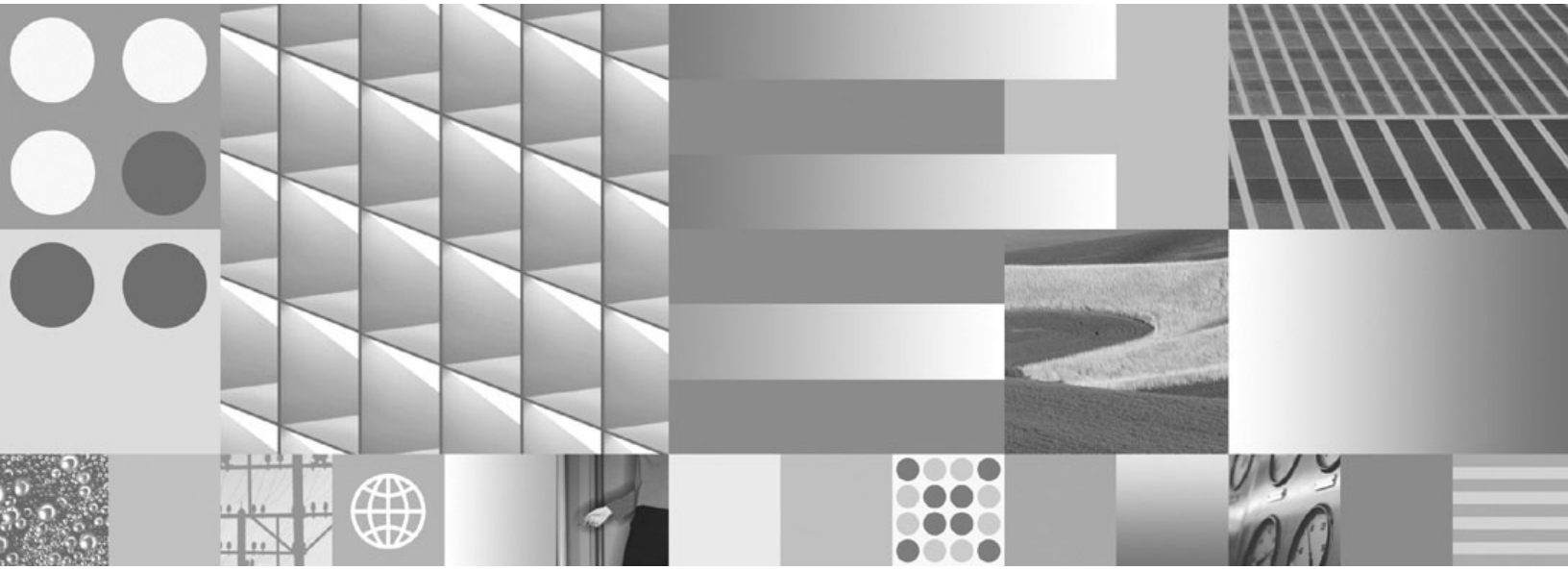
IBM DB2 9.7
for Linux, UNIX, and Windows

