "The C# external code file"
- "Example 1: C# parameter style GENERAL procedure with XML features" on page 107

The C# external code file:

The example consists of two parts: the CREATE PROCEDURE statement and the external C# code implementation of the procedure from which the associated assembly can be built.

The C# source file that contains the procedure implementations of the following examples is named `gwenProc.cs` and has the following format:

Table 19. C# external code file format

```
using System;
using System.IO;
using System.Data;
using IBM.Data.DB2;
using IBM.Data.DB2Types;

namespace bizLogic
{
   class empOps
   {          ...
     // C# procedures
              ...
   }
}
```

The file inclusions are indicated at the top of the file. The `IBM.Data.DB2` inclusion is required if any of the procedures in the file contain SQL. The `IBM.Data.DB2Types` inclusion is required if any of the procedures in the file contains parameters or variables of type XML. There is a namespace declaration in this file and a class `empOps` that contains the procedures. The use of namespaces is optional. If a namespace is used, the namespace must appear in the assembly path name provided in the EXTERNAL clause of the CREATE PROCEDURE statement.

It is important to note the name of the file, the namespace, and the name of the class, that contains a given procedure implementation. These names are important, because the EXTERNAL clause of the CREATE PROCEDURE statement for each procedure must specify this information so that DB2 can locate the assembly and class of the CLR procedure.

Example 1: C# parameter style GENERAL procedure with XML features:

This example shows the following:
- CREATE PROCEDURE statement for a parameter style GENERAL procedure
- C# code for a parameter style GENERAL procedure with XML parameters

This procedure takes two parameters, an integer `inNum` and `inXML`. These values are inserted into the table `xmltable`. Then an XML value is retrieved using XQuery. Another XML value is retrieved using SQL. The retrieved XML values are assigned to two output parameters, `outXML1` and `outXML2`. No result sets are returned.

Table 20. Code to create a C# parameter style GENERAL procedure

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                             IN inXML  XML as CLOB (1K),
                             OUT inXML  XML as CLOB (1K),
                             OUT inXML  XML as CLOB (1K)
                           )
LANGUAGE CLR
PARAMETER STYLE GENERAL
DYNAMIC RESULT SETS 0
FENCED
THREADSAFE
DETERMINISTIC
NO DBINFO
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!xmlProc1' ;
```

Table 20. Code to create a C# parameter style GENERAL procedure (continued)

```
//*************************************************************************
//  Stored Procedure: xmlProc1
//
//  Purpose:   insert XML data into XML column
//
//  Parameters:
//
//   IN:    inNum -- the sequence of XML data to be insert in xmldata table
//          inXML -- XML data to be inserted
//   OUT:   outXML1 -- XML data returned - value retrieved using XQuery
//          outXML2 -- XML data returned - value retrieved using SQL
//*************************************************************************
```

Table 20. Code to create a C# parameter style GENERAL procedure (continued)

```
public static void xmlProc1 (     int      inNum, DB2Xml   inXML,
                              out DB2Xml   outXML1, out DB2Xml   outXML2 )
{
   // Create new command object from connection context
   DB2Parameter parm;
   DB2Command cmd;
   DB2DataReader reader = null;
   outXML1 = DB2Xml.Null;
   outXML2 = DB2Xml.Null;

   // Insert input XML parameter value into a table
   cmd = DB2Context.GetCommand();
   cmd.CommandText = "INSERT INTO "
                   + "xmltable( num , xdata ) "
                   + "VALUES( ?, ? )";

   parm = cmd.Parameters.Add("@num", DB2Type.Integer );
   parm.Direction = ParameterDirection.Input;
   cmd.Parameters["@num"].Value = inNum;
   parm = cmd.Parameters.Add("@data", DB2Type.Xml);
   parm.Direction = ParameterDirection.Input;
   cmd.Parameters["@data"].Value = inXML ;
   cmd.ExecuteNonQuery();
   cmd.Close();

   // Retrieve XML value using XQuery
                and assign value to an XML output parameter
   cmd = DB2Context.GetCommand();
   cmd.CommandText = "XQUERY for $x " +
                   "in db2-fn:xmlcolumn(\"XMLTABLE.XDATA\")/doc "+
                   "where $x/make = \'Mazda\' " +
                   "return <carInfo>{$x/make}{$x/model}</carInfo>";
   reader = cmd.ExecuteReader();
   reader.CacheData= true;

   if (reader.Read())
   {  outXML1 = reader.GetDB2Xml(0);  }
   else
   {  outXML1 = DB2Xml.Null;  }

   reader.Close();
   cmd.Close();

   // Retrieve XML value using SQL
                and assign value to an XML output parameter value
   cmd = DB2Context.GetCommand();
   cmd.CommandText = "SELECT xdata "
                    + "FROM xmltable "
                    + "WHERE num = ?";

   parm = cmd.Parameters.Add("@num", DB2Type.Integer );
   parm.Direction = ParameterDirection.Input;
   cmd.Parameters["@num"].Value = inNum;
   reader = cmd.ExecuteReader();
   reader.CacheData= true;

   if (reader.Read())
   {  outXML2 = reader.GetDB2Xml(0);  }
   else
   {  outXML = DB2Xml.Null;  }

   reader.Close() ;
   cmd.Close();

   return;
}
```

Related concepts:
- ".NET common language runtime (CLR) routines" on page 57
- "Benefits of using routines" on page 20

Related tasks:
- "Building Common Language Runtime (CLR) .NET routines" on page 72
- "Creating .NET CLR routines from DB2 Command Windows" on page 69

Examples of Visual Basic .NET CLR functions

Once you understand the basics of user-defined functions (UDFs), and the essentials of CLR routines, you can start exploiting CLR UDFs in your applications and database environment. This topic contains some examples of CLR UDFs to get you started. For examples of CLR procedures in Visual Basic:
- "Examples of Visual Basic .NET CLR procedures" on page 85

Prerequisites:

Before working with the CLR UDF examples you may want to read the following concept topics:
- ".NET common language runtime (CLR) routines" on page 57
- "Creating .NET CLR routines from DB2 Command Windows" on page 69
- "External scalar functions" on page 23
- Routines: table user-defined functions
- Building common language runtime (CLR) .NET routines

The examples below make use of a table named `EMPLOYEE` that is contained in the `SAMPLE` database.

Procedure:

Use the following examples as references when making your own Visual Basic CLR UDFs:
- "The Visual Basic external code file"
- "Example 1: Visual Basic parameter style SQL table function" on page 111
- "Example 2: Visual Basic parameter style SQL scalar function" on page 113

The Visual Basic external code file:

The following examples show a variety of Visual Basic UDF implementations. The CREATE FUNCTION statement is provided for each UDF with the corresponding Visual Basic source code from which the associated assembly can be built. The Visual Basic source file that contains the functions declarations used in the following examples is named `gwenVbUDF.cs` and has the following format:

Table 21. Visual Basic external code file format

```
  using System;
  using System.IO;
  using IBM.Data.DB2;

  Namespace bizLogic

     ...
     ' Class definitions that contain UDF declarations
     ' and any supporting class definitions
     ...

End Namespace
```

The function declarations must be contained in a class within a Visual Basic file. The use of namespaces is optional. If a namespace is used, the namespace must appear in the assembly path name provided in the EXTERNAL clause of the CREATE PROCEDURE statement. The `IBM.Data.DB2.` inclusion is required if the function contains SQL.

Example 1: Visual Basic parameter style SQL table function:

This example shows the following:

- CREATE FUNCTION statement for a parameter style SQL table function
- Visual Basic code for a parameter style SQL table function

This table function returns a table containing rows of employee data that was created from a data array. There are two classes associated with this example. Class `person` represents the employees, and the class `empOps` contains the routine table UDF that uses class `person`. The employee salary information is updated based on the value of an input parameter. The data array in this example is created within the table function itself on the first call of the table function. Such an array could have also been created by reading in data from a text file on the filesystem. The array data values are written to a scratchpad so that the data can be accessed in subsequent calls of the table function.

On each call of the table function, one record is read from the array and one row is generated in the table that is returned by the function. The row is generated in the table, by setting the output parameters of the table function to the desired row values. After the final call of the table function occurs, the table of generated rows is returned.

Table 22. Code to create a Visual Basic parameter style SQL table function

```
  CREATE FUNCTION TableUDF(double)
  RETURNS TABLE (name varchar(20),
                 job varchar(20),
                 salary double)
  EXTERNAL NAME 'gwenVbUDF.dll:bizLogic.empOps!TableUDF'
  LANGUAGE CLR
  PARAMETER STYLE SQL
  NOT DETERMINISTIC
  FENCED
  SCRATCHPAD 10
  FINAL CALL
  DISALLOW PARALLEL
  NO DBINFO
  EXECUTION CONTROL SAFE
```

Table 22. Code to create a Visual Basic parameter style SQL table function (continued)

```
Class Person
' The class Person is a supporting class for
' the table function UDF, tableUDF, below.

  Private name As String
  Private position As String
  Private salary As Int32

  Public Sub New(ByVal newName As String, _
                 ByVal newPosition As String, _
                 ByVal newSalary As Int32)

    name = newName
    position = newPosition
    salary = newSalary
  End Sub

  Public Property GetName() As String
    Get
      Return name
    End Get

    Set (ByVal value As String)
      name = value
    End Set
  End Property

  Public Property GetPosition() As String
    Get
      Return position
    End Get

    Set (ByVal value As String)
      position = value
    End Set
  End Property

  Public Property GetSalary() As Int32
    Get
      Return salary
    End Get

    Set (ByVal value As Int32)
      salary = value
    End Set
  End Property

End Class
```

Table 22. Code to create a Visual Basic parameter style SQL table function (continued)

```
Class empOps

  Public Shared Sub TableUDF(byVal factor as Double, _
                             byRef name As String, _
                             byRef position As String, _
                             byRef salary As Double, _
                             byVal factorNullInd As Int16, _
                             byRef nameNullInd As Int16, _
                             byRef positionNullInd As Int16, _
                             byRef salaryNullInd As Int16, _
                             byRef sqlState As String, _
                             byVal funcName As String, _
                             byVal specName As String, _
                             byRef sqlMessageText As String, _
                             byVal scratchPad As Byte(), _
                             byVal callType As Int32)

    Dim intRow As Int16

    intRow = 0

    ' Create an array of Person type information
    Dim staff(2) As Person
    staff(0) = New Person("Gwen", "Developer", 10000)
    staff(1) = New Person("Andrew", "Developer", 20000)
    staff(2) = New Person("Liu", "Team Leader", 30000)

    ' Initialize output parameter values and NULL indicators
    salary = 0
    name = position = ""
    nameNullInd = positionNullInd = salaryNullInd = -1

    Select callType
       Case -2   ' Case SQLUDF_TF_FIRST:
       Case -1   ' Case SQLUDF_TF_OPEN:
         intRow = 1
         scratchPad(0) = intRow  ' Write to scratchpad
       Case 0    ' Case SQLUDF_TF_FETCH:
         intRow = scratchPad(0)
         If intRow > staff.Length
            sqlState = "02000"  ' Return an error SQLSTATE
         Else
            ' Generate a row in the output table
            ' based on the staff array data.
            name = staff(intRow).GetName()
            position = staff(intRow).GetPosition()
            salary = (staff(intRow).GetSalary()) * factor
            nameNullInd = 0
            positionNullInd = 0
            salaryNullInd = 0
         End If
         intRow = intRow + 1
         scratchPad(0) = intRow  ' Write scratchpad

       Case 1    ' Case SQLUDF_TF_CLOSE:

       Case 2    ' Case SQLUDF_TF_FINAL:
    End Select

  End Sub

End Class
```

Example 2: Visual Basic parameter style SQL scalar function:
This example shows the following:

- CREATE FUNCTION statement for a parameter style SQL scalar function
- Visual Basic code for a parameter style SQL scalar function

This scalar function returns a single count value for each input value that it operates on. For an input value in the nth position of the set of input values, the output scalar value is the value n. On each call of the scalar function, where one call is associated with each row or value in the input set of rows or values, the count is increased by one and the current value of the count is returned. The count is then saved in the scratchpad memory buffer to maintain the count value between each call of the scalar function.

This scalar function can be easily invoked if for example we have a table defined as follows:

```
CREATE TABLE T (i1 INTEGER);
INSERT INTO T VALUES 12, 45, 16, 99;
```

A simple query such as the following can be used to invoke the scalar function:

```
SELECT my_count(i1) as count, i1 FROM T;
```

The output of such a query would be:

```
COUNT          I1
-----------    ----------
1              12
2              45
3              16
4              99
```

This scalar UDF is quite simple. Instead of returning just the count of the rows, you could use a scalar function to format data in an existing column. For example you might append a string to each value in an address column or you might build up a complex string from a series of input strings or you might do a complex mathematical evaluation over a set of data where you must store an intermediate result.

Table 23. Code to create a Visual Basic parameter style SQL scalar function

```
CREATE FUNCTION mycount(INTEGER)
RETURNS INTEGER
LANGUAGE CLR
PARAMETER STYLE SQL
NO SQL
SCRATCHPAD 10
FINAL CALL
FENCED
EXECUTION CONTROL SAFE
NOT DETERMINISTIC
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!CountUp';
```

Table 23. Code to create a Visual Basic parameter style SQL scalar function (continued)

```
Class empOps
  Public Shared Sub CountUp(byVal input As Int32, _
                            byRef outCounter As Int32, _
                            byVal nullIndInput As Int16, _
                            byRef nullIndOutCounter As Int16, _
                            byRef sqlState As String, _
                            byVal qualName As String, _
                            byVal specName As String, _
                            byRef sqlMessageText As String, _
                            byVal scratchPad As Byte(), _
                            byVal callType As Int32)

     Dim counter As Int32
     counter = 1

     Select callType
        case -1           ' case SQLUDF_TF_OPEN_CALL
           scratchPad(0) = counter
           outCounter = counter
           nullIndOutCounter = 0
        case 0              'case SQLUDF_TF_FETCH_CALL:
           counter = scratchPad(0)
           counter = counter + 1
           outCounter = counter
           nullIndOutCounter = 0
           scratchPad(0) = counter
        case 1              'case SQLUDF_CLOSE_CALL:
           counter = scratchPad(0)
           outCounter = counter
           nullIndOutCounter = 0
        case Else          ' Should never enter here
           ' These cases won't occur for the following reasons:
           ' Case -2  (SQLUDF_TF_FIRST)     —>No FINAL CALL in CREATE stmt
           ' Case 2   (SQLUDF_TF_FINAL)     —>No FINAL CALL in CREATE stmt
           ' Case 255 (SQLUDF_TF_FINAL_CRA) —>No SQL used in the function
           '
           ' * Note!*
           ' ---------
           ' The Else is required so that at compile time
           ' out parameter outCounter is always set *
           outCounter = 0
           nullIndOutCounter = -1
     End Select
  End Sub

End Class
```

Related concepts:

- ".NET common language runtime (CLR) routines" on page 57
- "External scalar functions" on page 23
- "User-defined table functions" in *Developing SQL and External Routines*

Related tasks:

- "Examples of Visual Basic .NET CLR procedures" on page 85
- "Creating .NET CLR routines from DB2 Command Windows" on page 69
- "Building Common Language Runtime (CLR) .NET routines" on page 72
- "Examples of .NET CLR routines" on page 84
- "Examples of C# .NET CLR functions" on page 116
- "Examples of C# .NET CLR procedures" on page 94

Examples of C# .NET CLR functions

Once you understand the basics of user-defined functions (UDFs), and the essentials of CLR routines, you can start exploiting CLR UDFs in your applications and database environment. This topic contains some examples of CLR UDFs to get you started. For examples of CLR procedures in C#:
- "Examples of C# .NET CLR procedures" on page 94

Prerequisites:

Before working with the CLR UDF examples you might want to read the following concept topics:
- ".NET common language runtime (CLR) routines" on page 57
- "Creating .NET CLR routines from DB2 Command Windows" on page 69
- "External scalar functions" on page 23
- Routines: table user-defined functions
- Building common language runtime (CLR) .NET routines

The examples below make use of a table named `EMPLOYEE` that is contained in the `SAMPLE` database.

Procedure:

Use the following examples as references when making your own C# CLR UDFs:
- "The C# external code file"
- "Example 1: C# parameter style SQL table function"
- "Example 2: C# parameter style SQL scalar function" on page 120

The C# external code file:

The following examples show a variety of C# UDF implementations. The CREATE FUNCTION statement is provided for each UDF with the corresponding C# source code from which the associated assembly can be built. The C# source file that contains the functions declarations used in the following examples is named `gwenUDF.cs` and has the following format:

Table 24. C# external code file format

```
using System;
using System.IO;
using IBM.Data.DB2;

namespace bizLogic
{
   ...
   // Class definitions that contain UDF declarations
   // and any supporting class definitions
   ...
}
```

The function declarations must be contained in a class within a C# file. The use of namespaces is optional. If a namespace is used, the namespace must appear in the assembly path name provided in the EXTERNAL clause of the CREATE PROCEDURE statement. The `IBM.Data.DB2.` inclusion is required if the function contains SQL.
Example 1: C# parameter style SQL table function:
This example shows the following:

- CREATE FUNCTION statement for a parameter style SQL table function
- C# code for a parameter style SQL table function

This table function returns a table containing rows of employee data that was created from a data array. There are two classes associated with this example. Class `person` represents the employees, and the class `empOps` contains the routine table UDF that uses class `person`. The employee salary information is updated based on the value of an input parameter. The data array in this example is created within the table function itself on the first call of the table function. Such an array could have also been created by reading in data from a text file on the filesystem. The array data values are written to a scratchpad so that the data can be accessed in subsequent calls of the table function.

On each call of the table function, one record is read from the array and one row is generated in the table that is returned by the function. The row is generated in the table, by setting the output parameters of the table function to the desired row values. After the final call of the table function occurs, the table of generated rows is returned.

Table 25. Code to create a C# parameter style SQL table function

```
CREATE FUNCTION tableUDF(double)
RETURNS TABLE (name varchar(20),
               job varchar(20),
               salary double)
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!tableUDF'
LANGUAGE CLR
PARAMETER STYLE SQL
NOT DETERMINISTIC
FENCED
THREADSAFE
SCRATCHPAD 10
FINAL CALL
EXECUTION CONTROL SAFE
DISALLOW PARALLEL
NO DBINFO
```

Table 25. Code to create a C# parameter style SQL table function (continued)