Version 9 Release 7

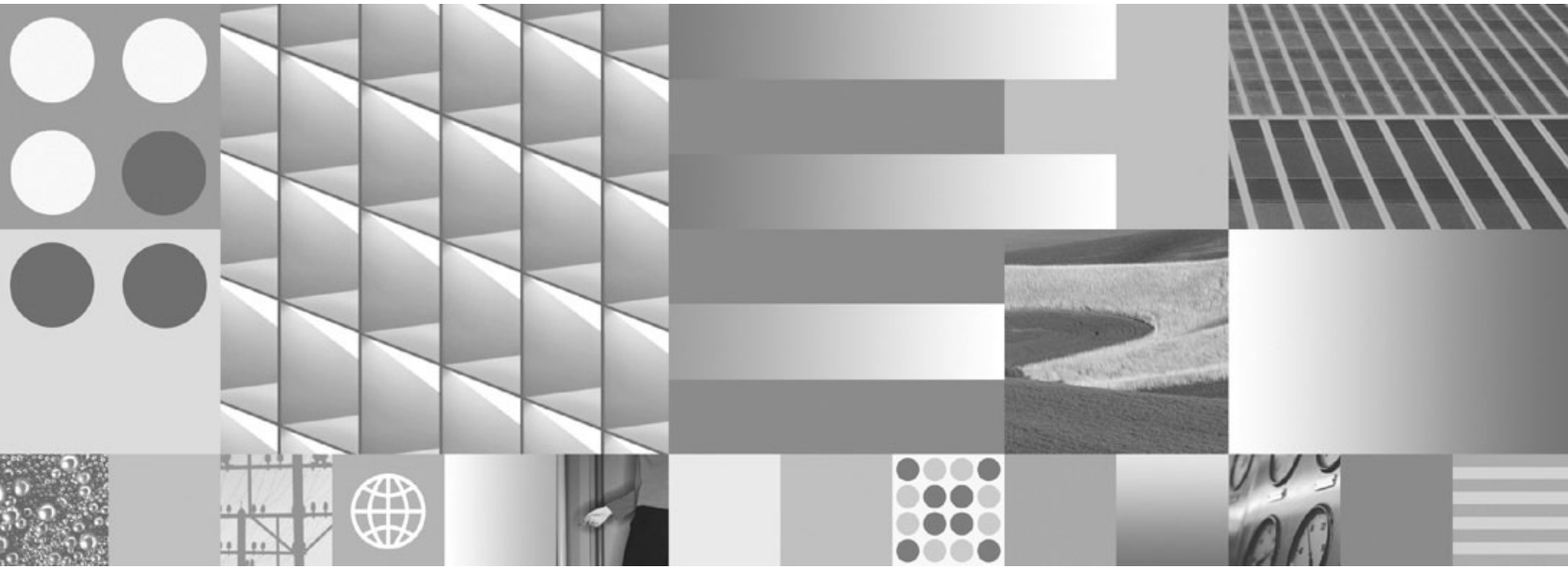


Developing ADO.NET and OLE DB Applications
Updated November, 2009

SC27-2444-01

IBM DB2 9.7
for Linux, UNIX, and Windows

Version 9 Release 7



Developing ADO.NET and OLE DB Applications
Updated November, 2009

SC27-2444-01

Note
Before using this information and the product it supports, read the general information under Appendix B, "Notices," on page 173.

Edition Notice

This document contains proprietary information of IBM. It is provided under a license agreement and is protected by copyright law. The information contained in this publication does not include any product warranties, and any statements provided in this manual should not be interpreted as such.

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1-800-IBM-4YOU (426-4968).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1. 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.

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

A .NET data provider features implementations of the following basic 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.

Microsoft provides two data providers, the OLE DB .NET Data Provider and ODBC .NET Data Provider. The OLE DB .NET Data Provider is a bridge provider that feeds ADO.NET requests to the IBM OLE DB Provider (by way of the COM interop module). ODBC .NET Data Provider is a bridge provider that feeds ADO.NET requests to the IBM ODBC Driver. These .NET data provider are not recommended for access to DB2 family databases. The IBM Data Server Provider for .NET is 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 IBM Data Server Provider for .NET has fewer restrictions, and provides significantly better performance than the OLE DB and ODBC .NET bridge providers.

Deploying .NET applications (Windows)

To simplify .NET application deployment, IBM® provides the IBM Data Server Driver Package, a small-footprint client that is ideal for use in mass deployment scenarios. You can use the IBM Data Server Runtime Client instead, if the additional features of that client are desired over the IBM Data Server Driver Package.

Prerequisites

- Before deployment, you must build your .NET application, which you can do with either Visual Studio or the command line. For more information about building .NET applications, see the related tasks.
- Computers that you use to build .NET applications and computers where you will deploy .NET applications must have a supported version of the Windows operating system, in addition to other software, as described in "Supported .NET development software":
  - Build systems
    - Windows operating system
    - Visual Studio
    - .NET Framework Redistributable Package
    - .NET Framework Software Development Kit
  - Deployment systems
    - Windows operating system
    - .NET Framework Redistributable Package

To deploy a .NET application:

1. Install the IBM Data Server Driver Package onto the computers where you will deploy your application. During the installation, set the IBM Data Server Driver Package installation to be the default database client interface copy.

   Note: Any existing database applications that run against an IBM data server will use this new installation of the IBM Data Server Driver Package. Test those applications against the new driver before rolling out your deployed .NET application.
2. Install your built application onto the computers where your application will run.

Supported .NET development software

To develop and deploy .NET applications that run against IBM data servers, you will need to use supported development software and operating systems.

Supported operating systems for developing and deploying .NET Framework 2.0, 3.0 and 3.5 applications

- Windows XP, Service Pack 2 (32-bit and 64-bit editions)
- Windows Server 2003 (32-bit and 64-bit editions)
- Windows Vista (32-bit and 64-bit editions)
- Windows Server 2008 (32-bit and 64-bit editions)

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 2005 (for .NET Framework 2.0 and 3.0 applications)
- Visual Studio 2008 (for .NET Framework 2.0, 3.0 and 3.5 applications)

Supported deployment software for .NET Framework applications

In addition to a DB2 runtime client , you will need one of the following three options to deploy .NET Framework applications. In most cases one of these are included with a Windows installation.

- .NET Framework Version 2.0 Redistributable Package (for .NET Framework 2.0 applications)
- .NET Framework Version 3.0 Redistributable Package (for .NET Framework 3.0 applications)
- .NET Framework Version 3.5 Redistributable Package (for .NET Framework 3.5 applications)

DB2 integration in Visual Studio

The IBM Database Add-Ins for Visual Studio 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.