```
// The class Person is a supporting class for
// the table function UDF, tableUDF, below.
class Person
{
     private String name;
     private String position;
     private Int32 salary;

     public Person(String newName, String newPosition, Int32
     newSalary)
     {
        this.name = newName;
        this.position = newPosition;
        this.salary = newSalary;
     }

     public String getName()
     {
        return this.name;
     }

     public String getPosition()
     {
        return this.position;
     }

     public Int32 getSalary()
     {
        return this.salary;
     }
 }
```

Table 25. Code to create a C# parameter style SQL table function (continued)

```
class empOps
{
  {
   public static void TableUDF( Double factor, out String name,
                     out String position, out Double salary,
                     Int16 factorNullInd, out Int16 nameNullInd,
                     out Int16 positionNullInd, out Int16 salaryNullInd,
                     ref String sqlState, String funcName,
                     String specName, ref String sqlMessageText,
                     Byte[] scratchPad, Int32 callType)
   {

      Int16 intRow = 0;

      // Create an array of Person type information
      Person[] Staff = new
      Person[3];
      Staff[0] = new Person("Gwen", "Developer", 10000);
      Staff[1] = new Person("Andrew", "Developer", 20000);
      Staff[2] = new Person("Liu", "Team Leader", 30000);

      salary = 0;
      name = position = "";
      nameNullInd = positionNullInd = salaryNullInd = -1;

      switch(callType)
      {
         case (-2):  // Case SQLUDF_TF_FIRST:
           break;

         case (-1):  // Case SQLUDF_TF_OPEN:
           intRow = 1;
           scratchPad[0] = (Byte)intRow;  // Write to scratchpad
           break;
         case (0):   // Case SQLUDF_TF_FETCH:
           intRow = (Int16)scratchPad[0];
           if (intRow > Staff.Length)
           {
              sqlState = "02000";  // Return an error SQLSTATE
           }
           else
           {
              // Generate a row in the output table
              // based on the Staff array data.
              name =
              Staff[intRow-1].getName();
              position = Staff[intRow-1].getPosition();
              salary = (Staff[intRow-1].getSalary[]] * factor;
              nameNullInd = 0;
              positionNullInd = 0;
              salaryNullInd = 0;
           }
           intRow++;
           scratchPad[0] = (Byte)intRow;  // Write scratchpad
           break;

         case (1):   // Case SQLUDF_TF_CLOSE:
           break;

         case (2):   // Case SQLUDF_TF_FINAL:
           break;
      }
   }
}
```

Example 2: C# parameter style SQL scalar function:
This example shows the following:

- CREATE FUNCTION statement for a parameter style SQL scalar function
- C# code for a parameter style SQL scalar function

This scalar function returns a single count value for each input value that it operates on. For an input value in the nth position of the set of input values, the output scalar value is the value n. On each call of the scalar function, where one call is associated with each row or value in the input set of rows or values, the count is increased by one and the current value of the count is returned. The count is then saved in the scratchpad memory buffer to maintain the count value between each call of the scalar function.

This scalar function can be easily invoked if for example we have a table defined as follows:

```
CREATE TABLE T (i1 INTEGER);
INSERT INTO T VALUES 12, 45, 16, 99;
```

A simple query such as the following can be used to invoke the scalar function:

```
SELECT countUp(i1) as count, i1 FROM T;
```

The output of such a query would be:

```
COUNT           I1
-----------     ----------
1               12
2               45
3               16
4               99
```

This scalar UDF is quite simple. Instead of returning just the count of the rows, you could use a scalar function to format data in an existing column. For example you might append a string to each value in an address column or you might build up a complex string from a series of input strings or you might do a complex mathematical evaluation over a set of data where you must store an intermediate result.

Table 26. Code to create a C# parameter style SQL scalar function

```
CREATE FUNCTION countUp(INTEGER)
RETURNS INTEGER
LANGUAGE CLR
PARAMETER STYLE SQL
SCRATCHPAD 10
FINAL CALL
NO SQL
FENCED
THREADSAFE
NOT DETERMINISTIC
EXECUTION CONTORL SAFE
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!CountUp' ;
```

Table 26. Code to create a C# parameter style SQL scalar function (continued)