IBM Database Add-Ins for Visual Studio 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 designers and 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 or later you can do this from the Server 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 2005 and 2008

The IBM Database Add-Ins for Visual Studio 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. If you do not have Visual Studio installed on your computer, the add-ins will not install. Once you install Visual Studio, you can then install the add-ins at any time from the DB2 product's setup menu.

For more details about using the IBM Database Add-Ins and the Data Server Provider for .NET for rapid application development, visit the IBM Information Management and Visual Studio .NET zone at http://www.ibm.com/developerworks/data/zones/vstudio/index.html.

Chapter 2. 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.

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.

External routine implementation

An external routine implementation is one in which the routine logic is defined by programming language code that resides external to the database. As with other routine implementations, routines with external implementations are created in the database by executing a CREATE statement. The routine logic stored in a compiled library resides on the database server in a special directory path. The association of the routine name with the external code application is asserted by the specification of the EXTERNAL clause in the CREATE statement.

External routines can be written in any of the supported external routine programming languages.

External routine implementation can be somewhat more complex than SQL routine implementation. However, they are extremely powerful because they allow you to harness the full functionality and performance of the chosen implementation programming language. External functions also have the advantage of being able to access and manipulate entities that reside outside of the database, such as the network or file system. For routines that require a smaller degree of interaction with the DB2 database, but that must contain a lot of logic or very complex logic, an external routine implementation is a good choice.

As an example, external routines are ideal to use to implement new functions that operate on and enhance the utility of built-in data types, such as a new string function that operate on a VARCHAR data type or a complicated mathematical function that operates on a DOUBLE data type. External routine implementations are also ideal for logic that might involve an external action, such as sending an email.

If you are already comfortable programming in one of the supported external routine programming languages, and need to encapsulate logic with a greater emphasis on programming logic than data access, once you learn the steps involved in creating routines with external implementation, you will soon discover just how powerful they can be.

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

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.	• 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 always run in the same memory as the database manager. This corresponds to the routine being created by default with the keywords NOT FENCED.	• 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 • Use of the default clause values minimizes the possibility of database manager interruption at runtime; however because CLR routines must run as FENCED, they might perform slightly more slowly than other external routines that can be specified as NOT FENCED.	• 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.

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.

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

Note:

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.

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.

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.

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.

The statements that follow the above syntax style can be rewritten using the FOR EACH ROW OF clause. The outcome of the query is the same as the old style. For example:

```
SELECT B.x,B.y, B.z,...
  TABLE(table_func_1(
    FOR EACH ROW OF (SELECT * FROM table_1 WHERE ...))) as B
```

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.

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.

As 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.

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 */
}
```

As 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.

SQL in external routines

All routines written in an external programming language (such as C, Visual Basic, C#, Java, and others) can contain SQL.

The CREATE statement for a routine (stored procedure, UDF), or the CREATE TYPE statement, in the case of a method, contains a clause that defines the level of SQL access for the routine or method. Based on the nature of the SQL included in your routine, you must choose the applicable clause:

NO SQL
: the routine contains no SQL at all

CONTAINS SQL
: Contains SQL, but neither reads nor writes data (for example: SET SPECIAL REGISTER).

READS SQL DATA
: Contains SQL that can read from tables (SELECT, VALUES statements), but does not modify table data.

MODIFIES SQL DATA
: Contains SQL that updates tables, either user tables directly (INSERT, UPDATE, DELETE statements) or DB2's catalog tables implicitly (DDL statements). This clause is only applicable to stored procedures and SQL-bodied table functions.

DB2 will validate at execution time that a routine does not exceed its defined level. For example, if a routine defined as CONTAINS SQL tries to SELECT from a table, an error (SQLCODE -579, SQLSTATE 38004) will result because it is attempting a read of SQL data. Also note that nested routine references, must be of the same or of a more restrictive SQL level that contains the reference. For example, routines

that modify SQL data can invoke routines that read SQL data, but routines that can only read SQL data, that are defined with the READS SQL DATA clause, cannot invoke routines that modify SQL data.

A routine executes SQL statements within the database connection scope of the calling application. A routine cannot establish its own connection, nor can it reset the calling application's connection (SQLCODE -751, SQLSTATE 38003).

Only a stored procedure defined as MODIFIES SQL DATA can issue COMMIT and ROLLBACK statements. Other types of routines (UDFs and methods) cannot issue COMMITs or ROLLBACKs (SQLCODE -751, SQLSTATE 38003). Even though a stored procedure defined as MODIFIES SQL DATA can attempt to COMMIT or ROLLBACK a transaction, it is recommended that a COMMIT or ROLLBACK be done from the calling application so changes are not unexpectedly committed. Stored procedures cannot issue COMMIT or ROLLBACK statements if the stored procedure was invoked from an application that established a type 2 connection to the database.

Also, only stored procedures defined as MODIFIES SQL DATA can establish their own savepoints, and rollback their own work within the savepoint. Other types of routines (UDFs and methods) cannot establish their own savepoints. A savepoint created within a stored procedure is not released when the stored procedure completes. The application will be able to roll back the savepoint. Similarly, a stored procedure could roll back a savepoint defined in the application. DB2 will implicitly release any savepoints established by the routine when it returns.

A routine can inform DB2 about whether it has succeeded by assigning an SQLSTATE value to the sqlstate argument that DB2 passes to it. Some parameter styles (PARAMETER STYLEs JAVA, GENERAL, and GENERAL WITH NULLS) do not support the exchange of SQLSTATE values.

If, in handling the SQL issued by a routine, DB2 encounters an error, it returns that error to the routine, just as it does for any application. For normal user errors, the routine has an opportunity to take alternative or corrective action. For example, if a routine is trying to INSERT to a table and gets a duplicate key error (SQLCODE -813), it can instead UPDATE the existing row of the table.

There are, however, certain more serious errors that can occur that make it impossible for DB2 to proceed in a normal fashion. Examples of these include deadlock, or database partition failure, or user interrupt. Some of these errors are propagated up to the calling application. Other severe errors that are unit of work related go all the way out to either (a) the application, or (b) a stored procedure that is permitted to issue transaction control statements (COMMIT or ROLLBACK), whichever occurs first in backing out.

If one of these errors occurs during the execution of SQL issued by a routine, the error is returned to the routine, but DB2 remembers that a serious error has occurred. Additionally, in this case, DB2 will automatically fail (SQLCODE -20139, SQLSTATE 51038) any subsequent SQL issued by this routine and by any calling routines. The only exception to this is if the error only backs out to the outermost stored procedure that is permitted to issue transaction control statements. In this case, this stored procedure can continue to issue SQL.

Routines can issue both static and dynamic SQL, and in either case they must be precompiled and bound if embedded SQL is used. For static SQL, the information used in the precompile/bind process is the same as it is for any client application

using embedded SQL. For dynamic SQL, you can use the DYNAMICRULES precompile/bind option to control the current schema and current authentication ID for embedded dynamic SQL. This behavior is different for routines and applications.

The isolation level defined for the routine packages or statements is respected. This can result in a routine running at a more restrictive, or a more generous, isolation level than the calling application. This is important to consider when calling a routine that has a less restrictive isolation level than the calling statement. For example, if a cursor stability function is called from a repeatable read application, the UDF can exhibit non-repeatable read characteristics.

The invoking application or routine is not affected by any changes made by the routine to special register values. Updatable special registers are inherited by the routine from the invoker. Changes to updatable special registers are not passed back to the invoker. Non-updatable special registers get their default value. For further details on updatable and non-updatable special registers, see the related topic, "Special registers".

Routines can OPEN, FETCH, and CLOSE cursors in the same manner as client applications. Multiple invocations (for example, in the case of recursion) of the same function each get their own instance of the cursor. UDFs and methods must close their cursors before the invoking statement completes, otherwise an error will occur (SQLCODE -472, SQLSTATE 24517). The final call for a UDF or method is a good time to close any cursors that remain open. Any opened cursors not closed before completion in a stored procedure are returned to the client application or calling routine as result sets.