```
class empOps
{
   public static void CountUp(     Int32 input,
                               out Int32 outCounter,
                                   Int16 inputNullInd,
                               out Int16 outCounterNullInd,
                               ref String sqlState,
                                   String funcName,
                                   String specName,
                               ref String sqlMessageText,
                                   Byte[] scratchPad,
                                   Int32 callType)
   {
      Int32 counter = 1;        switch(callType)
      {
         case -1: // case SQLUDF_FIRST_CALL
           scratchPad[0] = (Byte)counter;
           outCounter = counter;
           outCounterNullInd = 0;
           break;
         case 0:  // case SQLUDF_NORMAL_CALL:
           counter = (Int32)scratchPad[0];
           counter = counter + 1;
           outCounter = counter;
           outCounterNullInd = 0;
           scratchPad[0] =
           (Byte)counter;
           break;
         case 1:  // case SQLUDF_FINAL_CALL:
           counter =
           (Int32)scratchPad[0];
           outCounter = counter;
           outCounterNullInd = 0;
           break;
         default: // Should never enter here
                  // * Required so that at compile time
                  //   out parameter outCounter is always set *
           outCounter = (Int32)(0);
           outCounterNullInd = -1;
           sqlState="ABCDE";
           sqlMessageText = "Should not get here: Default
           case!";
           break;
      }
   }
}
```

Related concepts:

- ".NET common language runtime (CLR) routines" on page 57
- "External scalar functions" on page 23
- "User-defined table functions" in *Developing SQL and External Routines*

Related tasks:

- "Examples of C# .NET CLR procedures" on page 94
- "Creating .NET CLR routines from DB2 Command Windows" on page 69
- "Building Common Language Runtime (CLR) .NET routines" on page 72
- "Examples of .NET CLR routines" on page 84

Related samples:

- “SpCreate.db2 -- Creates the external procedures implemented in spserver.cs”
- “SpServer.cs -- C# external code implementation of procedures created in spcat.db2”
- “SpCreate.db2 -- Creates the external procedures implemented in spserver.vb”
- “SpServer.vb -- VB.NET implementation of procedures created in SpCat.db2”

Chapter 6. IBM OLE DB Provider for DB2

IBM OLE DB Provider for DB2

Microsoft OLE DB is a set of OLE/COM interfaces that provides applications with uniform access to data stored in diverse information sources. The OLE DB architecture defines OLE DB consumers and OLE DB providers. An OLE DB consumer is any system or application that uses OLE DB interfaces; an OLE DB provider is a component that exposes OLE DB interfaces.

The IBM OLE DB Provider for DB2 allows DB2 to act as a resource manager for the OLE DB provider. This support gives OLE DB-based applications the ability to extract or query DB2 data using the OLE interface. The IBM OLE DB Provider for DB2, whose provider name is IBMDADB2, enables OLE DB consumers to access data on a DB2 database server. If DB2 Connect is installed, these OLE DB consumers can also access data on a host DBMS such as DB2 for MVS™, DB2 for VM/VSE, or SQL/400®.

The IBM OLE DB Provider for DB2 offers the following features:

- Support level 0 of the OLE DB provider specification, including some additional level 1 interfaces.
- A free threaded provider implementation, which enables the application to create components in one thread and use those components in any other thread.
- An Error Lookup Service that returns DB2 error messages.

Note that the IBM OLE DB Provider resides on the client and is different from the OLE DB table functions, which are also supported by DB2 database systems.

Subsequent sections of this document describe the specific implementation of the IBM OLE DB Provider for DB2. For more information on the Microsoft OLE DB 2.0 specification, refer to the Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK, available from Microsoft Press.

Version Compliance:

The IBM OLE DB Provider for DB2 complies with Version 2.7 of the Microsoft OLE DB specification.

System Requirements:

Refer to the announcement letter for the IBM OLE DB Provider for DB2 Servers to see the supported Windows operating systems.

To install the IBM OLE DB Provider for DB2, you must first be running on one of the supported operating systems listed above. You also need to install the DB2 Client. This client includes Microsoft Data Access Components (MDAC).

Related reference:
- "IBM OLE DB Provider support for OLE DB components and interfaces" on page 132

Application Types Supported by the IBM OLE DB Provider for DB2

With the IBM OLE DB Provider for DB2, you can create the following types of applications:
- ADO applications, including:
  - Microsoft Visual Studio C++ applications
  - Microsoft Visual Basic applications
- ADO.NET applications using the OLE DB .NET Data Provider
- C/C++ applications which access IBMDADB2 directly using the OLE DB interfaces, including ATL applications whose Data Access Consumer Objects were generated by the ATL COM AppWizard.

OLE DB Services

The sections that follow describe OLE DB services.

Thread model supported by the IBM OLE DB Provider

The IBM OLE DB Provider for DB2 supports the Free Threaded model. This allows applications to create components in one thread and use these components in any other thread.

Large object manipulation with the IBM OLE DB Provider

To get and set data as storage objects (`DBTYPE_IUNKNOWN`) with the IBMDADB2 provider, use the ISequentialStream interface as follows:

- To bind a storage object to a parameter, the DBOBJECT in the DBBINDING structure can only contain the value `STGM_READ` for the dwFlag field. IBMDADB2 will execute the Read method of the ISequentialStream interface of the bound object.
- To get data from a storage object, your application must run the Read method on the ISequentialStream interface of the storage object.
- When getting data, the value of the length part is the length of the real data, not the length of the IUnknown pointer.

Schema rowsets supported by the IBM OLE DB Provider

The following table shows the schema rowsets that are supported by `IDBSchemaRowset`. Unsupported columns will be set to `null` in the rowsets.

Table 27. Schema Rowsets Supported by the IBM OLE DB Provider for DB2

Supported GUIDs	**Supported Restrictions**	**Supported Columns**	**Notes**
DBSCHEMA _COLUMN_PRIVILEGES	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	COLUMN_NAME GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_COLUMNS	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH COLUMN_DEFAULT COLUMN_FLAGS COLUMN_HASDEFAULT COLUMN_NAME DATA_TYPE DESCRIPTION IS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_FOREIGN_KEYS	FK_TABLE_NAME FK_TABLE_SCHEMA PK_TABLE_NAME PK_TABLE_SCHEMA	DEFERRABILITY DELETE_RULE FK_COLUMN_NAME FK_NAME FK_TABLE_NAME FK_TABLE_SCHEMA ORDINAL PK_COLUMN_NAME PK_NAME PK_TABLE_NAME PK_TABLE_SCHEMA UPDATE_RULE	Must specify at least one of the following restrictions: PK_TABLE_NAME or FK_TABLE_NAME No “%” wildcard allowed.
DBSCHEMA_INDEXES	TABLE_NAME TABLE_SCHEMA	CARDINALITY CLUSTERED COLLATION COLUMN_NAME INDEX_NAME INDEX_SCHEMA ORDINAL_POSITION PAGES TABLE_NAME TABLE_SCHEMA TYPE UNIQUE	No sort order supported. Sort order, if specified, will be ignored.

Table 27. Schema Rowsets Supported by the IBM OLE DB Provider for DB2 (continued)

Supported GUIDs	Supported Restrictions	Supported Columns	Notes
DBSCHEMA_PRIMARY_KEYS	TABLE_NAME TABLE_SCHEMA	COLUMN_NAME ORDINAL PK_NAME TABLE_NAME TABLE_SCHEMA	Must specify at least the following restrictions: TABLE_NAME No “%” wildcard allowed.
DBSCHEMA _PROCEDURE_PARAMETERS	PARAMETER_NAME PROCEDURE_NAME PROCEDURE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH DATA_TYPE DESCRIPTION IS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION PARAMETER_DEFAULT PARAMETER_HASDEFAULT PARAMETER_NAME PARAMETER_TYPE PROCEDURE_NAME PROCEDURE_SCHEMA TYPE_NAME	
DBSCHEMA_PROCEDURES	PROCEDURE_NAME PROCEDURE_SCHEMA	DESCRIPTION PROCEDURE_NAME PROCEDURE_SCHEMA PROCEDURE_TYPE	
DBSCHEMA_PROVIDER_TYPES	DATA_TYPE BEST_MATCH	AUTO_UNIQUE_VALUE BEST_MATCH CASE_SENSITIVE CREATE_PARAMS COLUMN_SIZE DATA_TYPE FIXED_PREC_SCALE IS_FIXEDLENGTH IS_LONG IS_NULLABLE LITERAL_PREFIX LITERAL_SUFFIX LOCAL_TYPE_NAME MINIMUM_SCALE MAXIMUM_SCALE SEARCHABLE TYPE_NAME UNSIGNED_ATTRIBUTE	
DBSCHEMA_STATISTICS	TABLE_NAME TABLE_SCHEMA	CARDINALITY TABLE_NAME TABLE_SCHEMA	No sort order supported. Sort order, if specified, will be ignored.
DBSCHEMA _TABLE_PRIVILEGES	TABLE_NAME TABLE_SCHEMA	GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_TABLES	TABLE_NAME TABLE_SCHEMA TABLE_TYPE	DESCRIPTION TABLE_NAME TABLE_SCHEMA TABLE_TYPE	

OLE DB services automatically enabled by the IBM OLE DB Provider

By default, the IBM OLE DB Provider for DB2 automatically enables all the OLE DB services by adding a registry entry `OLEDB_SERVICES` under the class ID (CLSID) of the provider with the DWORD value of `0xFFFFFFFF`. The meaning of this value is as follows:

Table 28. OLE DB Services

Enabled Services	DWORD Value
All services (default)	`0xFFFFFFFF`
All except pooling and AutoEnlistment	`0xFFFFFFFC`
All except client cursor	`0xFFFFFFFB`
All except pooling, enlistment and cursor	`0xFFFFFFF8`
No services	`0x000000000`

Data Services

The sections that follow describe data services considerations.

Supported cursor modes for the IBM OLE DB Provider

The IBM OLE DB Provider for DB2 natively supports read-only, forward-only, updatable scrollable, and updatable scrollable cursors.

Data type mappings between DB2 and OLE DB

The IBM OLE DB Provider for DB2 supports data type mappings between DB2 data types and OLE DB data types. The following table provides a complete list of supported mappings and available names for indicating the data types of columns and parameters.

Table 29. Data type mappings between DB2 data types and OLE DB data types

DB2 Data Types	OLE DB Data Types Indicators	OLE DB Standard Type Names	DB2 Specific Names
SMALLINT	DBTYPE_I2	“DBTYPE_I2”	“SMALLINT”
INTEGER	DBTYPE_I4	“DBTYPE_I4”	“INTEGER” or “INT”
BIGINT	DBTYPE_I8	“DBTYPE_I8”	“BIGINT”
REAL	DBTYPE_R4	“DBTYPE_R4”	“REAL”
FLOAT	DBTYPE_R8	“DBTYPE_R8”	“FLOAT”
DOUBLE	DBTYPE_R8	“DBTYPE_R8″	“DOUBLE” or “DOUBLE PRECISION”
DECIMAL	DBTYPE_NUMERIC	“DBTYPE_NUMERIC”	“DEC” or “DECIMAL”
NUMERIC	DBTYPE_NUMERIC	“DBTYPE_NUMERIC”	“NUM” or “NUMERIC”
DATE	DBTYPE_DBDATE	“DBTYPE_DBDATE”	“DATE”
TIME	DBTYPE_DBTIME	“DBTYPE_DBTIME”	“TIME”
TIMESTAMP	DBTYPE_DBTIMESTAMP	“DBTYPE_DBTIMESTAMP”	“TIMESTAMP”
CHAR	DBTYPE_STR	“DBTYPE_CHAR”	“CHAR” or “CHARACTER”
VARCHAR	DBTYPE_STR	“DBTYPE_VARCHAR”	“VARCHAR”
LONG VARCHAR	DBTYPE_STR	“DBTYPE_LONGVARCHAR”	“LONG VARCHAR”
CLOB	DBTYPE_STR and DBCOLUMNFLAGS_ISLONG or DBPARAMFLAGS_ISLONG	“DBTYPE_CHAR” “DBTYPE_VARCHAR” “DBTYPE_LONGVARCHAR” and DBCOLUMNFLAGS_ISLONG or DBPARAMFLAGS_ISLONG	“CLOB”
GRAPHIC	DBTYPE_WSTR	“DBTYPE_WCHAR”	“GRAPHIC”
VARGRAPHIC	DBTYPE_WSTR	“DBTYPE_WVARCHAR”	“VARGRAPHIC”

Table 29. Data type mappings between DB2 data types and OLE DB data types (continued)

DB2 Data Types	OLE DB Data Types Indicators	OLE DB Standard Type Names	DB2 Specific Names
LONG VARGRAPHIC	DBTYPE_WSTR	"DBTYPE_WLONGVARCHAR"	"LONG VARGRAPHIC"
DBCLOB	DBTYPE_WSTR and DBCOLUMNFLAGS_ISLONG or DBPARAMFLAGS_ISLONG	"DBTYPE_WCHAR" "DBTYPE_WVARCHAR" "DBTYPE_WLONGVARCHAR" and DBCOLUMNFLAGS_ISLONG or DBPARAMFLAGS_ISLONG	"DBCLOB"
CHAR(n) FOR BIT DATA	DBTYPE_BYTES	"DBTYPE_BINARY"	
VARCHAR(n) FOR BIT DATA	DBTYPE_BYTES	"DBTYPE_VARBINARY"	
LONG VARCHAR FOR BIT DATA	DBTYPE_BYTES	"DBTYPE_LONGVARBINARY"	
BLOB	DBTYPE_BYTES and DBCOLUMNFLAGS_ISLONG or DBPARAMFLAGS_ISLONG	"DBTYPE_BINARY" "DBTYPE_VARBINARY" "DBTYPE_LONGVARBINARY" and DBCOLUMNFLAGS_ISLONG or DBPARAMFLAGS_ISLONG	"BLOB"
DATA LINK	DBTYPE_STR	"DBTYPE_CHAR"	"DATA LINK"

Data conversion for setting data from OLE DB Types to DB2 Types

The IBM OLE DB Provider for DB2 supports data conversions for setting data from OLE DB types to DB2 types. Note that truncation of the data may occur in some cases, depending on the types and the value of the data.

Table 30. Data conversions from OLE DB types to DB2 types

	DB2 Data Types																					
																		For Bit Data				
OLE DB Type Indicator	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATA LINK
DBTYPE_EMPTY																						
DBTYPE_NULL																						
DBTYPE_RESERVED																						
DBTYPE_I1	X	X	X	X	X	X				X	X											
DBTYPE_I2	X	X	X	X	X	X				X	X											
DBTYPE_I4	X	X	X	X	X	X				X	X											
DBTYPE_I8	X	X	X	X	X	X				X	X											
DBTYPE_UI1	X	X	X	X	X	X				X	X											
DBTYPE_UI2	X	X	X	X	X	X				X	X											

Table 30. Data conversions from OLE DB types to DB2 types (continued)

	DB2 Data Types																					
																		For Bit Data				
OLE DB Type Indicator	**SMALLINT**	**INTEGER**	**BIGINT**	**REAL**	**FLOAT DOUBLE**	**DECIMAL NUMERIC**	**DATE**	**TIME**	**TIMESTAMP**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**CLOB**	**GRAPHIC**	**VARGRAPHIC**	**LONG VARGRAPHIC**	**DBCLOB**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**BLOB**	**DATALINK**
DBTYPE_UI4	X	X	X	X	X	X				X	X											
DBTYPE_UI8	X	X	X	X	X	X				X	X											
DBTYPE_R4	X	X	X	X	X	X				X	X											
DBTYPE_R8	X	X	X	X	X	X				X	X											
DBTYPE_CY																						
DBTYPE_DECIMAL	X	X	X	X	X	X				X	X											
DBTYPE_NUMERIC	X	X	X	X	X	X				X	X											
DBTYPE_DATE																						
DBTYPE_BOOL	X	X	X	X	X	X				X	X											
DBTYPE_BYTES			X			X				X	X	X				X		X	X	X		
DBTYPE_BSTR – to be determined																						
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_WSTR														X	X	X						
DBTYPE_VARIANT – to be determined																						
DBTYPE_IDISPATCH																						
DBTYPE_IUNKNOWN										X	X	X	X	X	X	X	X	X	X	X		
DBTYPE_GUID																						
DBTYPE_ERROR																						
DBTYPE_BYREF																						
DBTYPE_ARRAY																						
DBTYPE_VECTOR																						
DBTYPE_UDT																						
DBTYPE_DBDATE							X		X	X	X											
DBTYPE_DBTIME								X	X	X	X											
DBTYPE_DBTIMESTAMP							X	X	X	X	X											
DBTYPE_FILETIME																						
DBTYPE_PROP_VARIANT																						
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

Related reference:

- "Data conversion for setting data from DB2 types to OLE DB types" on page 130

Data conversion for setting data from DB2 types to OLE DB types

For getting data, the IBM OLE DB Provider allows data conversions from DB2 types to OLE DB types. Note that truncation of the data may occur in some cases, depending on the types and the value of the data.

Table 31. Data conversions from DB2 types to OLE DB types

	DB2 Data Types																					
																		For Bit Data				
OLE DB Type Indicator	**SMALLINT**	**INTEGER**	**BIGINT**	**REAL**	**FLOAT DOUBLE**	**DECIMAL NUMERIC**	**DATE**	**TIME**	**TIMESTAMP**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**CLOB**	**GRAPHIC**	**VARGRAPHIC**	**LONG VARGRAPHIC**	**DBCLOB**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**BLOB**	**DATA LINK**
DBTYPE_EMPTY																						
DBTYPE_NULL																						
DBTYPE_RESERVED																						
DBTYPE_I1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R8	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_CY	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_DECIMAL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_NUMERIC	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_DATE	X	X		X	X		X	X	X	X	X	X		X	X	X						X
DBTYPE_BOOL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_BYTES	X	X		X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_BSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_WSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_VARIANT	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_IDISPATCH																						
DBTYPE_IUNKNOWN	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
DBTYPE_GUID										X	X	X		X	X	X		X	X	X		X
DBTYPE_ERROR																						
DBTYPE_BYREF																						

Table 31. Data conversions from DB2 types to OLE DB types (continued)

	DB2 Data Types																					
																		For Bit Data				
OLE DB Type Indicator	**SMALLINT**	**INTEGER**	**BIGINT**	**REAL**	**FLOAT DOUBLE**	**DECIMAL NUMERIC**	**DATE**	**TIME**	**TIMESTAMP**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**CLOB**	**GRAPHIC**	**VARGRAPHIC**	**LONG VARGRAPHIC**	**DBCLOB**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**BLOB**	**DATALINK**
DBTYPE_ARRAY																						
DBTYPE_VECTOR																						
DBTYPE_UDT																						
DBTYPE_DBDATE							X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_DBTIME							X	X	X	X	X	X		X	X	X						X
DBTYPE_DBTIMESTAMP							X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_FILETIME			X				X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_PROP_VARIANT	X	X	X	X	X					X	X	X		X	X	X		X	X	X		X
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

Note: When the application performs the ISequentialStream::Read to get the data from the storage object, the format of the data returned depends on the column data type:

- For non character and binary data types, the data of the column is exposed as a sequence of bytes which represent those values in the operating system.
- For character data type, the data is first converted to DBTYPE_STR.
- For DBCLOB, the data is first converted to DBTYPE_WCHAR.

Related reference:

- "Data conversion for setting data from OLE DB Types to DB2 Types" on page 128

IBM OLE DB Provider restrictions

Following are the restrictions for the IBM OLE DB Provider:

- IBMDADB2 supports auto commit and user-controlled transaction scope with the `ITransactionLocal` interface. Auto commit transaction scope is the default scope. Nested transactions are not supported.
- `RestartPosition` is not supported when the command text contains parameters.
- IBMDADB2 does not quote table names passed through the DBID parameters, which are parameters used by the `IOpenRowset` interface. Instead, the OLE DB consumer must add quotes to the table names when quotes are required.

IBM OLE DB Provider support for OLE DB components and interfaces

The following table lists the OLE DB components and interfaces that are supported by the IBM OLE DB Provider for DB2 and the Microsoft OLE DB Provider for ODBC.

Table 32. Comparison of OLE DB Components and Interfaces Supported by the IBM OLE DB Provider for DB2 and the Microsoft OLE DB Provider for ODBC

	Interface	DB2	ODBC Provider
BLOB			
	ISequentialStream	Yes	Yes
Command			
	IAccessor	Yes	Yes
	ICommand	Yes	Yes
	ICommandPersist	No	No
	ICommandPrepare	Yes	Yes
	ICommandProperties	Yes	Yes
	ICommandText	Yes	Yes
	ICommandWithParameters	Yes	Yes
	IColumnsInfo	Yes	Yes
	IColumnsRowset	Yes	Yes
	IConvertType	Yes	Yes
	ISupportErrorInfo	Yes	Yes
DataSource			
	IConnectionPoint	No	Yes
	IDBAsynchNotify (consumer)	No	No
	IDBAsynchStatus	No	No
	IDBConnectionPointContainer	No	Yes
	IDBCreateSession	Yes	Yes
	IDBDataSourceAdmin	No	No
	IDBInfo	Yes	Yes
	IDBInitialize	Yes	Yes
	IDBProperties	Yes	Yes
	IPersist	Yes	No
	IPersistFile	Yes	Yes
	ISupportErrorInfo	Yes	Yes
Enumerator			
	IDBInitialize	Yes	Yes
	IDBProperties	Yes	Yes
	IParseDisplayName	Yes	No
	ISourcesRowset	Yes	Yes
	ISupportErrorInfo	Yes	Yes
Error Lookup Service			

Table 32. Comparison of OLE DB Components and Interfaces Supported by the IBM OLE DB Provider for DB2 and the Microsoft OLE DB Provider for ODBC (continued)

	Interface	DB2	ODBC Provider
	IErrorLookUp	Yes	Yes
Error Object			
	IErrorInfo	Yes	No
	IErrorRecords	Yes	No
	ISQLErrorInfo (custom)	Yes	No
Multiple Results			
	IMultipleResults	Yes	Yes
	ISupportErrorInfo	Yes	Yes
RowSet			
	IAccessor	Yes	Yes
	IColumnsRowset	Yes	Yes
	IColumnsInfo	Yes	Yes
	IConvertType	Yes	Yes
	IChapteredRowset	No	No
	IConnectionPointContainer	Yes	Yes
	IDBAsynchStatus	No	No
	IParentRowset	No	No
	IRowset	Yes	Yes
	IRowsetChange	Yes	Yes
	IRowsetChapterMember	No	No
	IRowsetFind	No	No
	IRowsetIdentity	Yes	Yes
	IRowsetIndex	No	No
	IRowsetInfo	Yes	Yes
	IRowsetLocate	Yes	Yes
	IRowsetNotify (consumer)	Yes	No
	IRowsetRefresh	Cursor Service Component	Yes
	IRowsetResynch	Cursor Service Component	Yes
	IRowsetScroll	Yes[1]	Yes
	IRowsetUpdate	Cursor Service Component	Yes
	IRowsetView	No	No
	ISupportErrorInfo	Yes	Yes
Notes: 1. The values to be returned are approximations. Deleted rows will not be skipped.			
Session			
	IAlterIndex	No	No
	IAlterTable	No	No
	IDBCreateCommand	Yes	Yes
	IDBSchemaRowset	Yes	Yes

Table 32. Comparison of OLE DB Components and Interfaces Supported by the IBM OLE DB Provider for DB2 and the Microsoft OLE DB Provider for ODBC (continued)

	Interface	DB2	ODBC Provider
	IGetDataSource	Yes	Yes
	IIndexDefinition	No	No
	IOpenRowset	Yes	Yes
	ISessionProperties	Yes	Yes
	ISupportErrorInfo	Yes	Yes
	ITableDefinition	No	No
	ITableDefinitionWithConstraints	No	No
	ITransaction	Yes	Yes
	ITransactionJoin	Yes	Yes
	ITransactionLocal	Yes	Yes
	ITransactionObject	No	No
	ITransactionOptions	No	Yes
View Objects			
	IViewChapter	No	No
	IViewFilter	No	No
	IViewRowset	No	No
	IViewSort	No	No

IBM OLE DB Provider support for OLE DB properties

The following table shows the OLE DB properties that are supported by the IBM OLE DB Provider for DB2:

Table 33. Properties Supported by the IBM OLE DB Provider for DB2

Property Group	Property Set	Properties	Default Value	R/W
Data Source	DBPROPSET_DATASOURCE	DBPROP_MULTIPLECONNECTIONS	VARIANT_FALSE	R
		DBPROP_RESETDATASOURCE	DBPROPVAL_RD_RESETALL	R/W
DB2 Data Source	DBPROPSET_DB2DATASOURCE	DB2PROP_REPORTISLONGFORLONGTYPES	VARIANT_FALSE	R/W
		DB2PROP_RETURNCHARASWCHAR	VARIANT_TRUE	R/W
		DB2PROP_SORTBYORDINAL	VARIANT_FALSE	R/W
Data Source Information	DBPROPSET _DATASOURCEINFO	DBPROP_ACTIVESESSIONS	0	R
		DBPROP_ASYNCTXNABORT	VARIANT_FALSE	R
		DBPROP_ASYNCTXNCOMMIT	VARIANT_FALSE	R
		DBPROP_BYREFACCESSORS	VARIANT_FALSE	R
		DBPROP_COLUMNDEFINITION	DBPROPVAL_CD_NOTNULL	R
		DBPROP_CONCATNULLBEHAVIOR	DBPROPVAL_CB_NULL	R
		DBPROP_CONNECTIONSTATUS	DBPROPVAL_CS_INITIALIZED	R
		DBPROP_DATASOURCENAME	N/A	R
		DBPROP_DATASOURCEREADONLY	VARIANT_FALSE	R
		DBPROP_DBMSNAME	N/A	R
		DBPROP_DBMSVER	N/A	R
		DBPROP_DSOTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R
		DBPROP_GROUPBY	DBPROPVAL_GB_CONTAINS_SELECT	R
		DBPROP_IDENTIFIERCASE	DBPROPVAL_IC_UPPER	R
		DBPROP_MAXINDEXSIZE	0	R

Table 33. Properties Supported by the IBM OLE DB Provider for DB2 (continued)

Property Group	Property Set	Properties	Default Value	R/W
		DBPROP_MAXROWSIZE	0	R
		DBPROP_MAXROWSIZEINCLUDESBLOB	VARIANT_TRUE	R
		DBPROP_MAXTABLEINSELECT	0	R
		DBPROP_MULTIPLEPARAMSETS	VARIANT_FALSE	R
		DBPROP_MULTIPLERESULTS	DBPROPVAL_MR_SUPPORTED	R
		DBPROP_MULTIPLESTORAGEOBJECTS	VARIANT_TRUE	R
		DBPROP_MULTITABLEUPDATE	VARIANT_FALSE	R
		DBPROP_NULLCOLLATION	DBPROPVAL_NC_LOW	R
		DBPROP_OLEOBJECTS	DBPROPVAL_OO_BLOB	R
		DBPROP_ORDERBYCOLUMNSINSELECT	VARIANT_FALSE	R
		DBPROP _OUTPUTPARAMETERAVAILABILITY	DBPROPVAL_OA_ATEXECUTE	R
		DBPROP_PERSISTENTIDTYPE	DBPROPVAL_PT_NAME	R
		DBPROP_PREPAREABORTBEHAVIOR	DBPROPVAL_CB_DELETE	R
		DBPROP_PROCEDURETERM	“STORED PROCEDURE”	R
		DBPROP_PROVIDERFRIENDLYNAME	“IBM OLE DB Provider for DB2”	R
		DBPROP_PROVIDERNAME	“IBMDADB2.DLL”	R
		DBPROP_PROVIDEROLEDBVER	“02.7”	R
		DBPROP_PROVIDERVER	N/A	R
		DBPROP_QUOTEIDENTIFIERCASE	DBPROPVAL_IC_SENSITIVE	R
		DBPROP _ROWSETCONVERSIONSONCOMMAND	VARIANT_TRUE	R
		DBPROP_SCHEMATERM	“SCHEMA”	R
		DBPROP_SCHEMAUSAGE	DBPROPVAL_SU_DML_STATEMENTS | DBPROPVAL_SU_TABLE_DEFINITION | DBPROPVAL_SU_INDEX_DEFINITION | DBPROPVAL_SU_PRIVILEGE_DEFINITION	R
		DBPROP_SQLSUPPORT	DBPROPVAL_SQL_ODBC_EXTENDED | DBPROPVAL_SQL_ESCAPECLAUSES | DBPROPVAL_SQL_ANSI92_ENTRY	R
		DBPROP_SERVERNAME	N/A	R
		DBPROP_STRUCTUREDSTORAGE	DBPROPVAL_SS_ISEQUENTIALSTREAM	R
		DBPROP_SUBQUERIES	DBPROPVAL_SQ_CORRELATEDSUBQUERIES | DBPROPVAL_SQ_COMPARISON | DBPROPVAL_SQ_EXISTS | DBPROPVAL_SQ_IN | DBPROPVAL_SQ_QUANTIFIED |	R
		DBPROP_SUPPORTEDTXNDDL	DBPROPVAL_TC_ALL	R
		DBPROP_SUPPORTEDTXNISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY | DBPROPVAL_TI_READCOMMITTED | DBPROPVAL_TI_READUNCOMMITTED | DBPROPVAL_TI_SERIALIZABLE |	R
		DBPROP_SUPPORTEDTXNISORETAIN	DBPROPVAL_TR_COMMIT_DC | DBPROPVAL_TR_ABORT_NO |	R
		DBPROP_TABLETERM	“TABLE”	R
		DBPROP_USERNAME	N/A	R
Initialization	DBPROPSET_DBINIT	DBPROP_AUTH_PASSWORD	N/A	R/W
		DBPROP_INIT_TIMEOUT (1)	0	R/W
		DBPROP_AUTH_PERSIST _SENSITIVE_AUTHINFO	VARIANT_FALSE	R/W
		DBPROP_AUTH_USERID	N/A	R/W
		DBPROP_INIT_DATASOURCE	N/A	R/W
		DBPROP_INIT_HWND	N/A	R/W
		DBPROP_INIT_MODE	DB_MODE_READWRITE	R/W
		DBPROP_INIT_OLEDBSERVICES	0xFFFFFFFF	R/W
		DBPROP_INIT_PROMPT	DBPROMPT_NOPROMPT	R/W
		DBPROP_INIT_PROVIDERSTRING	N/A	R/W
Rowset	DBPROPSET_ROWSET	DBPROP_ABORTPRESERVE	VARIANT_FALSE	R
		DBPROP_ACCESSORDER	DBPROPVAL_AO_RANDOM	R
		DBPROP_BLOCKINGSTORAGEOBJECTS	VARIANT_FALSE	R
		DBPROP_BOOKMARKS	VARIANT_FALSE	R/W
		DBPROP_BOOKMARKSKIPPED	VARIANT_FALSE	R
		DBPROP_BOOKMARKTYPE	DBPROPVAL_BMK_NUMERIC	R

Table 33. Properties Supported by the IBM OLE DB Provider for DB2 (continued)

Property Group	Property Set	Properties	Default Value	R/W
		DBPROP_CACHEDEFERRED	VARIANT_FALSE	R/W
		DBPROP_CANFETCHBACKWARDS	VARIANT_FALSE	R/W
		DBPROP_CANHOLDROWS	VARIANT_FALSE	R
		DBPROP_CANSCROLLBACKWARDS	VARIANT_FALSE	R/W
		DBPROP_CHANGEINSERTEDROWS	VARIANT_FALSE	R
		DBPROP_COMMITPRESERVE	VARIANT_TRUE	R/W
		DBPROP_COMMANDTIMEOUT	0	R/W
		DBPROP_DEFERRED	VARIANT_FALSE	R
		DBPROP_IAccessor	VARIANT_TRUE	R
		DBPROP_IColumnsInfo	VARIANT_TRUE	R
		DBPROP_IColumnsRowset	VARIANT_TRUE	R/W
		DBPROP_IConvertType	VARIANT_TRUE	R
		DBPROP_IMultipleResults	VARIANT_FALSE	R/W
		DBPROP_IRowset	VARIANT_TRUE	R
		DBPROP_IRowChange	VARIANT_FALSE	R/W
		DBPROP_IRowsetFind	VARIANT_FALSE	R
		DBPROP_IRowsetIdentity	VARIANT_TRUE	R
		DBPROP_IRowsetInfo	VARIANT_TRUE	R
		DBPROP_IRowsetLocate	VARIANT_FALSE	R/W
		DBPROP_IRowsetScroll	VARIANT_FALSE	R/W
		DBPROP_IRowsetUpdate	VARIANT_FALSE	R
		DBPROP_ISequentialStream	VARIANT_TRUE	R
		DBPROP_ISupportErrorInfo	VARIANT_TRUE	R
		DBPROP_LITERALBOOKMARKS	VARIANT_FALSE	R
		DBPROP_LITERALIDENTITY	VARIANT_TRUE	R
		DBPROP_LOCKMODE	DBPROPVAL_LM_SINGLEROW	R/W
		DBPROP_MAXOPENROWS	32767	R
		DBPROP_MAXROWS	0	R/W
		DBPROP_NOTIFICATIONGRANULARITY	DBPROPVAL_NT_SINGLEROW	R/W
		DBPROP_NOTIFICATION PHASES	DBPROPVAL_NP_OKTODO DBPROPBAL_NP_ABOUTTODO DBPROPVAL_NP_SYNCHAFTER DBPROPVAL_NP_FAILEDTODO DBPROPVAL_NP_DIDEVENT	R
		DBPROP_NOTIFYROWSETRELEASE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP _NOTIFYROWSETFETCHPOSITIONCHANGE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP_NOTIFYCOLUMNSET	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP_NOTIFYROWDELETE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP_NOTIFYROWINSERT	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP_ORDEREDBOOKMARKS	VARIANT_FALSE	R
		DBPROP_OTHERINSERT	VARIANT_FALSE	R
		DBPROP_OTHERUPDATEDELETE	VARIANT_FALSE	R/W
		DBPROP_OWNINSERT	VARIANT_FALSE	R
		DBPROP_OWNUPDATEDELETE	VARIANT_FALSE	R
		DBPROP_QUICKRESTART	VARIANT_FALSE	R/W
		DBPROP_REMOVEDELETED	VARIANT_FALSE	R/W
		DBPROP_ROWTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R
		DBPROP_SERVERCURSOR	VARIANT_TRUE	R
		DBPROP_SERVERDATAONINSERT	VARIANT_FALSE	R
		DBPROP_UNIQUEROWS	VARIANT_FALSE	R/W
		DBPROP_UPDATABILITY	0	R/W
Rowset	DBPROPSET_DB2ROWSET	DBPROP_ISLONGMINLENGTH	32000	R/W
Session	DBPROPSET_SESSION	DBPROP_SESS_AUTOCOMMITISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY	R/W

Notes:

1. The timeout is applicable only when using the TCP/IP protocol to connect to the server. The timeout is enforced only during the TCP/IP sock connect. If the sock connect completes before the specified timeout expires, the timeout will no longer be enforced for the rest of the initialization process. If the client-reroute feature is used then the timeout will be doubled. In general, when client re-route is enabled, the connection timeout behavior is dictated by client re-route.

Connections to data sources using the IBM OLE DB Provider

The following examples show how to connect to a DB2 data source using the IBM OLE DB Provider for DB2:

Example 1: Visual Basic application using ADO:

```
Dim db As ADODB.Connection
Set db = New ADODB.Connection
db.Provider = "IBMDADB2"
db.CursorLocation = adUseClient
...
```

Example 2: C/C++ application using IDBPromptInitialize and Data Links:

```
// Create DataLinks
hr = CoCreateInstance (
     CLSID_DataLinks,
     NULL,
     CLSCTX_INPROC_SERVER,
     IID_IDBPromptInitialize,
     (void**)&pIDBPromptInitialize);

// Invoke the DataLinks UI to select the provider and data source
hr = pIDBPromptInitialize->PromptDataSource (
     NULL,
     GetDesktopWindow(),
     DBPROMPTOPTIONS_PROPERTYSHEET,
     0,
     NULL,
     NULL,
     IID_IDBInitialize,
     (IUnknown**)&pIDBInitialize);
```

Example 3: C/C++ application using IDataInitialize and Service Component:

```
hr = CoCreateInstance (
     CLSID_MSDAINITIALIZE,
     NULL,
     CLSCTX_INPROC_SERVER,
     IID_IDataInitialize,
     (void**)&pIDataInitialize);

hr = pIDataInitialize->CreateDBInstance(
     CLSID_IBMDADB2, // ClassID of IBMDADB2
     NULL,
     CLSCTX_INPROC_SERVER,
     NULL,
     IID_IDBInitialize,
     (IUnknown**)&pIDBInitialize);
```

ADO Applications

The sections that follow describe considerations for ADO applications.

ADO connection string keywords

To specify ADO (ActiveX Data Objects) connection string keywords, specify the keyword using the `keyword=value` format in the provider (connection) string. Delimit multiple keywords with a semicolon (;).

The following table describes the keywords supported by the IBM OLE DB Provider for DB2:

Table 34. Keywords supported by the IBM OLE DB Provider for DB2

Keyword	**Value**	**Meaning**
DSN	Name of the database alias	The DB2 database alias in the database directory.
UID	User ID	The user ID used to connect to the DB2 server.
PWD	Password of UID	Password for the user ID used to connect to the DB2 server.

Other DB2 CLI configuration keywords also affect the behavior of the IBM OLE DB Provider.

Related reference:

- "CLI/ODBC configuration keywords listing by category" in *Call Level Interface Guide and Reference, Volume 1*

Connections to data sources with Visual Basic ADO applications

To connect to a DB2 data source using the IBM OLE DB Provider for DB2, specify the IBMDADB2 provider name.

Related concepts:

- "Connections to data sources using the IBM OLE DB Provider" on page 137

Related tasks:

- "Building ADO applications with Visual Basic" on page 144

Updatable scrollable cursors in ADO applications

The IBM OLE DB Provider for DB2 natively supports read-only, forward-only, read-only scrollable, and updatable scrollable cursors. An ADO application that wants to access updatable scrollable cursors can set the cursor location to either `adUseClient` or `adUseServer`. Setting the cursor location to `adUseServer` causes the cursor to materialize on the server.

Limitations for ADO applications

Following are the limitations for ADO applications:

- ADO applications calling stored procedures must have their parameters created and explicitly bound. The `Parameters.Refresh` method for automatically generating parameters is not supported for DB2 Server for VSE & VM.

- There is no support for default parameter values.
- When inserting a new row using a server-side scrollable cursor, use the AddNew() method with the Fieldlist and Values arguments. This is more efficient than calling AddNew() with no arguments following Update() calls for each column. Each AddNew() and Update() call is a separate request to the server and therefore, is less efficient than a single call to AddNew().
- Newly inserted rows are not updatable with a server-side scrollable cursor.
- Tables with long data, LOB, or Datalink columns are not updatable when using a server-side scrollable cursor.

IBM OLE DB Provider support for ADO methods and properties

The IBM OLE DB Provider supports the following ADO methods and properties:

Table 35. ADO methods and properties supported by the IBM OLE DB Provider for DB2

ADO	Method/Property	OLE DB Interface/Property	IBM OLE DB Support
Command Methods	Cancel	ICommand	Yes
	CreateParameter		Yes
	Execute		Yes
Command Properties	ActiveConnection	(ADO specific)	
	Command Text	ICommandText	Yes
	Command Timeout	ICommandProperties::SetProperties DBPROP_COMMANDTIMEOUT	Yes
	CommandType	(ADO specific)	
	Prepared	ICommandPrepare	Yes
	State	(ADO specific)	
Command Collection	Parameters	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	Yes
	Properties	ICommandProperties IDBProperties	Yes
Connection Methods	BeginTrans CommitTrans RollbackTrans	ITransactionLocal	Yes (but not nested) Yes (but not nested) Yes (but not nested)
	Execute	ICommand IOpenRowset	Yes
	Open	IDBCreateSession IDBInitialize	Yes
	OpenSchema adSchemaColumnPrivileges adSchemaColumns adSchemaForeignKeys adSchemaIndexes adSchemaPrimaryKeys adSchemaProcedureParam adSchemaProcedures adSchemaProviderType adSchemaStatistics adSchemaTablePrivileges adSchemaTables	IDBSchemaRowset	 Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes
	Cancel		Yes

Table 35. ADO methods and properties supported by the IBM OLE DB Provider for DB2 (continued)

ADO	Method/Property	OLE DB Interface/Property	IBM OLE DB Support
Connection Properties	Attributes adXactCommitRetaining adXactRollbackRetaining	ITransactionLocal	 Yes Yes
	CommandTimeout	ICommandProperties DBPROP_COMMAND_TIMEOUT	Yes
	ConnectionString	(ADO specific)	
	ConnectionTimeout	IDBProperties DBPROP_INIT_TIMEOUT	No
	CursorLocation: adUseClient adUseNone adUseServer	 (Use OLE DB Cursor Service) (Not Used)	 Yes No Yes
	DefaultDataBase	IDBProperties DBPROP_CURRENTCATALOG	No
	IsolationLevel	ITransactionLocal DBPROP_SESS _AUTOCOMMITISOLEVELS	Yes
	Mode adModeRead adModeReadWrite adModeShareDenyNone adModeShareDenyRead adModeShareDenyWrite adModeShareExclusive adModeUnknown adModeWrite	IDBProperties DBPROP_INIT_MODE	 No Yes No No No No No No
	Provider	ISourceRowset::GetSourceRowset	Yes
	State	(ADO specific)	
	Version	(ADO specific)	
Connection Collection	Errors	IErrorRecords	Yes
	Properties	IDBProperties	Yes
Error Properties	Description NativeError Number Source SQLState	IErrorRecords	Yes Yes Yes Yes Yes
	HelpContext HelpFile		No No
Field Methods	AppendChunk GetChunk	ISequentialStream	Yes Yes
Field Properties	Actual Size	IAccessor IRowset	Yes
	Attributes DataFormat DefinedSize Name NumericScale Precision Type	IColumnInfo	 Yes Yes Yes Yes Yes Yes
	OriginalValue	IRowsetUpdate	Yes (Cursor Service)
	UnderlyingValue	IRowsetRefresh IRowsetResynch	Yes (Cursor Service) Yes (Cursor Service)

Table 35. ADO methods and properties supported by the IBM OLE DB Provider for DB2 (continued)

ADO	Method/Property	OLE DB Interface/Property	IBM OLE DB Support
	Value	IAccessor IRowset	Yes
Field Collection	Properties	IDBProperties IRowsetInfo	Yes
Parameter Methods	AppendChunk	ISequentialStream	Yes
	Attributes Direction Name NumericScale Precision Scale Size Type	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	 Yes No Yes Yes Yes Yes Yes
	Value	IAccessor ICommand	Yes
Parameter Collection	Properties		Yes
RecordSet Methods	AddNew	IRowsetChange	Yes
	Cancel		Yes
	CancelBatch	IRowsetUpdate::Undo	Yes (Cursor Service)
	CancelUpdate		Yes (Cursor Service)
	Clone	IRowsetLocate	Yes
	Close	IAccessor IRowset	Yes
	CompareBookmarks		No
	Delete	IRowsetChange	Yes
	GetRows	IAccessor IRowset	Yes
	Move	IRowset IRowsetLocate	Yes
	MoveFirst	IRowset IRowsetLocate	Yes
	MoveNext	IRowset IRowsetLocate	Yes
	MoveLast	IRowsetLocate	Yes
	MovePrevious	IRowsetLocate	Yes
	NextRecordSet	IMultipleResults	Yes
	Open	ICommand IOpenRowset	Yes
	Requery	ICommand IOpenRowset	Yes
	Resync	IRowsetRefresh	Yes (Cursor Service)
	Supports	IRowsetInfo	Yes
	Update UpdateBatch	IRowsetChange IRowsetUpdate	Yes Yes (Cursor Service)
RecordSet Properties	AbsolutePage	IRowsetLocate IRowsetScroll	Yes Yes[1]
	AbsolutePosition	IRowsetLocate IRowsetScroll	Yes Yes[1]

Table 35. ADO methods and properties supported by the IBM OLE DB Provider for DB2 (continued)

ADO	Method/Property	OLE DB Interface/Property	IBM OLE DB Support
	ActiveConnection	IDBCreateSession IDBInitialize	Yes
	BOF	(ADO specific)	
	Bookmark	IAccessor IRowsetLocate	Yes
	CacheSize	cRows in IRowsetLocate IRowset	Yes
	CursorType adOpenDynamic adOpenForwardOnly adOpenKeySet adOpenStatic	ICommandProperties	 No Yes Yes Yes
	EditMode	IRowsetUpdate	Yes (Cursor Service)
	EOF	(ADO specific)	
	Filter	IRowsetLocate IRowsetView IRowsetUpdate IViewChapter IViewFilter	No
	LockType	ICommandProperties	Yes
	MarshallOption		No
	MaxRecords	ICommandProperties IOpenRowset	Yes
	PageCount	IRowsetScroll	Yes[1]
	PageSize	(ADO specific)	
	Sort	(ADO specific)	
	Source	(ADO specific)	
	State	(ADO specific)	
	Status	IRowsetUpdate	Yes (Cursor Service)
Notes: 1. The values to be returned are approximations. Deleted rows will not be skipped.			
RecordSet Collection	Fields	IColumnInfo	Yes
	Properties	IDBProperties IRowsetInfo::GetProperties	Yes

Object Linking and Embedding Database (OLE DB) table functions

DB2 supports OLE DB table functions. For these functions, there is no application building needed besides creating the `CREATE FUNCTION` DDL. OLE DB table function sample files are provided by DB2 in the `sqllib\samples\oledb` directory. These are Command Line Processor (CLP) files. They can be built with the following steps:

1. db2 connect to database_name
2. db2 -t -v -f *file_name*.db2
3. db2 terminate

where `database_name` is the database you are connecting to, and *file_name* is the name of the CLP file, with extension `.db2`.

These commands must be done in a DB2 Command Window.

Related concepts:

- "OLE DB user-defined table functions" in *Developing SQL and External Routines*

Related reference:

- "Object Linking and Embedding Database (OLE DB) table function samples" in *Samples Topics*

Object Linking and Embedding (OLE) automation with Visual Basic

You can implement OLE automation UDFs and stored procedures in any language, as OLE is language independent. You do this by exposing methods of OLE automation servers, and registering the methods as UDFs with DB2. Application development environments which support the development of OLE automation servers include certain versions of the following: Microsoft Visual Basic, Microsoft Visual C++, Microsoft Visual J++, Microsoft FoxPro, Borland Delphi, Powersoft PowerBuilder, and Micro Focus COBOL. Also, Java beans objects that are wrapped properly for OLE, for example with Microsoft Visual J++, can be accessed via OLE automation.

You need to refer to the documentation of the appropriate application development environment for further information on developing OLE automation servers.

OLE automation UDFs and stored procedures

Microsoft Visual Basic supports the creation of OLE automation servers. A new kind of object is created in Visual Basic by adding a class module to the Visual Basic project. Methods are created by adding public sub-procedures to the class module. These public procedures can be registered to DB2 as OLE automation UDFs and stored procedures. For further information on creating and building OLE servers, refer to the Microsoft Visual Basic manual, *Creating OLE Servers, Microsoft Corporation, 1995*, and to the OLE samples provided by Microsoft Visual Basic.

DB2 provides self-containing samples of OLE automation UDFs and stored procedures in Microsoft Visual Basic, located in the directory `sqllib\samples\ole\msvb`. For information on building and running the OLE automation UDF and stored procedure samples, please see the `README` file in `sqllib\samples\ole`.

Related concepts:

- "OLE automation routine design" in *Developing SQL and External Routines*
- "OLE automation routines in BASIC and C++" in *Developing SQL and External Routines*

Related reference:

- "Object Linking and Embedding (OLE) samples" in *Samples Topics*

Object Linking and Embedding (OLE) automation with Visual C++

You can implement OLE automation UDFs and stored procedures in any language, as OLE is language independent. You do this by exposing methods of OLE automation servers, and registering the methods as UDFs with DB2. Application development environments which support the development of OLE automation servers include certain versions of the following: Microsoft Visual Basic, Microsoft Visual C++, Microsoft Visual J++, Microsoft FoxPro, Borland Delphi, Powersoft PowerBuilder, and Micro Focus COBOL. Also, Java beans objects that are wrapped properly for OLE, for example with Microsoft Visual J++, can be accessed via OLE automation.

You need to refer to the documentation of the appropriate application development environment for further information on developing OLE automation servers.

OLE automation UDFs and stored procedures

Microsoft Visual C++ supports the creation of OLE automation servers. Servers can be implemented using Microsoft Foundation Classes and the Microsoft Foundation Class application wizard, or implemented as Win32 applications. Servers can be DLLs or EXEs. Refer to the Microsoft Visual C++ documentation and to the OLE samples provided by Microsoft Visual C++ for further information.

DB2 provides self-containing samples of OLE automation UDFs and stored procedures in Microsoft Visual C++, located in the directory `sqllib\samples\ole\msvc`. For information on building and running the OLE automation UDF and stored procedure samples, please see the `README` file in `sqllib\samples\ole`.

Related concepts:

- "OLE automation routine design" in *Developing SQL and External Routines*
- "OLE automation routines in BASIC and C++" in *Developing SQL and External Routines*

Related reference:

- "Object Linking and Embedding (OLE) samples" in *Samples Topics*

Building ADO applications with Visual Basic

ActiveX Data Objects (ADO) allow you to write an application to access and manipulate data in a database server through an OLE DB provider. The primary benefits of ADO are high speed, ease of use, low memory overhead, and a small disk footprint.

Visual Basic ADO sample programs are located in the `sqllib\samples\VB\ADO` directory.

Note: To run the DB2 ADO samples, these versions or later of the following components are recommended:

1. Visual Basic 6.0 Professional Edition
2. Microsoft Data Access 2.7 SDK (optionally installed with DB2 Version 8)
3. Visual Basic Service pack 5 from http://msdn.microsoft.com/vstudio/sp/vs6sp5/vbfixes.asp.

4. The latest Visual Studio Service Pack from http://msdn.microsoft.com/vstudio/.

Procedure:

You can use either of two ODBC-compliant providers:
- IBM OLE DB provider for DB2
- Microsoft OLE DB provider for ODBC

Using the IBM OLE DB provider for DB2

DB2 Version 8.2 Clients on Windows operating systems will optionally install IBMDADB2, the IBM OLE DB 2.0-compliant provider for DB2. The provider exposes interfaces for consumers who want to access data in a DB2 database. The IBM OLE DB provider for DB2 supports the following ADO application types:
- Microsoft Active Server Pages (ASP)
- Microsoft Visual Studio C++ and Visual Basic applications
- Microsoft Visual Interdev

For details on these types of applications, refer to the ADO documentation.

To access a DB2 server using the IBM OLE DB provider for DB2, the Visual Basic application should specify the PROVIDER keyword in the ADO connection string as follows:

```
Dim c1 As ADODB.Connection
Dim c1str As String
c1str = "Provider=ibmdadb2; DSN=db2alias; UID=userid; PWD=password"
c1.Open c1str
...
```

where `db2alias` is the alias for the DB2 database which is cataloged in the DB2 database directory.

Note: When using the IBM OLE DB provider for DB2, you do not need to perform the ODBC catalog step for the datasource. This step is required when you are using the OLE DB provider for ODBC.

Using the Microsoft OLE DB provider for ODBC

To use ADO with the Microsoft OLE DB provider and Visual Basic, you need to establish a reference to the ADO type library. Do the following:
1. Select ″References″ from the Project menu
2. Check the box for ″Microsoft ActiveX Data Objects <version_number> Library″
3. Click ″OK″.

where <version_number> is the current version the ADO library.

Once this is done, ADO objects, methods, and properties will be accessible through the VBA Object Browser and the IDE Editor.

Establish a connection:

```
Dim db As Connection
Set db = New Connection
```

Set client-side cursors supplied by the local cursor library:

```
db.CursorLocation = adUseClient
```

and set the provider so ADO will use the Microsoft ODBC Driver.

Accessing the sample database with ADO

A full Visual Basic program includes forms and other graphical elements, and you need to view it inside the Visual Basic environment. Here are Visual Basic commands as part of a program to access the DB2 `sample` database, after you have connected to the database with either the IBM OLE DB provider or the Microsoft OLE DB provider, as discussed above.

Open the `sample` database without specifying a user ID or password; that is, use the current user:

```
db.Open "SAMPLE"
```

Create a record set:

```
Set adoPrimaryRS = New Recordset
```

Use a select statement to fill the record set:

```
adoPrimaryRS.Open "select EMPNO,LASTNAME,FIRSTNME,MIDINIT,EDLEVEL,JOB
from EMPLOYEE Order by EMPNO", db
```

From this point, the programmer can use the ADO methods to access the data such as moving to the next record set:

```
adoPrimaryRS.MoveNext
```

Deleting the current record in the record set:

```
adoPrimaryRS.Delete
```

As well, the programmer can do the following to access an individual field:

```
Dim Text1 as String
Text1 = adoPrimaryRS!LASTNAME
```

Related concepts:

- "Application Types Supported by the IBM OLE DB Provider for DB2" on page 124
- "Connections to data sources with Visual Basic ADO applications" on page 138
- "COM+ distributed transaction support and the IBM OLE DB Provider" on page 150
- "IBM OLE DB Provider restrictions" on page 131
- "Large object manipulation with the IBM OLE DB Provider" on page 125
- "OLE DB services automatically enabled by the IBM OLE DB Provider" on page 126
- "IBM OLE DB Provider for DB2" on page 123

Related reference:

- "IBM OLE DB Provider support for ADO methods and properties" on page 139
- "IBM OLE DB Provider support for OLE DB components and interfaces" on page 132
- "IBM OLE DB Provider support for OLE DB properties" on page 134
- "Visual Basic samples" in *Samples Topics*

Building RDO applications with Visual Basic

Remote Data Objects (RDO) provide an information model for accessing remote data sources through ODBC. RDO offers a set of objects that make it easy to connect to a database, execute queries and stored procedures, manipulate results, and commit changes to the server. It is specifically designed to access remote ODBC relational data sources, and makes it easier to use ODBC without complex application code, and is a primary means of accessing a relational database that is exposed with an ODBC driver. RDO implements a thin code layer over the Open Database Connectivity (ODBC) API and driver manager that establishes connections, creates result sets and cursors, and executes complex procedures using minimal workstation resources.

DB2 provides Visual Basic RDO sample programs in the `sqllib\samples\VB` directory.

Procedure:

To use RDO with Microsoft Visual Basic, you need to establish a reference to your Visual Basic project. Do the following:

1. Select ″References″ from the Project menu
2. Check the box for ″Microsoft Remote Data Object <Version Number>″
3. Click ″OK″.

where <version_number> is the current RDO version.

A full Visual Basic program includes forms and other graphical elements, and you need to view it inside the Visual Basic environment. Here are Visual Basic commands as part of a DB2 program that connects to the `sample` database, opens a record set that selects all the columns from the `EMPLOYEE` table, and then displays the employee names in a message window, one by one:

```
Dim rdoEn As rdoEngine
Dim rdoEv As rdoEnvironment
Dim rdoCn As rdoConnection
Dim Cnct$
Dim rdoRS As rdoResultset
Dim SQLQueryDB As String
```

Assign the connection string:

```
 Cnct$ = "DSN=SAMPLE;UID=;PWD=;"
```

Set the RDO environment:

```
Set rdoEn = rdoEngine
Set rdoEv = rdoEn.rdoEnvironments(0)
```

Connect to the database:

```
 Set rdoCn = rdoEv.OpenConnection("", , , Cnct$)
```

Assign the SELECT statement for the record set:

```
SQLQueryDB = "SELECT * FROM EMPLOYEE"
```

Open the record set and execute the query:

```
Set rdoRS = rdoCn.OpenResultset(SQLQueryDB)
```

While not at the end of the record set, display Message Box with LASTNAME, FIRSTNME from table, one employee at a time:

```
While Not rdoRS.EOF
MsgBox rdoRS!LASTNAME & ", " & rdoRS!FIRSTNME
```

Move to the next row in the record set:

```
rdoRS.MoveNext
Wend
```

Close the program:

```
rdoRS.Close
rdoCn.Close
rdoEv.Close
```

Related reference:

- "Visual Basic samples" in *Samples Topics*

Building ADO applications with Visual C++

ActiveX Data Objects (ADO) allow you to write an application to access and manipulate data in a database server through an OLE DB provider. The primary benefits of ADO are high speed, ease of use, low memory overhead, and a small disk footprint.

DB2 provides Visual C++ ADO sample programs in the `sqllib\samples\VC` directory.

Procedure:

You can use either of two ODBC-compliant providers:

- IBM OLE DB provider for DB2
- Microsoft OLE DB provider for ODBC

Using the IBM OLE DB provider for DB2

DB2 Version 8.2 Clients on Windows operating systems will optionally install IBMDADB2, the IBM OLE DB 2.0-compliant provider for DB2. The provider exposes interfaces for consumers who want to access data in a DB2 database. The IBM OLE DB provider for DB2 supports the following ADO application types:

- Microsoft Active Server Pages (ASP)
- Microsoft Visual Studio C++ and Visual Basic applications
- Microsoft Visual Interdev

For details on these types of applications, refer to the ADO documentation.

Using the Microsoft OLE DB provider for ODBC

DB2 ADO programs using the Microsoft OLE DB provider and Visual C++ can be compiled the same as regular C++ programs, once you make the following change.

To have your C++ source program run as an ADO program, you can put the following import statement at the top of your source program file:

```
#import "C:\program files\common files\system\ado\msado<VERSION NUMBER>.dll" \
 no_namespace \
 rename( "EOF", "adoEOF")
```

where <VERSION NUMBER> is the version number of the ADO library.

When the program is compiled, the user will need to verify that the msado<VERSION NUMBER>.dll is in the path specified. An alternative is to add C:\program files\common files\system\ado to the environment variable LIBPATH, and then use this shorter import statement in your source file:

```
#import <msado<VERSION NUMBER>.dll> \
 no_namespace \
 rename( "EOF", "adoEOF")
```

This is the method used in the DB2 sample program, BLOBAccess.dsp.

With this IMPORT statement, your DB2 program will have access to the ADO library. You can now compile your Visual C++ program as you would any other program. If you are also using another programming interface, such as DB2 APIs, or DB2 CLI, refer to the appropriate topic for additional information on building your program.

Related concepts:

- "Application Types Supported by the IBM OLE DB Provider for DB2" on page 124
- "Compilation and linking of C/C++ applications and the IBM OLE DB Provider" on page 149
- "Connections to data sources in C/C++ applications using the IBM OLE DB Provider" on page 150
- "IBM OLE DB Provider restrictions" on page 131
- "Large object manipulation with the IBM OLE DB Provider" on page 125
- "OLE DB services automatically enabled by the IBM OLE DB Provider" on page 126
- "IBM OLE DB Provider for DB2" on page 123

Related reference:

- "Data conversion for setting data from DB2 types to OLE DB types" on page 130
- "Data conversion for setting data from OLE DB Types to DB2 Types" on page 128
- "Data type mappings between DB2 and OLE DB" on page 127
- "IBM OLE DB Provider support for ADO methods and properties" on page 139
- "IBM OLE DB Provider support for OLE DB components and interfaces" on page 132
- "IBM OLE DB Provider support for OLE DB properties" on page 134
- "Visual C++ samples" in *Samples Topics*

C and C++ Applications

The sections that follow describe considerations for C and C++ applications.

Compilation and linking of C/C++ applications and the IBM OLE DB Provider

C/C++ applications that use the constant CLSID_IBMDADB2 must include the ibmdadb2.h file, which can be found in the SQLLIB\include directory. These applications must define DBINITCONSTANTS before the include statement. The following example shows the correct sequence of C/C++ preprocessor directives:

```
#define DBINITCONSTANTS
#include "ibmdadb2.h"
```

Connections to data sources in C/C++ applications using the IBM OLE DB Provider

To connect to a DB2 data source using the IBM OLE DB Provider for DB2 in a C/C++ application, you can use one of the two OLE DB core interfaces, `IDBPromptInitialize` or `IDataInitialize`, or you can call the COM API CoCreateInstance. The `IDataInitialize` interface is exposed by the OLE DB Service Component, and the `IDBPromptInitialize` is exposed by the Data Links Component.

Related concepts:

- "Connections to data sources using the IBM OLE DB Provider" on page 137

Related tasks:

- "Building ADO applications with Visual C++" on page 148

MTS and COM+ Distributed Transactions

The sections that follow describe considerations for MTS and COM+ distributed transactions.

COM+ distributed transaction support and the IBM OLE DB Provider

OLE DB applications running in a Microsoft Component Services (COM+) environment on Windows 2000 or XP can use the `ITransactionJoin` interface to participate in distributed transactions with multiple DB2 Database for Linux, UNIX, and Windows, host, and iSeries database servers as well as other resource managers that comply with the COM+ specifications.

Prerequisites:

To use the COM+ distributed transaction support offered by the IBM OLE DB Provider for DB2, ensure that your server meets the following prerequisites.

Note: These requirements are only for the Windows-based computers where DB2 clients are installed.

- Windows 2000 with Service Pack 3 or later
- Windows XP

Related concepts:

- "Loosely coupled support with Microsoft Component Services (COM+)" on page 153
- "Microsoft Component Services (COM+) as transaction manager" on page 151

Enablement of COM+ support in C/C++ database applications

To run a C or C++ application in COM+ transactional mode, you can create the IBMDADB2 data source instance using the `DataLink` interface. You could also use `CoCreateInstance`, get a session object, and use `JoinTransaction`. See the description of how to connect a C or C++ application to a data source for more information.

To run an ADO application in COM+ transactional mode, see the description of how to connect a C or C++ application to a data source.

To use a component in an COM+ package in transactional mode, set the Transactions property of the component to one of the following values:

- "Required"
- "Required New"
- "Supported"

For information about these values, see the COM+ documentation.

Related concepts:

- "Loosely coupled support with Microsoft Component Services (COM+)" on page 153
- "Microsoft Component Services (COM+) as transaction manager" on page 151

Microsoft Component Services (COM+) as transaction manager

Microsoft Component Services (COM+) as transaction manager

DB2 database systems can be fully integrated with Microsoft Component Services (COM+) on Windows 2000 and Windows XP to coordinate two-phase commit with multiple DB2 Database for Linux, UNIX, and Windows, zSeries®, and iSeries database servers, as well as with other resource managers that comply with COM+ specifications.

Prerequisites:

To use COM+ distributed transaction support, ensure that the following requirements are met for the Windows machine where the DB2 client is installed:

- Windows 2000: Service Pack 3 or later

For DB2 CLI applications using COM+:

- Do not change the default value of the SQL_ATTR_CONNECTION_POOLING CLI environment attribute (default SQL_CP_OFF)
- The installation of the DB2 ODBC driver on Windows operating systems will automatically add a new keyword to the registry:

```
HKEY_LOCAL_MACHINE\software\ODBC\odbcinit.ini\IBM DB2 ODBC DRIVER - <DB2 Copy Name>:
  Keyword Value Name: CPTimeout
  Data Type: REG_SZ
  Value: 60
```

Supported DB2 database servers:

The following servers are supported for multisite update using COM+ coordinated transactions:

- DB2 Enterprise Server Edition (ESE)

  Note: Loosely coupled global transactions for COM+ are not supported in massively parallel processing (MPP) environments. Loosely coupled global transactions exist when each of a number of application processes accesses resource managers as if it was in a separate global transaction, however, those application processes are under the coordination of the transaction manager. Each application process will have its own transaction branch within a resource manager. When a commit or rollback is requested by any one of the application processes, transaction manager, or resource manager, the transaction branches are completed altogether. It is the application's responsibility to ensure that resource deadlock does not occur among the branches.

  (Tightly coupled global transactions exist when multiple application processes take turns to do work under the same transaction branch in a resource manager. To the resource manager, the two application processes are a single entity. The resource manager must ensure that resource deadlock does not occur within the transaction branch.)
- DB2 Universal Database for z/OS
- DB2 Universal Database for iSeries
- DB2 Server for VSE & VM

Related concepts:

- "Loosely coupled support with Microsoft Component Services (COM+)" on page 153
- "Microsoft Component Services (COM+) transaction timeout" on page 153
- "COM+ distributed transaction support and the IBM OLE DB Provider" on page 150
- "X/Open distributed transaction processing model" in *Administration Guide: Planning*
- "ODBC and ADO connection pooling with Microsoft Component Services (COM+)" on page 154

Related tasks:

- "Installing a DB2 Connect server product (Windows)" in *Quick Beginnings for DB2 Connect Servers*

Related reference:

- "Connection attributes (CLI) list" in *Call Level Interface Guide and Reference, Volume 2*
- "Environment attributes (CLI) list" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLSetConnectAttr function (CLI) - Set connection attributes" in *Call Level Interface Guide and Reference, Volume 2*
- "DB2 Connect product offerings" in *DB2 Connect User's Guide*

Loosely coupled support with Microsoft Component Services (COM+)

Loosely coupled global transactions exist when each of a number of application processes accesses resource managers as if it was in a separate global transaction, however, those application processes are under the coordination of the transaction manager. Each application process will have its own transaction branch within a resource manager. When a commit or rollback is requested by any one of the application processes, transaction manager, or resource manager, the transaction branches are completed altogether. It is the application's responsibility to ensure that resource deadlock does not occur among the branches.

DB2 Version 9 supports loosely coupled global transactions for COM+ objects, with no lock timeout or deadlock, given the following restrictions:

- Data definition language (DDL) is supported if it is executed on a single branch while no other loosely coupled transactions are active. If a loosely coupled branch attempts to start while a single branch executing DDL is active, the loosely coupled branch will be rejected. Conversely, if there is at least one active loosely coupled transaction, then any attempts to execute DDL on another branch will be rejected.
- Loosely coupled global transactions are not supported on massively parallel processing (MPP) environments. In an MPP environment, each global transaction is treated in isolation, where deadlock or timeout might occur.
- You cannot have multiple connections to the same data source in a single two-phase commit transaction
- Savepoint processing and SQL statements are executed serially across multiple connections.
- When an implicit rollback has been performed on one connection, all branches on other connections that are related to the loosely coupled transaction will return SQL0998N, with reason code: 225 and subcode 4: ″Only rollbacks are allowed for this transaction″.

Related concepts:

- "Microsoft Component Services (COM+) as transaction manager" on page 151
- "Microsoft Component Services (COM+) transaction timeout" on page 153
- "COM+ distributed transaction support and the IBM OLE DB Provider" on page 150
- "ODBC and ADO connection pooling with Microsoft Component Services (COM+)" on page 154
- "X/Open distributed transaction processing model" in *Administration Guide: Planning*

Microsoft Component Services (COM+) transaction timeout

Transaction timeout can be set through the following tool when COM+ is used: (Microsoft Windows 2000 and XP): Component Services, located under Administrative Tools of the Windows Control Panel.

If a transaction takes longer than the transaction timeout value (the default value is 60 seconds), COM+ will asynchronously issue an abort to all Resource Managers involved, and the entire transaction is aborted.

The abort is translated into a DB2 rollback request at the server. The rollback request is serialized on the connection, on servers other than DB2 for z/OS and

DB2 for iSeries, to guarantee the integrity of the data on the database server. When the server is DB2 for z/OS or DB2 for iSeries, then the connection should be defined with the INTERRUPT_ENABLED option in the DCS catalog entry so that when a timeout occurs, the connection from the DB2 Connect server to the z/OS or iSeries server will be disconnected, forcing a rollback on the z/OS or iSeries server.

As a result:
- If the connection is idle, the rollback is executed immediately.
- If a long-running SQL statement is processing, the rollback request waits until the SQL statement finishes.

Related concepts:
- "Loosely coupled support with Microsoft Component Services (COM+)" on page 153
- "Microsoft Component Services (COM+) as transaction manager" on page 151
- "COM+ distributed transaction support and the IBM OLE DB Provider" on page 150
- "ODBC and ADO connection pooling with Microsoft Component Services (COM+)" on page 154
- "DCS directory values" in *DB2 Connect User's Guide*
- "X/Open distributed transaction processing model" in *Administration Guide: Planning*

ODBC and ADO connection pooling with Microsoft Component Services (COM+)

Connection pooling enables an application to use a connection from a pool of connections, so that the connection does not need to be re-established for each use. Once a connection has been created and placed in a pool, an application can reuse that connection without performing a complete connection process. The connection is pooled when the application disconnects from the data source and will be given to a new connection whose attributes are the same.

ODBC connection pooling:

Connection pooling has been a feature of the ODBC Driver Manager since ODBC 2.x. With the latest ODBC Driver Manager (version 3.5) available as part of the Microsoft Data Access Components (MDAC) download, connection pooling has some configuration changes and new behavior for ODBC connections of transactional COM+ objects.

The ODBC Driver Manager 3.5 requires that the ODBC driver register a new keyword in the registry before it allows connection pooling to be activated. The keyword is:

```
Key Name: SOFTWARE\ODBC\ODBCINST.INI\IBM DB2 ODBC DRIVER - <DB2 Copy Name>
Name: CPTimeout
Type: REG_SZ
Data: 60
```

The DB2 ODBC driver for the Windows operating system fully supports connection pooling; therefore, this keyword is registered.

The default value of 60 means that the connection will be pooled for 60 seconds before it is disconnected.

In a busy environment, it is better to increase the CPTimeout value to a large number to prevent too many physical connects and disconnects, because these consume large amounts of system resource, including system memory and communications stack resources.

In addition, to ensure that the same connection is used between objects in the same transaction in a multiple processor machine, you must turn off ″multiple pool per processor″ support. To do this, copy the following registry setting into a file called `odbcpool.reg`, save it as a plain text file, and issue the command **odbcpool.reg**. The Windows operating system will import this registry setting.

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\ODBC Connection Pooling]
"NumberOfPools"="1"
```

Without this keyword set to 1, COM+ may pool connections for the same transaction in different pools, and hence may not reuse the same connection.