Arguments passed to routines are not automatically treated as host variables. This means for a routine to use a parameter as a host variable in its SQL, it must declare its own host variable and copy the parameter value to this host variable.

Note: Embedded SQL routines must be precompiled and bound with the DATETIME option set to ISO.

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.

Table 2. Parameter styles (continued)

Parameter style	Supported language	Supported routine type	Description
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.
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 for z/OS®.
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 for z/OS.

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.

Performance considerations for developing routines

One of the most significant benefits of developing routines, instead of expanding client applications, is performance. Consider the following performance impacts when choosing an approach for routine implementation.

NOT FENCED mode

A NOT FENCED routine runs in the same process as the database manager. In general, running your routine as NOT FENCED results in better performance as compared with running it in FENCED mode, because FENCED routines run in a special DB2 process outside of the engine's address space.

While you can expect improved routine performance when running routines in NOT FENCED mode, user code can accidentally or maliciously corrupt the database or damage the database control structures. You should only use NOT FENCED routines when you need to maximize the performance benefits, and if you deem the routine to be secure. (For information on assessing and mitigating the risks of registering C/C++ routines as NOT FENCED, see the topic, "Security considerations for routines".) If the routine is not safe enough to run in the database

manager's process, use the FENCED clause when registering the routine. To limit the creation and running of potentially unsafe code, DB2 requires that a user have a special privilege, CREATE_NOT_FENCED_ROUTINE in order to create NOT FENCED routines.

If an abnormal termination occurs while you are running a NOT FENCED routine, the database manager will attempt an appropriate recovery if the routine is registered as NO SQL. However, for routines not defined as NO SQL, the database manager will fail.

NOT FENCED routines must be precompiled with the WCHARTYPE NOCONVERT option if the routine uses GRAPHIC or DBCLOB data.

FENCED THREADSAFE mode

FENCED THREADSAFE routines run in the same process as other routines. More specifically, non-Java routines share one process, while Java routines share another process, separate from routines written in other languages. This separation protects Java routines from the potentially more error prone routines written in other languages. Also, the process for Java routines contains a JVM, which incurs a high memory cost and is not used by other routine types. Multiple invocations of FENCED THREADSAFE routines share resources, and therefore incur less system overhead than FENCED NOT THREADSAFE routines, which each run in their own dedicated process.

If you feel your routine is safe enough to run in the same process as other routines, use the THREADSAFE clause when registering it. As with NOT FENCED routines, information on assessing and mitigating the risks of registering C/C++ routines as FENCED THREADSAFE is in the topic, "Security considerations for routines".

If a FENCED THREADSAFE routine abends, only the thread running this routine is terminated. Other routines in the process continue running. However, the failure that caused this thread to abend can adversely affect other routine threads in the process, causing them to trap, hang, or have damaged data. After one thread abends, the process is no longer used for new routine invocations. Once all the active users complete their jobs in this process, it is terminated.

When you register Java routines, they are deemed THREADSAFE unless you indicate otherwise. All other LANGUAGE types are NOT THREADSAFE by default. Routines using LANGUAGE OLE and OLE DB cannot be specified as THREADSAFE.

NOT FENCED routines must be THREADSAFE. It is not possible to register a routine as NOT FENCED NOT THREADSAFE (SQLCODE -104).

Users on UNIX® can see their Java and C THREADSAFE processes by looking for `db2fmp (Java)` or `db2fmp (C)`.

FENCED NOT THREADSAFE mode

FENCED NOT THREADSAFE routines each run in their own dedicated process. If you are running numerous routines, this can have a detrimental effect on database system performance. If the routine is not safe enough to run in the same process as other routines, use the NOT THREADSAFE clause when registering the routine.

On UNIX, NOT THREADSAFE processes appear as `db2fmp (pid)` (where pid is the process id of the agent using the fenced mode process) or as `db2fmp (idle)` for a pooled NOT THREADSAFE db2fmp.

Java routines

If you intend to run a Java routine with large memory requirements, it is recommended that you register it as FENCED NOT THREADSAFE. For FENCED THREADSAFE Java routine invocations, DB2 attempts to choose a threaded Java fenced mode process with a Java heap that is large enough to run the routine. Failure to isolate large heap consumers in their own process can result in-out-of-Java-heap errors in multithreaded Java `db2fmp` processes. If your Java routine does not fall into this category, FENCED routines will run better in threadsafe mode where they can share a small number of JVMs.

NOT FENCED Java routines are currently not supported. A Java routine defined as NOT FENCED will be invoked as if it had been defined as FENCED THREADSAFE.

C/C++ routines

C or C++ routines are generally faster than Java routines, but are more prone to errors, memory corruption, and crashing. For these reasons, the ability to perform memory operations makes C or C++ routines risky candidates for THREADSAFE or NOT FENCED mode registration. These risks can be mitigated by adhering to programming practices for secure routines (see the topic, ″Security considerations for routines″), and thoroughly testing your routine.

SQL routines

SQL routines, particularly SQL procedures, are also generally faster than Java routines, and usually share comparable performance with C routines. SQL routines always run in NOT FENCED mode, providing a further performance benefit over external routines. UDFs that contain complex logic will generally run more quickly if written in C than in SQL. If the logic is simple, than an SQL UDF will be comparable to any external UDF.

Scratchpads

A scratchpad is a block of memory that can be assigned to UDFs and methods. The scratchpad only applies to the individual reference to the routine in an SQL statement. If there are multiple references to a routine in a statement, each reference has its own scratchpad. A scratchpad enables a UDF or method to save its state from one invocation to the next.

For UDFs and methods with complex initializations, you can use scratchpads to store any values required in the first invocation for use in all future invocations. The logic of other UDFs and methods might also require that intermediate values be saved from invocation to invocation.

Use VARCHAR parameters instead of CHAR parameters

You can improve the performance of your routines by using VARCHAR parameters instead of CHAR parameters in the routine definition. Using VARCHAR data types instead of CHAR data types prevents DB2 from padding parameters with spaces before passing the parameter and decreases the amount of time required to transmit the parameter across a network.

For example, if your client application passes the string ″A SHORT STRING″ to a routine that expects a CHAR(200) parameter, DB2 has to pad the parameter with 186 spaces, null-terminate the string, then send the entire 200 character string and null-terminator across the network to the routine.

In comparison, passing the same string, "A SHORT STRING," to a routine that expects a VARCHAR(200) parameter results in DB2 simply passing the 14 character string and a null terminator across the network.

Security considerations for routines

Developing and deploying routines provides you with an opportunity to greatly improve the performance and effectiveness of your database applications. There can, however, be security risks if the deployment of routines is not managed correctly by the database administrator. The following sections describe security risks and means by which you can mitigate these risks. The security risks are followed by a section on how to safely deploy routines whose security is unknown.

Security risks

NOT FENCED routines can access database manager resources

NOT FENCED routines run in the same process as the database manager. Because of their close proximity to the database engine, NOT FENCED routines can accidentally or maliciously corrupt the database manager's shared memory, or damage the database control structures. Either form of damage will cause the database manager to fail. NOT FENCED routines can also corrupt databases and their tables.

To ensure the integrity of the database manager and its databases, you must thoroughly screen routines you intend to register as NOT FENCED. These routines must be fully tested, debugged, and exhibit no unexpected side-effects. In the examination of the routine, pay close attention to memory management and the use of static variables. The greatest potential for corruption arises when code does not properly manage memory or incorrectly uses static variables. These problems are prevalent in languages other than Java and .NET programming languages.

In order to register a NOT FENCED routine, the CREATE_NOT_FENCED_ROUTINE authority is required. When granting the CREATE_NOT_FENCED_ROUTINE authority, be aware that the recipient can potentially gain unrestricted access to the database manager and all its resources.

Note: NOT FENCED routines are not supported in Common Criteria compliant configurations.

FENCED THREADSAFE routines can access memory in other FENCED THREADSAFE routines

FENCED THREADSAFE routines run as threads inside a shared process. Each of these routines are able to read the memory used by other routine threads in the same process. Therefore, it is possible for one threaded routine to collect sensitive data from other routines in the threaded process. Another risk inherent in the sharing of a single process, is that one routine thread with flawed memory management can corrupt other routine threads, or cause the entire threaded process to crash.

To ensure the integrity of other FENCED THREADSAFE routines, you must thoroughly screen routines you intend to register as FENCED THREADSAFE. These routines must be fully tested, debugged, and exhibit no unexpected side-effects. In the examination of the routine, pay close attention to memory management and the use of static variables. This is where the greatest potential for corruption exists, particularly in languages other than Java.

In order to register a FENCED THREADSAFE routine, the CREATE_EXTERNAL_ROUTINE authority is required. When granting the CREATE_EXTERNAL_ROUTINE authority, be aware that the recipient can potentially monitor or corrupt the memory of other FENCED THREADSAFE routines.

Write access to the database server by the owner of fenced processes can result in database manager corruption

The user ID under which fenced processes run is defined by the db2icrt (create instance) or db2iupdt (update instance) system commands. This user ID must not have write access to the directory where routine libraries and classes are stored (in UNIX environments, sqllib/function; in Windows environments, sqllib\function). This user ID must also not have read or write access to any database, operating system, or otherwise critical files and directories on the database server.

If the owner of fenced processes does have write access to various critical resources on the database server, the potential for system corruption exists. For example, a database administrator registers a routine received from an unknown source as FENCED NOT THREADSAFE, thinking that any potential harm can be averted by isolating the routine in its own process. However, the user ID that owns fenced processes has write access to the sqllib/function directory. Users invoke this routine, and unbeknownst to them, it overwrites a library in sqllib/function with an alternate version of a routine body that is registered as NOT FENCED. This second routine has unrestricted access to the entire database manager, and can thereby distribute sensitive information from database tables, corrupt the databases, collect authentication information, or crash the database manager.

Ensure the user ID that owns fenced processes does not have write access to critical files or directories on the database server (especially sqllib/function and the database data directories).

Vulnerability of routine libraries and classes

If access to the directory where routine libraries and classes are stored is not controlled, routine libraries and classes can be deleted or overwritten. As discussed in the previous item, the replacement of a NOT FENCED routine body with a malicious (or poorly coded) routine can severely compromise the stability, integrity, and privacy of the database server and its resources.