ADO connection pooling:

If the MTS or COM+ objects use ADO to access the database, you must turn off the OLE DB resource pooling so that the Microsoft OLE DB provider for ODBC (MSDASQL) will not interfere with ODBC connection pooling. This feature was initialized to `OFF` in ADO 2.0, but is initialized to `ON` in ADO 2.1. To turn OLE DB resource pooling off, copy the following lines into a file called `oledb.reg`, save it as a plain text file, and issue the command **oledb.reg**. The Windows operating system will import these registry settings.

```
REGEDIT4

[HKEY_CLASSES_ROOT\CLSID\{c8b522cb-5cf3-11ce-ade5-00aa0044773d}]
@="MSDASQL"
"OLEDB_SERVICES"=dword:fffffffc
```

Related concepts:

- “Loosely coupled support with Microsoft Component Services (COM+)” on page 153
- “Microsoft Component Services (COM+) as transaction manager” on page 151
- “Microsoft Component Services (COM+) transaction timeout” on page 153
- “COM+ distributed transaction support and the IBM OLE DB Provider” on page 150
- “X/Open distributed transaction processing model” in *Administration Guide: Planning*

Troubleshooting a Visual Basic loosely-coupled transaction project

A feature of the Visual Basic integrated environment is that each time you compile a data-linked library (dll), Visual Basic creates a new globally unique identifier (GUID) for it, and registers it in the Windows registry. After the project dll is built, it is registered with the distributed transaction coordinator (DTC) using the GUID. If the project dll is rebuilt, Visual Basic gives it a new GUID, and also registers it in the Windows registry. Therefore, the GUID that the DTC is using to refer to the project dll is now out of date, and there are two different entries in the windows registry for the project.

Procedure:

To avoid this problem, after building the project for the first time, modify the project properties:

1. Under the Component tab, select ″Binary Compatibility″, and then use the ″...″ button to find the whole path for the dll you just built.
2. Then click on the ″OK″ button, and save the project immediately.

Now, every time you recompile the project dll, it will keep the same GUID. However, if you change the interface to the dll (such as by adding or removing methods, or changing the parameters of existing methods) then you have to use a new GUID.

If there are already multiple GUID entries in the windows registry, do the following:

1. Search for the project name with the registry editor.
2. Remove all occurrences of the project name other than what is found in the Visual Basic recent project list.

The connection information, including user ID and password, must be identical in the DTC and in the Visual Basic application in order for the loosely-coupled transaction to occur. Normally, the public methods of a dll are used to directly access it. However, when the DTC is involved, it encapsulates the dll, and intercepts all incoming calls to the methods, and outgoing results from those methods. In this way, the DTC can tell when the database activity in one of those methods should be loosely-coupled with other database activity of the same object or with database activity of a different object.

To avoid any problem this might cause, do the following:

1. In the Application properties in the DTC under the ″Identity″ tab, select ″This user:″.
2. Use the Browse button to find the ID of the user who will run this project.
3. Use the same user ID and password in the project connection string.

If you use the same user ID and password in the Visual Basic application connection string as the one you used to log onto the computer, you do not have to take this additional step.

To ensure the loosely-coupled transaction project is working properly, you can do the following:

1. Look at the output file that the executable creates, and confirm that the database updates are happening.
2. Examine a cli trace for ENLIST_IN_DTC.

If either of these tests fail, then the dll is not registered properly with the DTC, and loosely-coupled transactions are not occurring.

Related tasks:

- "Building ADO applications with Visual Basic" on page 144
- "Building loosely-coupled transactions with Visual Basic" on page 157

Related reference:

- "Visual Basic samples" in *Samples Topics*

Building loosely-coupled transactions with Visual Basic

XA provides two ways by which application threads of control can participate in a single XA global transaction: tightly-coupled and loosely-coupled. The sample project, `LCTransTest`, demonstrates XA loosely-coupled transactions. The sample files are located in the `sqllib\samples\VB\MTS` directory.

Procedure:

To build and run the loosely-coupled transactions sample, follow these steps:

1. **Build the LCTransTest.vbp project**
   a. Open the ″LCTransTest.vbp″ project by double clicking it.
   b. If you received an error message: ″Unable to set the version compatible component X:\...\LCTransTest.dll″, click ″OK″ to continue.
   c. Compile the project. Go to ″File″ -> ″Make LCTransTest.dll″, then click ″OK″.
   d. To fix the version incompatibility problem, right click on the ″LCTransTest (LCTransTest.vbp)″ project located on the upper right panel. Then choose ″LCTransTest Properties″. On the ″LCTransTest - Project Properties″ window. Click on the ″Component″ tab. Under the ″Version Compatibility″ Section, select ″Binary Compatibility″.
   e. Save the project. Go to ″File″ -> ″Save Project″.
   f. Close the project.
2. **Build the Main.vbp project**
   a. Open the ″Main.vbp″ project by double clicking it.
   b. It is likely that you will receive the warning message: ″Could not create reference: X:\...\LCTransTest.dll″, click ″OK″ to continue.
   c. Go to ″Project″ -> ″References″ (on the tool bar).
   d. On the ″References - main.vbp″ window, make sure the ″Microsoft ActiveX Data Objects 2.7 Library″ box is checked. Go to ″Browse...″. Find the LCTransTest.dll you generated in Step 1 and click ″Open″ to add this reference. Click ″OK″ in ″References - main.vbp″ window.
   e. Compile the project. Go to ″File″ -> ″Make main.exe″, then click ″OK″.
   f. Save the project. Go to ″File″ -> ″Save Project″.
   g. Close the project.
3. **Other settings**
   a. On Windows, go to ″Start″ -> ″Settings″ -> ″Control Panel″ -> ″Administration Tools″ -> ″Component Services″.
   b. On the ″Component Services″ window, expand ″Component Services″ on the left panel until you see ″COM+ Applications″.
   c. Right click on ″COM+ Applications″, select ″New″ -> ″Application″.
   d. On the pop-up window, ″COM Application Install Wizard″, click ″Next″.
   e. Select ″Create an empty application″.
   f. Enter ″LCTransTest″ as the name of the new application. Keep ″Activation type″ as ″Server application″. Click ″Next″.
   g. Click ″Next″, then ″Finish″.
   h. Expand ″LCTransTest″, right click on ″Components″. Go to ″New″ -> ″Components″ -> choose ″Import components that are already registered″ _> click on ″LCTransTest.TestClass″ -> ″Next″ -> ″Finish″.

   i. Expand ″Components″. Right click on ″LCTransTest.TestClass″. Go to ″Properties″. Under the ″Transaction″ tab, check ″Required″ for ″Transaction support″
   j. Restart the Microsoft Distributed Transaction Coordinator by right clicking on ″Component Services″ -> ″Computers″ -> ″My Computer″. Choose ″Stop MS DTC″. Wait until it stopped, then right click on ″My Computer″ -> ″Start MS DTC″ to restart DTC.
4. **To run the sample in debug mode**
   a. Open LCTransTest.vbp.
   b. In LCTransTest, ensure that ″project properties*″, under the ″Debugging″ tab has ″Wait for components to be created″ checked. (It should be by default.)
   c. Put a break point on the line ″con1.Open connString″ (by putting your cursor on that line, and pressing F9).
   d. Press F5 (or select ″Start″ under the ″Run″ pull-down menu). When this dll gets loaded, and this method runs, the debugger will stop execution at that break point.
   e. Open main.vbp.
   f. In the main.vbp, set up the command line arguments. In Project properties, under the ″Make″ tab, in the ″Command Line arguments″ text box, type the following:
      ```
      provider=ibmdadb2;dsn=<dbname>;uid=<userid>;pwd=<password> <filename>
      ```
      where <dbname> is the name of a database you have, and <filename> is the name of a file (including the path) that will contain output information. For instance, C:\lctoutput.txt. Then click OK.
   g. Within Visual Basic, you can use the ″Debug ->Step Into″ <F8> or ″Debug->Step Over″ <shift + F8> to run the executable one line of code at a time.
   h. When you get to the line that calls ″transTest.RunTest″ in the main executable, and try to step over it, the other Visual Basic window (the LCTransTest project that you have open) will come to the front, and you'll be stopped at the breakpoint you put there. Then you can use ″Step Into″ or ″Step Over″ to progress through the RunTest method one line of code at a time.

Related tasks:
- "Building ADO applications with Visual Basic" on page 144
- "Troubleshooting a Visual Basic loosely-coupled transaction project" on page 155

Related reference:
- "Visual Basic samples" in *Samples Topics*

Chapter 7. OLE DB .NET Data Provider

OLE DB .NET Data Provider

The OLE DB .NET Data Provider uses the IBM DB2 OLE DB Driver, which is referred to in a `ConnectionString` object as `IBMDADB2`. The connection string keywords supported by the OLE DB .NET Data Provider are the same as those supported by the IBM OLE DB Provider for DB2. Also, the OLE DB .NET Data Provider has the same restrictions as the IBM DB2 OLE DB Provider. There are additional restrictions for the OLE DB .NET Data Provider, which are identified in the topic: OLE DB .NET Data Provider restrictions.

In order to use the OLE DB .NET Data Provider, you must have the .NET Framework Version 1.1 or Version 2.0 installed.

For DB2 Universal Database for AS/400 and iSeries, the following fix is required on the server: APAR ii13348.

The following are all the supported connection keywords for the OLE DB .NET Data Provider:

Table 36. **ConnectionString** *keywords for the OLE DB .NET Data Provider*

Keyword	Value	Meaning
`PROVIDER`	`IBMDADB2`	Specifies the IBM OLE DB Provider for DB2 (required)
`DSN` or `Data Source`	database alias	The DB2 database alias as cataloged in the database directory
`UID`	user ID	The user ID used to connect to the DB2 server
`PWD`	password	The password for the user ID used to connect to the DB2 server

The following is an example of creating an `OleDbConnection` to connect to the SAMPLE database:

```
[Visual Basic .NET]
Dim con As New OleDbConnection("Provider=IBMDADB2;" +
    "Data Source=sample;UID=userid;PWD=password;")
con.Open()

[C#]
OleDbConnection con = new OleDbConnection("Provider=IBMDADB2;" +
                "Data Source=sample;UID=userid;PWD=password;" );
con.Open()
```

OLE DB .NET Data Provider restrictions

The following table identifies usage restrictions for the IBM OLE DB .NET Data Provider:

Table 37. IBM OLE DB .NET Data Provider restrictions

Class or feature	Restriction description	DB2 servers affected
ASCII character streams	You cannot use ASCII character streams with `OleDbParameters` when using `DbType.AnsiString` or `DbType.AnsiStringFixedLength`. The OLE DB .NET Data Provider will throw the following exception: `"Specified cast is not valid"` **Workaround:** Use `DbType.Binary` instead of using `DbType.AnsiString` or `DbType.AnsiStringFixedLength`.	All
`ADORecord`	`ADORecord` is not supported.	All

`ADORecordSet` and `Timestamp`	As documented in MSDN, the `ADORecordSet` variant time resolves to one second. Consequently, all fractional seconds are lost when a DB2 `Timestamp` column is stored into a `ADORecordSet`. Similarly, after filling a `DataSet` from a `ADORecordSet`, the `Timestamp` columns in the `DataSet` will not have any fractional seconds.
	
	Workaround:
	
	This workaround only works for DB2 Universal Database for Linux, UNIX, and Windows, Version 8.1, FixPak 4 or later. In order to avoid the loss of fraction of seconds, you can set the following CLI keyword:
	`MAPTIMESTAMPDESCRIBE = 2`
	
	This keyword will describe the `Timestamp` as a WCHAR(26). To set the keyword, execute the following command from a DB2 Command Window:
	`db2 update cli cfg for section common using MAPTIMESTAMPDESCRIBE 2`
		All

`Chapters`	Chapters are not supported.	All
Key information	The OLE DB .NET Data Provider cannot retrieve key information when opening an `IDataReader` at the same time.	DB2 for VM/VSE
Key information from stored procedures	The OLE DB .NET Data Provider can retrieve key information about a result set returned by a stored procedure only from DB2 Database for Linux, UNIX, and Windows. This is because the DB2 servers for platforms other than Linux, UNIX, and Windows do not return extended describe information for the result sets opened in the stored procedure. In order to retrieve key information of a result set returned by a stored procedure on DB2 Database for Linux, UNIX, and Windows, you need to set the following registry variable on the DB2 server: `db2set DB2_APM_PERFORMANCE=8` Setting this server-side DB2 registry variable will keep the result set meta-data available on the server for a longer period of time, thus allowing OLE DB to successfully retrieve the key information. However, depending on the server workload, the meta-data might not be available long enough before the OLE DB Provider queries for the information. As such, there is no guarantee that the key information will always be available for result sets returned from a store procedure. In order to retrieve any key information about a CALL statement, the application must execute the CALL statement. Calling `OleDbDataAdapter.FillSchema()` or `OleDbCommand.ExecuteReader(CommandBehavior.SchemaOnly | CommandBehavior.KeyInfo)`, will not actually execute the stored procedure call. Therefore, you will not retrieve any key information for the result set that is to be returned by the stored procedure.	All

Table 37. IBM OLE DB .NET Data Provider restrictions (continued)

Class or feature	Restriction description	DB2 servers affected

Key information from batched SQL statements	When using batched SQL statements that return multiple results, the `FillSchema()` method attempts to retrieve schema information only for the first SQL statement in the batched SQL statement list. If this statement does not return a result set then no table is created. For example:
	`[C#]`
	`cmd.CommandText = "INSERT INTO ORG(C1) VALUES(1000); SELECT C1 FROM ORG;";`
	`da = new OleDbDataAdapter(cmd);`
	`da.FillSchema(ds, SchemaType.Source);`
	
	No table will be created in the data set because the first statement in the batch SQL statement is an ″INSERT″ statement, which does not return a result set.
		All

`OleDbCommandBuilder`	The UPDATE, DELETE and INSERT statements automatically generated by the `OleDbCommandBuilder` are incorrect if the SELECT statement contains any columns of the following data types: • CLOB • BLOB • DBCLOB • LONG VARCHAR • LONG VARCHAR FOR BIT DATA • LONG VARGRAPHIC If you are connecting to a DB2 server other than DB2 Database for Linux, UNIX, and Windows, then columns of the following data types also cause this problem: • VARCHAR[1] • VARCHAR FOR BIT DATA[1] • VARGRAPHIC[1] • REAL • FLOAT or DOUBLE • TIMESTAMP **Notes:** 1. Columns of these data types are applicable if they are defined to be VARCHAR values greater than 254 bytes, VARCHAR values FOR BIT DATA greater than 254 bytes, or VARGRAPHICs greater than 127 bytes. This condition is only valid if you are connecting to a DB2 server other than DB2 Database for Linux, UNIX, and Windows. The `OleDbCommandBuilder` generates SQL statements that use all of the selected columns in an equality comparison in the WHERE clause, but the data types listed previously cannot be used in an equality comparison. **Note:** Note that this restriction will affect the `IDbDataAdapter.Update()` method that relies on the `OleDbCommandBuilder` to automatically generate the UPDATE, DELETE, and INSERT statements. The UPDATE operation will fail if the generated statement contains any one of the data types listed previously. **Workaround:** You will need to explicitly remove all columns that are of the data types listed previously from the WHERE clause of the generated SQL statement. It is recommended that you code your own UPDATE, DELETE and INSERT statements.	All
`OleDbCommandBuilder.` `DeriveParameters`	Case-sensitivity is important when using `DeriveParameters()`. The stored procedure name specified in the `OleDbCommand.CommandText` needs to be in the same case as how it is stored in the DB2 system catalog tables. To see how stored procedure names are stored, call OpenSchema(OleDbSchemaGuid.Procedures) without supplying the procedure name restriction. This will return all the stored procedure names. By default, DB2 stores stored procedure names in uppercase, so most often, you need to specify the stored procedure name in uppercase.	All

Table 37. IBM OLE DB .NET Data Provider restrictions (continued)

Class or feature	Restriction description	DB2 servers affected
`OleDbCommandBuilder.` `DeriveParameters`	The `OleDbCommandBuilder.DeriveParameters()` method does not include the `ReturnValue` parameter in the generated `OleDbParameterCollection`. SqlClient and the DB2 .NET Data Provider by default adds the parameter with `ParameterDirection.ReturnValue` to the generated `ParameterCollection`.	All
`OleDbCommandBuilder.` `DeriveParameters`	The `OleDbCommandBuilder.DeriveParameters()` method will fail for overloaded stored procedures. If you have multiple stored procedures of the name ″MYPROC″ with each of them taking a different number of parameters or different type of parameter, the `OleDbCommandBuilder.DeriveParameters()` will retrieve all the parameters for all the overloaded stored procedures.	All
`OleDbCommandBuilder.` `DeriveParameters`	If the application does not qualify a stored procedure with a schema, `DeriveParameters()` will return all the parameters for that procedure name. Therefore, if multiple schemas exist for the same procedure name, `DeriveParameters()` will return all the parameters for all the procedures with the same name.	All
`OleDbConnection.` `ChangeDatabase`	The `OleDbConnection.ChangeDatabase()` method is not supported.	All

`OleDbConnection.` `ConnectionString`	Use of nonprintable characters such as ’\b’, ’\a’ or ’\O’ in the connection string will cause an exception to be thrown.
	
	The following keywords have restrictions:
	
	`Data Source`
	The data source is the name of the database, not the server. You can specify the SERVER keyword, but it is ignored by the IBMDADB2 provider.
	
	`Initial Catalog and Connect Timeout`
	These keywords are not supported. In general, the OLE DB .NET Data Provider will ignore all unrecognized and unsupported keywords. However, specifying these keywords will cause the following exception:
	`Multiple-step OLE DB operation generated errors. Check each`
	`OLE DB status value, if available. No work was done.`
	
	`ConnectionTimeout`
	`ConnectionTimeout` is read only.
		All

`OleDbConnection.` `GetOleDbSchemaTable`	Restriction values are case-sensitive, and need to match the case of the database objects stored in the system catalog tables, which defaults to uppercase.
	
	For instance, if you have created a table in the following manner:
	`CREATE TABLE abc(c1 SMALLINT)`
	
	DB2 will store the table name in uppercase (″ABC″) in the system catalog. Therefore, you will need to use ″ABC″ as the restriction value. For instance:
	`schemaTable = con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables,`
	`        new object[] { null, null, "ABC", "TABLE" });`
	
	Workaround:
	
	If you need case-sensitivity or spaces in your data definitions, you must put quotation marks around them. For example:
	`cmd.CommandText = "create table \"Case Sensitive\"(c1 int)";`
	`cmd.ExecuteNonQuery();`
	`tablename = "\"Case Sensitive\"";`
	`schemaTable = con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables,`
	`    new object[] { null, null, tablename, "TABLE" });`
		All

`OleDbDataAdapter` and `DataColumnMapping`	The source column name is case-sensitive. It needs to match the case as stored in the DB2 catalogs, which by default is uppercase. For example: `colMap = new DataColumnMapping("EMPNO", "Employee ID");`	All

Table 37. IBM OLE DB .NET Data Provider restrictions (continued)

Class or feature	Restriction description	DB2 servers affected
`OleDbDataReader.` `GetSchemaTable`	The OLE DB .NET Data Provider is not able to retrieve extended describe information from servers that do not return extended describe information. if you are connecting to a server that does not support extended describe (the affected servers), the following columns in the metadata table returned from `IDataReader.GetSchemaTable()` are invalid: • `IsReadOnly` • `IsUnique` • `IsAutoIncrement` • `BaseSchemaName` • `BaseCatalogName`	DB2 for OS/390, version 7 or lower DB2 for OS/400 DB2 for VM/VSE
Stored procedures: no column names for result sets	The DB2 for OS/390 version 6.1 server does not return column names for result sets returned from a stored procedure. The OLE DB .NET Data Provider maps these unnamed columns to their ordinal position (for example, ″1″, ″2″ ″3″). This is contrary to the mapping documented in MSDN: `"Column1", "Column2", "Column3"`.	DB2 for OS/390 version 6.1

Connection pooling in OLE DB .NET Data Provider applications

The OLE DB .NET Data Provider automatically pools connections using OLE DB session pooling. Connection string arguments can be used to enable or disable OLE DB services including pooling. For example, the following connection string will disable OLE DB session pooling and automatic transaction enlistment.

```
Provider=IBMDADB2;OLE DB Services=-4;Data Source=SAMPLE;
```

The following table describes the ADO connection string attributes you can use to set the OLE DB services:

Table 38. Setting OLE DB services by using ADO connection string attributes

Services enabled	Value in connection string
All services (the default)	″OLE DB Services = -1;″
All services except pooling	″OLE DB Services = -2;″
All services except pooling and auto-enlistment	″OLE DB Services = -4;″
All services except client cursor	″OLE DB Services = -5;″
All services except client cursor and pooling	″OLE DB Services = -6;″
No services	″OLE DB Services = 0;″

For more information about OLE DB session pooling or resource pooling, as well as how to disable pooling by overriding OLE DB provider service defaults, see the OLE DB Programmer's Reference in the MSDN library located at:

```
http://msdn.microsoft.com/library
```

Time columns in OLE DB .NET Data Provider applications

The following sections describe how to implement time columns in OLE DB .NET Data Provider applications.

Inserting using parameter markers:

You want to insert a time value into a Time column:

```
command.CommandText = "insert into mytable(c1) values( ? )";
```

where column c1 is a Time column. Here are two methods to bind a time value to the parameter marker:

Using `OleDbParameter.OleDbType = OleDbType.DBTime`

Because OleDbType.DBTime maps to a TimeSpan object, you must supply a TimeSpan object as the parameter value. The parameter value cannot be a String or a DateTime object, it must be a TimeSpan object. For example:

```
p1.OleDbType = OleDbType.DBTime;
p1.Value = TimeSpan.Parse("0.11:20:30");
rowsAffected = cmd.ExecuteNonQuery();
```

The format of the TimeSpan is represented as a string in the format ″[-]d.hh:mm:ss.ff″ as documented in the MSDN documentation.

Using `OleDbParameter.OleDbType = OleDbType.DateTime`

This will force the OLE DB .NET Data Provider to convert the parameter value to a DateTime object, instead of a TimeSpan object, consequently the parameter value can be any valid string/object that can be converted into a DateTime object. This means values such as ″11:20:30″ will work. The value can also be a DateTime object. The value cannot be a TimeSpan object since a TimeSpan object cannot be converted to a DateTime object -- TimeSpan doesn't implement IConvertible.

For example:

```
p1.OleDbType = OleDbType.DBTimeStamp;
p1.Value = "11:20:30";
rowsAffected = cmd.ExecuteNonQuery();
```

Retrieval:

To retrieve a time column you need to use the `IDataRecord.GetValue()` method or the `OleDbDataReader.GetTimeSpan()` method.

For example:

```
TimeSpan ts1 = ((OleDbDataReader)reader).GetTimeSpan( 0 );
TimeSpan ts2 = (TimeSpan) reader.GetValue( 0 );
```

ADORecordset objects in OLE DB .NET Data Provider applications

Following are considerations regarding the use of `ADORecordset` objects.

- The ADO type `adDBTime` class is mapped to the .NET Framework `DateTime` class. `OleDbType.DBTime` corresponds to a `TimeSpan` object.
- You cannot assign a `TimeSpan` object to an `ADORecordset` object's `Time` field. This is because the `ADORecordset` object's `Time` field expects a `DateTime` object. When you assign a `TimeSpan` object to an `ADORecordset` object, you will get the following message:

  ```
  Method's type signature is not Interop compatible.
  ```

  You can only populate the `Time` field with a `DateTime` object, or a `String` that can be parsed into a `DateTime` object.
- When you fill a `DataSet` with a `ADORecordset` using the `OleDbDataAdapter`, the `Time` field in the `ADORecordset` is converted to a `TimeSpan` column in the `DataSet`.

- `Recordsets` do not store primary keys or constraints. Therefore, no key information is added when filling out a `DataSet` from a `Recordset` using the `MissingSchemaAction.AddWithKey`.

Chapter 8. ODBC .NET Data Provider

ODBC .NET Data Provider

The ODBC .NET Data Provider makes ODBC calls to a DB2 data source using the DB2 CLI Driver. Therefore, the connection string keywords supported by the ODBC .NET Data Provider are the same as those supported by the DB2 CLI driver. Also, the ODBC .NET Data Provider has the same restrictions as the DB2 CLI driver. There are additional restrictions for the ODBC .NET Data Provider, which are identified in the topic: ODBC .NET Data Provider restrictions.

In order to use the ODBC .NET Data Provider, you must have the .NET Framework Version 1.1 or Version 2.0 installed. For DB2 Universal Database for AS/400 and iSeries, the following fix is required on the server: APAR II13348.

The following are the supported connection keywords for the ODBC .NET Data Provider:

Table 39. **ConnectionString** *keywords for the ODBC .NET Data Provider*

Keyword	Value	Meaning
`DSN`	database alias	The DB2 database alias as cataloged in the database directory
`UID`	user ID	The user ID used to connect to the DB2 server
`PWD`	password	The password for the user ID used to connect to the DB2 server

The following is an example of creating an `OdbcConnection` to connect to the SAMPLE database:

```
[Visual Basic .NET]
Dim con As New OdbcConnection("DSN=sample;UID=userid;PWD=password;")
con.Open()

[C#]
OdbcConnection con = new OdbcConnection("DSN=sample;UID=userid;PWD=password;");
con.Open()
```

ODBC .NET Data Provider restrictions

The following table identifies usage restrictions for the IBM ODBC .NET Data Provider:

Table 40. IBM ODBC .NET Data Provider restrictions

Class or feature	Restriction description	DB2 servers affected
ASCII character streams	You cannot use ASCII character streams with `OdbcParameters` when using `DbType.AnsiString` or `DbType.AnsiStringFixedLength`. The ODBC .NET Data Provider will throw the following exception: `"Specified cast is not valid"` **Workaround:** Use `DbType.Binary` instead of using `DbType.AnsiString` or `DbType.AnsiStringFixedLength`.	All

`Command.Prepare`	Before executing a command (`Command.ExecuteNonQuery` or `Command.ExecuteReader`), you must explicitly run `OdbcCommand.Prepare()` if the `CommandText` has changed since the last prepare. If you do not call `OdbcCommand.Prepare()` again, the ODBC .NET Data Provider will execute the previously prepared `CommandText`.
	
	For Example:
	`[C#]`
	`command.CommandText="select CLOB('ABC') from table1";`
	`command.Prepare();`
	`command.ExecuteReader();`
	`command.CommandText="select CLOB('XYZ') from table2";`
	`command.ExecuteReader();   // This ends up re-executing the first statement`
		All

CommandBehavior. SequentialAccess	When using `IDataReader.GetChars()` to read from a reader created with `CommandBehavior.SequentialAccess`, you must allocate a buffer that is large enough to hold the entire column. Otherwise, you will hit the following exception:
	`Requested range extends past the end of the array.`
	`   at System.Runtime.InteropServices.Marshal.Copy(Int32 source,`
	`     Char[] destination, Int32 startIndex, Int32 length)`
	`   at System.Data.Odbc.OdbcDataReader.GetChars(Int32 i,`
	`     Int64 dataIndex, Char[] buffer, Int32 bufferIndex, Int32 length)`
	`   at OleRestrict.TestGetCharsAndBufferSize(IDbConnection con)`
	
	The following example demonstrates how to allocate an adequate buffer:
	`CREATE TABLE myTable(c0 int, c1 CLOB(10K))`
	`SELECT c1 FROM myTable;`
	
	`[C#]`
	`cmd.CommandText = "SELECT c1 from myTable";`
	`IDataReader reader = cmd.ExecuteReader(CommandBehavior.SequentialAccess);`
	
	`Int32 iChunkSize = 10;`
	`Int32 iBufferSize = 10;`
	`Int32 iFieldOffset = 0;`
	
	`Char[] buffer = new Char[ iBufferSize ];`
	
	`reader.Read();`
	`reader.GetChars(0, iFieldOffset, buffer, 0, iChunkSize);`
	
	The call to `GetChars()` will throw the following exception:
	`"Requested range extends past the end of the array"`
	
	To ensure that `GetChars()` does not throw the above exception, you must set the `BufferSize` to the size of the column, as follows:
	`Int32 iBufferSize = 10000;`
	
	Note that the value of 10,000 for `iBufferSize` corresponds to the value of 10K allocated to the CLOB column `c1`.
		All

Table 40. IBM ODBC .NET Data Provider restrictions (continued)

Class or feature	Restriction description	DB2 servers affected

CommandBehavior. SequentialAccess	The ODBC .NET Data Provider throws the following exception when there is no more data to read when using `OdbcDataReader.GetChars()`:
	`NO_DATA - no error information available`
	`   at System.Data.Odbc.OdbcConnection.HandleError(HandleRef hrHandle,`
	`     SQL_HANDLE hType, RETCODE retcode)`
	`   at System.Data.Odbc.OdbcDataReader.GetData(Int32 i, SQL_C sqlctype,`
	`     Int32 cb)`
	`   at System.Data.Odbc.OdbcDataReader.GetChars(Int32 i, Int64 dataIndex,`
	`     Char[] buffer, Int32 bufferIndex, Int32 length)`
		All

CommandBehavior. SequentialAccess	You may not be able to use large chunksizes, such as a value of 5000, when using `OdbcDataReader.GetChars()`. When you attempt to use a large chunk size, the ODBC .NET Data Provider will throw the following exception:
	`Object reference not set to an instance of an object.`
	`   at System.Runtime.InteropServices.Marshal.Copy(Int32 source,`
	`     Char[] destination, Int32 startIndex, Int32 length)`
	`   at System.Data.Odbc.OdbcDataReader.GetChars(Int32 i, Int64 dataIndex,`
	`     Char[] buffer, Int32 bufferIndex, Int32 length)`
	`   at OleRestrict.TestGetCharsAndBufferSize(IDbConnection con)`
		All

Connection pooling	The ODBC .NET Data Provider does not control connection pooling. Connection pooling is handled by the ODBC Driver Manager. For more information on connection pooling, see the ODBC Programmer's Reference in the MSDN library located at `http://msdn.microsoft.com/library`	All
DataColumnMapping	The case of the source column name needs to match the case used in the system catalog tables, which is upper-case by default.	All

Decimal columns	Parameter markers are not supported for Decimal columns.
	
	You generally use `OdbcType.Decimal` for an `OdbcParameter` if the target `SQLType` is a `Decimal` column; however, when the ODBC .NET Data Provider sees the `OdbcType.Decimal`, it binds the parameter using C-type of `SQL_C_WCHAR` and `SQLType` of `SQL_VARCHAR`, which is invalid.
	
	For example:
	`[C#]`
	`cmd.CommandText = "SELECT dec_col FROM MYTABLE WHERE dec_col > ? ";`
	`OdbcParameter p1 = cmd.CreateParameter();`
	`p1.DbType = DbType.Decimal;`
	`p1.Value = 10.0;`
	`cmd.Parameters.Add(p1);`
	`IDataReader rdr = cmd.ExecuteReader();`
	
	You will get an exception:
	`ERROR [07006] [IBM][CLI Driver][SQLDS/VM] SQL0301N  The value of input`
	`   host variable or parameter number "" cannot be used because of its`
	`   data type.  SQLSTATE=07006`
	
	Workaround:
	
	Instead of using `OdbcParameter` values, use literals exclusively.
		DB2 for VM/VSE

Table 40. IBM ODBC .NET Data Provider restrictions (continued)

Class or feature	Restriction description	DB2 servers affected

Key information	The schema name used to qualify the table name (for example, `MYSCHEMA.MYTABLE`) must match the connection user ID. The ODBC .NET Data Provider is unable to retrieve any key information in which the specified schema is different from the connection user id.
	
	For example:
	`CREATE TABLE USERID2.TABLE1(c1 INT NOT NULL PRIMARY KEY);`
	
	`[C#]`
	`// Connect as user bob`
	`odbcCon = new OdbcConnection("DSN=sample;UID=bob;PWD=mypassword");`
	
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	
	`// Select from table with schema USERID2`
	`cmd.CommandText="SELECT * FROM USERID2.TABLE1";`
	
	`// Fails - No key info retrieved`
	`da.FillSchema(ds, SchemaType.Source);`
	
	`// Fails - SchemaTable has no primary key`
	`cmd.ExecuteReader(CommandBehavior.KeyInfo)`
	
	`// Throws exception because no primary key`
	`cbuilder.GetUpdateCommand();`
		All

Key information	The ODBC .NET Data Provider cannot retrieve key information when opening a `IDataReader` at the same time. When the ODBC .NET Data Provider opens a `IDataReader`, a cursor on the server is opened. If key information is requested, it will then call `SQLPrimaryKeys()` or `SQLStatistic()` to get the key information, but these schema functions will open another cursor. Since DB2 for VM/VSE does not support cursor withhold, the first cursor is then closed. Consequently, `IDataReader.Read()` calls to the `IDataReader` will result in the following exception:
	`System.Data.Odbc.OdbcException: ERROR [HY010] [IBM][CLI Driver]`
	`  CLI0125E  Function sequence error. SQLSTATE=HY010`
	
	Workaround:
	
	You will need to retrieve key information first then retrieve the data.
	
	For example:
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`OdbcDataAdapter da = new OdbcDataAdapter(cmd);`
	
	`cmd.CommandText  = "SELECT * FROM MYTABLE";`
	
	`// Use FillSchema to retrieve just the schema information`
	`da.FillSchema(ds, SchemaType.Source);`
	`// Use FillSchema to retrieve just the schema information`
	`da.Fill(ds);`
		DB2 for VM/VSE

Key information	You must refer to database objects in your SQL statements using the same case that the database objects are stored in the system catalog tables. By default database objects are stored in uppercase in the system catalog tables, so most often, you need to use uppercase. The ODBC .NET Data Provider scans SQL statements to retrieve database object names and passes them to schema functions such as `SQLPrimaryKeys` and `SQLStatistics`, which issue queries for these objects in the system catalog tables. The database object references must match exactly how they are stored in the system catalog tables, otherwise, an empty result set is returned.	DB2 for OS/390 DB2 for OS/400 DB2 for VM/VSE
Key information for batched non-select SQL statements	The ODBC .NET Data Provider is unable to retrieve any key information for a batch statement that does not start with ″SELECT″.	DB2 for OS/390 DB2 for OS/400 DB2 for VM/VSE

Table 40. IBM ODBC .NET Data Provider restrictions (continued)

Class or feature	Restriction description	DB2 servers affected

LOB columns	The ODBC .NET Data Provider does not support LOB datatypes. Consequently, whenever the DB2 server returns a SQL_CLOB (-99), SQL_BLOB (-98) or SQL_DBCLOB (-350) the ODBC .NET Data Provider will throw the following exception:
	`"Unknown SQL type - -98"     (for Blob column)`
	`"Unknown SQL type - -99"     (for Clob column)`
	`"Unknown SQL type - -350"    (for DbClob column)`
	
	Any methods that directly or indirectly access LOB columns will fail.
	
	Workaround:
	
	Set the CLI/ODBC `LongDataCompat` keyword to 1. Doing so will force the DB2 CLI driver to make the following data type mappings to data types the ODBC .NET Data Provider will understand:
	• SQL_CLOB to SQL_LONGVARCHAR
	• SQL_BLOB to SQL_LONGVARBINARY
	• SQL_DBCLOB to SQL_WLONGVARCHAR
	
	To set the `LongDataCompat` keyword, run the following DB2 command from a DB2 command window on the client machine:
	`db2 update cli cfg for section common using longdatacompat 1`
	
	You can also set this keyword in your application, using the connection string as follows:
	`[C#]`
	`OdbcConnection con =`
	`  new OdbcConnection("DSN=SAMPLE;UID=uid;PWD=mypwd;LONGDATACOMPAT=1;");`
	
	For a list of all the CLI/ODBC keywords, see the following topic: UID CLI/ODBC configuration keyword in the DB2 CLI Guide and Reference.
		All

`OdbcCommand.Cancel`	Executing statements after running `OdbcCommand.Cancel` can lead to the following exception: `"ERROR [24000] [Microsoft][ODBC Driver Manager] Invalid cursor state"`	All

`OdbcCommandBuilder`	The `OdbcCommandBuilder` fails to generate commands against servers that do not support escape characters. When the `OdbcCommandBuilder` generates commands, it first makes a call to `SQLGetInfo`, requesting the `SQL_SEARCH_PATTERN_ESCAPE` attribute. If the server does not support escape characters an empty string is returned, which causes the ODBC .NET Data Provider to throw the following exception:
	`Index was outside the bounds of the array.`
	`   at System.Data.Odbc.OdbcConnection.get_EscapeChar()`
	`   at System.Data.Odbc.OdbcDataReader.GetTableNameFromCommandText()`
	`   at System.Data.Odbc.OdbcDataReader.BuildMetaDataInfo()`
	`   at System.Data.Odbc.OdbcDataReader.GetSchemaTable()`
	`   at System.Data.Common.CommandBuilder.BuildCache(Boolean closeConnection)`
	`   at System.Data.Odbc.OdbcCommandBuilder.GetUpdateCommand()`
		DB2 for OS/390, DBCS servers only; DB2 for VM/VSE, DBCS servers only

Table 40. IBM ODBC .NET Data Provider restrictions (continued)

Class or feature	Restriction description	DB2 servers affected

`OdbcCommandBuilder`	Case-sensitivity is important when using the `OdbcCommandBuilder` to automatically generate UPDATE, DELETE, and INSERT statements. By default, DB2 stores schema information (such as table names, and column names) in the system catalog tables in upper case, unless they have been explicitly created with case-sensitivity (by adding quotes around database objects during create-time). As such, your SQL statements must match the case that is stored in the catalogs (which by default is uppercase).
	