To protect the integrity of routines, you must manage access to the directory containing the routine libraries and classes. Ensure that the fewest possible number of users can access this directory and its files. When assigning write access to this directory, be aware that this privilege can provide the owner of the user ID unrestricted access to the database manager and all its resources.

Deploying potentially insecure routines

If you happen to acquire a routine from an unknown source, be sure you know exactly what it does before you build, register, and invoke it. It is recommended that you register it as FENCED and NOT THREADSAFE unless you have tested it thoroughly, and it exhibits no unexpected side-effects.

If you need to deploy a routine that does not meet the criteria for secure routines, register the routine as FENCED and NOT THREADSAFE. To ensure that database integrity is maintained, FENCED and NOT THREADSAFE routines:

- Run in a separate DB2 process, shared with no other routines. If they abnormally terminate, the database manager will be unaffected.
- Use memory that is distinct from memory used by the database. An inadvertent mistake in a value assignment will not affect the database manager.

Routine code page considerations

Character data is passed to external routines in the code page implied by the PARAMETER CCSID option used when the routine was created. Similarly, a character string that is output from the routine is assumed by the database to use the code page implied by the PARAMETER CCSID option.

When a client program (using, for example, code page C) accesses a section with a different code page (for example, code page S) that invokes a routine using a different code page (for example, code page R), the following events occur:

1. When an SQL statement is invoked, input character data is converted from the code page of the client application (C) to the one associated with the section (S). Conversion does not occur for BLOBs or data that will be used as FOR BIT DATA.
2. If the code page of the routine is not the same as the code page of the section, then before the routine is invoked, input character data (except for BLOB and FOR BIT DATA) is converted to the code page of the routine (R).

   It is strongly recommended that you precompile, compile, and bind the server routine using the code page that the routine will be invoked under (R). This might not be possible in all cases. For example, you can create a Unicode database in a Windows environment. However, if the Windows environment does not have the Unicode code page, you have to precompile, compile, and bind the application that creates the routine in a Windows code page. The routine will work if the application has no special delimiter characters that the precompiler does not understand.
3. When the routine finishes, the database manager converts all output character data from the routine code page (R) to the section code page (S) if necessary. If the routine raised an error during its execution, the SQLSTATE and diagnostic message from the routine will also be converted from the routine code page to the section code page. Conversion does not happen for BLOB or FOR BIT DATA character strings.
4. When the statement finishes, output character data is converted from the section code page (S) back to code page of the client application (C). Conversion does not occur for BLOBS or for data that was used as FOR BIT DATA.

By using the DBINFO option on the CREATE FUNCTION, CREATE PROCEDURE, and CREATE TYPE statements, the routine code page is passed to the routine. Using this information, a routine that is sensitive to the code page can be written to operate in many different code pages.

32-bit and 64-bit application and routine support

DB2 Database for Linux®, UNIX, and Windows provides support for the development and deployment of applications and routines, including procedures and user-defined functions (UDFs), on various platforms. For your applications

and routines to work properly it is important to review and understand DB2 database support for 32-bit and 64-bit considerations.

To start off, it is good to first clarify a few points:

- A 32-bit hardware platform is running a 32-bit operating system, and that a 64 bit hardware platform is running a 64-bit operating system.
- You can install a 32-bit instance of the DB2 database on either a 32-bit operating system or a 64 bit operating system, but that you can only install a 64-bit instance of DB2 database on a 64-bit operating system.
- A 32-bit application is an application that was built on a 32 bit operating system.
- A 64-bit application is an application that was built on a 64-bit operating system.

The following tables outline DB2 database 32-bit and 64-bit support for client applications and routines, with the following presumptions:

Table 3. Support for running 32-bit and 64-bit applications on 32-bit or 64-bit hardware platforms

	32 bit hardware + operating system	**64 bit hardware + operating system**
32 bit application	YES	YES
64 bit application	NO	YES

The following table indicates the support for creating a connection to a DB2 database server from a DB2 client application.

Table 4. Support for connections from 32-bit and 64-bit clients to 32-bit and 64-bit servers

	32 bit server	**64 bit server**
32 bit client	YES	YES
64 bit application	YES	YES

Table 5. Support for running 32-bit and 64-bit applications on 32-bit or 64-bit hardware platforms

	32 bit hardware and operating system	**64 bit hardware and operating system**
32 bit application	YES	YES
64 bit application	NO	YES

The following table indicates the support for creating a connection to a DB2 database server from a DB2 client application.

Table 6. Support for connections from 32-bit and 64-bit clients to 32-bit and 64-bit servers

	32 bit server	**64 bit server**
32 bit client	YES	YES
64 bit application	YES	YES

Table 7. Support for running fenced and unfenced procedures and UDFs on 32-bit and 64-bit servers

	32 bit server	**64 bit server**
32 bit fenced procedure or UDF	YES	YES[1,2,3]
64 bit fenced procedure or UDF	NO	YES
32 bit unfenced procedure or UDF	YES	NO[2]
64 bit unfenced procedure or UDF	NO	YES

Note:

1. Running a 32-bit procedure on a 64-bit server can be slow.
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 a Linux/IA-64 database server.

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 8. 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 [1]
64-bit fenced procedure or UDF	Not supported [3]	Supported
32-bit unfenced procedure or UDF	Supported	Supported [1,2]
64-bit unfenced procedure or UDF	Not supported [3]	Supported

Note:

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. 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.

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