	For example, if you created a table using the following statement:
	`"db2 create table mytable (c1 int) "`
	
	then DB2 will store the table name ″mytable″ in the system catalog tables as ″MYTABLE″.
	
	The following code example demonstrates proper use the `OdbcCommandBuilder`class:
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`cmd.CommandText  = "SELECT * FROM MYTABLE";`
	`OdbcDataAdapter da = new OdbcDataAdapter(cmd);`
	`OdbcCommandBuilder cb = new OdbcCommandBuilder(da);`
	`OdbcCommand updateCmd = cb.GetUpdateCommand();`
	
	In this example, if you do not refer to the table name in upper-case characters, then you will get the following exception:
	`"Dynamic SQL generation for the UpdateCommand is not`
	`supported against a SelectCommand that does not return`
	`any key column information."`
		All

OdbcCommandBuilder	The commands generated by the `OdbcCommandBuilder` are incorrect when the SELECT statement contains the following column data types:
	`REAL`
	`FLOAT or DOUBLE`
	`TIMESTAMP`
	
	These data types cannot be used in the WHERE clause for SELECT statements.
		DB2 for OS/390
		DB2 for OS/400
		DB2 for VM/VSE

OdbcCommandBuilder. DeriveParameters	The `DeriveParameters()` method is mapped to `SQLProcedureColumns` and it uses the `CommandText` property for the name of the stored procedure. Since `CommandText` does not contain the name of the stored procedure (using full ODBC call syntax), `SQLProcedureColumns` is called with the procedure name identified according to the ODBC call syntax. For example: `"{ CALL myProc(?) }"` This which will result in an empty result set, where no columns are found for the procedure).	All
OdbcCommandBuilder. DeriveParameters	To use `DeriveParameters()`, specify the stored procedure name in the `CommandText` (for example, `cmd.CommandText = "MYPROC"`). The procedure name must match the case stored in the system catalog tables. `DeriveParameters()` will return all the parameters for that procedure name it finds in the system catalog tables. Remember to change the `CommandText` back to the full ODBC call syntax before executing the statement.	All
OdbcCommandBuilder. DeriveParameters	The `ReturnValue` parameter is not returned for the ODBC .NET Data Provider.	All
OdbcCommandBuilder. DeriveParameters	`DeriveParameters()` does not support fully qualified stored procedure names. For example, calling `DeriveParameters()` for `CommandText = "MYSCHEMA.MYPROC"` will fail. Here, no parameters are returned.	All
OdbcCommandBuilder. DeriveParameters	`DeriveParameters()` will not work for overloaded stored procedures. The `SQLProcedureColumns` will return all the parameters for all versions of the stored procedure.	All
`OdbcConnection. ChangeDatabase`	The `OdbcConnection.ChangeDatabase()` method is not supported.	All

Table 40. IBM ODBC .NET Data Provider restrictions (continued)

Class or feature	Restriction description	DB2 servers affected
`OdbcConnection.` `ConnectionString`	• The `Server` keyword is ignored. • The `Connect Timeout` keyword is ignored. DB2 CLI does not support connection timeouts, so setting this property will not affect the driver. • Connection pooling keywords are ignored. Specifically, this affects the following keywords: `Pooling`, `Min Pool Size`, `Max Pool Size`, `Connection Lifetime` and `Connection Reset`.	All
OdbcDataReader. GetSchemaTable	The ODBC .NET Data Provider is not able to retrieve extended describe information from servers that do not return extended describe information. Therefore, if you are connecting to a server that does not support extended describe (the affected servers), the following columns in the metadata table returned from `IDataReader.GetSchemaTable()` are invalid: • `IsReadOnly` • `IsUnique` • `IsAutoIncrement` • `BaseSchemaName` • `BaseCatalogName`	DB2 for OS/390, version 7 or lower DB2 for OS/400 DB2 for VM/VSE

Stored procedures	To call a stored procedure, you need to specify the full ODBC call syntax.
	
	For example, to call the stored procedure, `MYPROC`, that takes a VARCHAR(10) as a parameter:
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`cmd.CommandType = CommandType.Text;`
	`cmd.CommandText = "{ CALL MYPROC(?) }"`
	`OdbcParameter p1 = cmd.CreateParameter();`
	`p1.Value = "Joe";`
	`p1.OdbcType = OdbcType.NVarChar;`
	`cmd.Parameters.Add(p1);`
	`cmd.ExecuteNonQuery();`
	
	Note: Note that you must use the full ODBC call syntax even if you are using `CommandType.StoredProcedure`. This is documented in MSDN, under the OdbcCommand.CommandText Property.
		All

Stored procedures: no column names for result sets	The DB2 for OS/390 version 6.1 server does not return column names for result sets returned from a stored procedure. The ODBC .NET Data Provider maps these unnamed columns to their ordinal position (for example, ″1″, ″2″ ″3″). This is contrary to the mapping documented in MSDN: `"Column1"`, `"Column2"`, `"Column3"`.	DB2 for OS/390 version 6.1
Unique index promotion to primary key	The ODBC .NET Data Provider promotes nullable unique indexes to primary keys. This is contrary to the MSDN documentation, which states that nullable unique indexes should not be promoted to primary keys.	All

Appendix A. DB2 Database technical information

Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics
  - Help for DB2 tools
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF CD)
  - printed books
- Command line help
  - Command help
  - Message help
- Sample programs

IBM periodically makes documentation updates available. If you access the online version on the DB2 Information Center at ibm.com®, you do not need to install documentation updates because this version is kept up-to-date by IBM. If you have installed the DB2 Information Center, it is recommended that you install the documentation updates. Documentation updates allow you to update the information that you installed from the *DB2 Information Center CD* or downloaded from Passport Advantage as new information becomes available.

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hard-copy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com.

You can access additional DB2 technical information such as technotes, white papers, and Redbooks™ online at ibm.com. Access the DB2 Information Management software library site at http://www.ibm.com/software/data/sw-library/.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how we can improve the DB2 documentation, send an e-mail to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this e-mail address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

Related concepts:

- "Features of the DB2 Information Center" in *Online DB2 Information Center*
- "Sample files" in *Samples Topics*

Related tasks:

- "Invoking command help from the command line processor" in *Command Reference*
- "Invoking message help from the command line processor" in *Command Reference*
- "Updating the DB2 Information Center installed on your computer or intranet server" on page 181

Related reference:

- "DB2 technical library in hardcopy or PDF format" on page 176

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/shop/publications/order. DB2 Version 9 manuals in PDF format can be downloaded from www.ibm.com/software/data/db2/udb/support/manualsv9.html.

Although the tables identify books available in print, the books might not be available in your country or region.

The information in these books is fundamental to all DB2 users; you will find this information useful whether you are a programmer, a database administrator, or someone who works with DB2 Connect or other DB2 products.

Table 41. DB2 technical information

Name	Form Number	Available in print
Administration Guide: Implementation	SC10-4221	Yes
Administration Guide: Planning	SC10-4223	Yes
Administrative API Reference	SC10-4231	Yes
Administrative SQL Routines and Views	SC10-4293	No
Call Level Interface Guide and Reference, Volume 1	SC10-4224	Yes
Call Level Interface Guide and Reference, Volume 2	SC10-4225	Yes
Command Reference	SC10-4226	No
Data Movement Utilities Guide and Reference	SC10-4227	Yes
Data Recovery and High Availability Guide and Reference	SC10-4228	Yes
Developing ADO.NET and OLE DB Applications	SC10-4230	Yes
Developing Embedded SQL Applications	SC10-4232	Yes

Table 41. DB2 technical information (continued)

Name	Form Number	Available in print
Developing SQL and External Routines	SC10-4373	No
Developing Java Applications	SC10-4233	Yes
Developing Perl and PHP Applications	SC10-4234	No
Getting Started with Database Application Development	SC10-4252	Yes
Getting started with DB2 installation and administration on Linux and Windows	GC10-4247	Yes
Message Reference Volume 1	SC10-4238	No
Message Reference Volume 2	SC10-4239	No
Migration Guide	GC10-4237	Yes
Net Search Extender Administration and User's Guide **Note:** HTML for this document is not installed from the HTML documentation CD.	SH12-6842	Yes
Performance Guide	SC10-4222	Yes
Query Patroller Administration and User's Guide	GC10-4241	Yes
Quick Beginnings for DB2 Clients	GC10-4242	No
Quick Beginnings for DB2 Servers	GC10-4246	Yes
Spatial Extender and Geodetic Data Management Feature User's Guide and Reference	SC18-9749	Yes
SQL Guide	SC10-4248	Yes
SQL Reference, Volume 1	SC10-4249	Yes
SQL Reference, Volume 2	SC10-4250	Yes
System Monitor Guide and Reference	SC10-4251	Yes
Troubleshooting Guide	GC10-4240	No
Visual Explain Tutorial	SC10-4319	No
What's New	SC10-4253	Yes
XML Extender Administration and Programming	SC18-9750	Yes
XML Guide	SC10-4254	Yes
XQuery Reference	SC18-9796	Yes

Table 42. DB2 Connect-specific technical information

Name	Form Number	Available in print
DB2 Connect User's Guide	SC10-4229	Yes

Table 42. DB2 Connect-specific technical information (continued)

Name	Form Number	Available in print
Quick Beginnings for DB2 Connect Personal Edition	GC10-4244	Yes
Quick Beginnings for DB2 Connect Servers	GC10-4243	Yes

Table 43. WebSphere® Information Integration technical information

Name	Form Number	Available in print
WebSphere Information Integration: Administration Guide for Federated Systems	SC19-1020	Yes
WebSphere Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018	Yes
WebSphere Information Integration: Configuration Guide for Federated Data Sources	SC19-1034	No
WebSphere Information Integration: SQL Replication Guide and Reference	SC19-1030	Yes

Note: The DB2 Release Notes provide additional information specific to your product's release and fix pack level. For more information, see the related links.

Related concepts:

- "Overview of the DB2 technical information" on page 175
- "About the Release Notes" in *Release Notes*

Related tasks:

- "Ordering printed DB2 books" on page 178

Ordering printed DB2 books

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation* CD are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the DB2 PDF Documentation CD can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the DB2 PDF Documentation CD are available in print.

Note: The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at http://publib.boulder.ibm.com/infocenter/db2help/.

Procedure:

To order printed DB2 books:

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at http://www.ibm.com/shop/publications/order. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.
- To order printed DB2 books from your local IBM representative:
  - Locate the contact information for your local representative from one of the following Web sites:
    - The IBM directory of world wide contacts at www.ibm.com/planetwide
    - The IBM Publications Web site at http://www.ibm.com/shop/publications/order. You will need to select your country, region, or language to the access appropriate publications home page for your location. From this page, follow the ″About this site″ link.
  - When you call, specify that you want to order a DB2 publication.
  - Provide your representative with the titles and form numbers of the books that you want to order.

Related concepts:

- "Overview of the DB2 technical information" on page 175

Related reference:

- "DB2 technical library in hardcopy or PDF format" on page 176

Displaying SQL state help from the command line processor

DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

Procedure:

To invoke SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

For example, `? 08003` displays help for the 08003 SQL state, and `? 08` displays help for the 08 class code.

Related tasks:

- "Invoking command help from the command line processor" in *Command Reference*
- "Invoking message help from the command line processor" in *Command Reference*

Accessing different versions of the DB2 Information Center

For DB2 Version 9 topics, the DB2 Information Center URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/.

For DB2 Version 8 topics, go to the Version 8 Information Center URL at: http://publib.boulder.ibm.com/infocenter/db2luw/v8/.

Related tasks:

- "Setting up access to DB2 contextual help and documentation" in *Administration Guide: Implementation*

Displaying topics in your preferred language in the DB2 Information Center

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

Procedure:

To display topics in your preferred language in the Internet Explorer browser:

1. In Internet Explorer, click the **Tools** —> **Internet Options** —> **Languages...** button. The Language Preferences window opens.
2. Ensure your preferred language is specified as the first entry in the list of languages.
   - To add a new language to the list, click the **Add...** button.

     Note: Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.
   - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

To display topics in your preferred language in a Firefox or Mozilla browser:

1. Select the **Tools** —> **Options** —> **Languages** button. The Languages panel is displayed in the Preferences window.
2. Ensure your preferred language is specified as the first entry in the list of languages.
   - To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
   - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

On some browser and operating system combinations, you might have to also change the regional settings of your operating system to the locale and language of your choice.

Related concepts:

- "Overview of the DB2 technical information" on page 175

Updating the DB2 Information Center installed on your computer or intranet server

If you have a locally-installed DB2 Information Center, updated topics can be available for download. The 'Last updated' value found at the bottom of most topics indicates the current level for that topic.

To determine if there is an update available for the entire DB2 Information Center, look for the 'Last updated' value on the Information Center home page. Compare the value in your locally installed home page to the date of the most recent downloadable update at http://www.ibm.com/software/data/db2/udb/support/icupdate.html. You can then update your locally-installed Information Center if a more recent downloadable update is available.

Updating your locally-installed DB2 Information Center requires that you:

1. Stop the DB2 Information Center on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to download and apply updates.
2. Use the Update feature to determine if update packages are available from IBM.

   Note: Updates are also available on CD. For details on how to configure your Information Center to install updates from CD, see the related links.

   If update packages are available, use the Update feature to download the packages. (The Update feature is only available in stand-alone mode.)
3. Stop the stand-alone Information Center, and restart the DB2 Information Center service on your computer.

Procedure:

To update the DB2 Information Center installed on your computer or intranet server:

1. Stop the DB2 Information Center service.
   - On Windows, click **Start → Control Panel → Administrative Tools → Services**. Then right-click on **DB2 Information Center** service and select **Stop**.
   - On Linux, enter the following command:
     ```
     /etc/init.d/db2icdv9 stop
     ```
2. Start the Information Center in stand-alone mode.
   - On Windows:
     a. Open a command window.
     b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `C:\Program Files\IBM\DB2 Information Center\Version 9` directory.
     c. Run the `help_start.bat` file using the fully qualified path for the DB2 Information Center:
        ```
        <DB2 Information Center dir>\doc\bin\help_start.bat
        ```
   - On Linux:

   a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `/opt/ibm/db2ic/V9` directory.
   b. Run the `help_start` script using the fully qualified path for the DB2 Information Center:

      ```
      <DB2 Information Center dir>/doc/bin/help_start
      ```

   The systems default Web browser launches to display the stand-alone Information Center.
3. Click the Update button (). On the right hand panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the download process, check the selections you want to download, then click **Install Updates**.
5. After the download and installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center.
   - On Windows, run the `help_end.bat` file using the fully qualified path for the DB2 Information Center:

     ```
     <DB2 Information Center dir>\doc\bin\help_end.bat
     ```

     Note: The help_end batch file contains the commands required to safely terminate the processes that were started with the help_start batch file. Do not use `Ctrl-C` or any other method to terminate `help_start.bat`.
   - On Linux, run the `help_end` script using the fully qualified path for the DB2 Information Center:

     ```
     <DB2 Information Center dir>/doc/bin/help_end
     ```

     Note: The help_end script contains the commands required to safely terminate the processes that were started with the help_start script. Do not use any other method to terminate the `help_start` script.
7. Restart the DB2 Information Center service.
   - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click on **DB2 Information Center** service and select **Start**.
   - On Linux, enter the following command:

     ```
     /etc/init.d/db2icdv9 start
     ```

The updated DB2 Information Center displays the new and updated topics.

Related concepts:
- "DB2 Information Center installation options" in *Quick Beginnings for DB2 Servers*

Related tasks:
- "Installing the DB2 Information Center using the DB2 Setup wizard (Linux)" in *Quick Beginnings for DB2 Servers*
- "Installing the DB2 Information Center using the DB2 Setup wizard (Windows)" in *Quick Beginnings for DB2 Servers*

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 products. Lessons provide step-by-step instructions.

Before you begin:

You can view the XHTML version of the tutorial from the Information Center at http://publib.boulder.ibm.com/infocenter/db2help/.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 tutorials:

To view the tutorial, click on the title.

Native XML data store
: Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

Visual Explain Tutorial
: Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

Related concepts:

- "Visual Explain overview" in *Administration Guide: Implementation*

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 products.

DB2 documentation
: Troubleshooting information can be found in the DB2 Troubleshooting Guide or the Support and Troubleshooting section of the DB2 Information Center. There you will find information on how to isolate and identify problems using DB2 diagnostic tools and utilities, solutions to some of the most common problems, and other advice on how to solve problems you might encounter with your DB2 products.

DB2 Technical Support Web site
: Refer to the DB2 Technical Support Web site if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

  Access the DB2 Technical Support Web site at http://www.ibm.com/software/data/db2/udb/support.html

Related concepts:

- "Introduction to problem determination" in *Troubleshooting Guide*
- "Overview of the DB2 technical information" on page 175

Terms and Conditions

Appendix B. Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product, and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Trademarks

Index

Contacting IBM

To contact IBM in your country or region, check the IBM Directory of Worldwide Contacts at http://www.ibm.com/planetwide

To learn more about DB2 products, go to http://www.ibm.com/software/data/db2/.

Printed in USA

SC10-4230-00

Spine information:

IBM DB2 **DB2 Version 9** **Developing ADO.NET and OLE DB Applications**