It is possible to call routines with 32-bit routine libraries on 64-bit DB2 database servers. However, this does not perform as well as calling 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 call it as a 64-bit library. If this fails, the library is then called as a 32-bit library. A failed attempt to call a 32-bit library as a 64-bit library produces an error message (SQLCODE -444) in the db2diag log files.

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.

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 data is materialized when passed to stored procedures as IN, OUT, or INOUT parameters. If you are using Java stored procedures, the heap size (JAVA_HEAP_SZ configuration parameter) might need to be increased based on the quantity and size of XML arguments, and the number of external stored procedures that are being executed concurrently.

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

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 unexpected external 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. This ensures that allocated memory is freed before the routine returns.

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

- The IBM Data Server 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.

For a list of restrictions associated with external routines see:

- "Restrictions on external routines" on page 35

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 statements.
   - 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 24
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

Writing routines

The three types of routines (procedures, UDFs, and methods) have much in common with regards to how they are written. For instance, the three routine types employ some of the same parameter styles, support the use of SQL through various client interfaces (embedded SQL, CLI, and JDBC), and can all invoke other routines. To this end, the following steps represent a single approach for writing routines.

There are some routine features that are specific to a routine type. For example, result sets are specific to stored procedures, and scratchpads are specific to UDFs and methods. When you come across a step not applicable to the type of routine you are developing, go to the step that follows it.

Before writing a routine, you must decide the following:

- The type of routine you need.
- The programming language you will use to write it.
- Which interface to use if you require SQL statements in your routine.

See also the topics on Security, Library and Class Management, and Performance considerations.

To create a routine body, you must:

1. *Applicable only to external routines.* Accept input parameters from the invoking application or routine and declare output parameters. How a routine accepts parameters is dependent on the parameter style you will create the routine with. Each parameter style defines the set of parameters that are passed to the routine body and the order that the parameters are passed.

   For example, the following is a signature of a UDF body written in C (using `sqludf.h`) for PARAMETER STYLE SQL:

   ```
   SQL_API_RC SQL_API_FN product ( SQLUDF_DOUBLE *in1,
                                   SQLUDF_DOUBLE *in2,
                                   SQLUDF_DOUBLE *outProduct,
                                   SQLUDF_NULLIND *in1NullInd,
                                   SQLUDF_NULLIND *in2NullInd,
                                   SQLUDF_NULLIND *productNullInd,
                                   SQLUDF_TRAIL_ARGS )
   ```

2. Add the logic that the routine is to perform. Some features that you can employ in the body of your routines are as follows:
   - Calling other routines (nesting), or calling the current routine (recursion).
   - In routines that are defined to have SQL (CONTAINS SQL, READS SQL, or MODIFIES SQL), the routine can issue SQL statements. The types of statements that can be invoked is controlled by how routines are registered.
   - In external UDFs and methods, use scratchpads to save state from one call to the next.
   - In SQL procedures, use condition handlers to determine the SQL procedure's behavior when a specified condition occurs. You can define conditions based on SQLSTATEs.
3. *Applicable only to stored procedures.* Return one or more result sets. In addition to individual parameters that are exchanged with the calling application, stored procedures have the capability to return multiple result sets. Only SQL routines and CLI, ODBC, JDBC, and SQLJ routines and clients can accept result sets.

In addition to writing your routine, you also need to register it before you can invoke it. This is done with the CREATE statement that matches the type of routine you are developing. In general, the order in which you write and register your routine does not matter. However, the registration of a routine must precede its being built if it issues SQL that references itself. In this case, for a bind to be successful, the routine's registration must have already occurred.

Debugging routines

Before deploying routines on a production server you must thoroughly test and debug them on a test server. This is especially important for routines that need to be registered as NOT FENCED because they have unrestricted access to the database manager's memory, its databases, and database control structures. FENCED THREADSAFE routines also demand close attention because they share memory with other routines.

Checklist of common routine problems

To ensure that a routine executes properly, check that:

- The routine is registered properly. The parameters provided in the CREATE statement must match the arguments handled by the routine body. With this in mind, check the following specific items:
  - The data types of the arguments used by the routine body are appropriate for the parameter types defined in the CREATE statement.
  - The routine does not write more bytes to an output variable than were defined for the corresponding result in the CREATE statement.
  - The routine arguments for SCRATCHPAD, FINAL CALL, DBINFO are present if the routine was registered with corresponding CREATE options.
  - For external routines, the value for the EXTERNAL NAME clause in the CREATE statement must match the routine library and entry point (case sensitivity varies by operating system).
  - For C++ routines, the C++ compiler applies type decoration to the entry point name. Either the type decorated name needs to be specified in the EXTERNAL NAME clause, or the entry point should be defined as `extern "C"` in the user code.
  - The routine name specified during invocation must match the registered name (defined in the CREATE statement) of the routine. By default, routine identifiers are folded to uppercase. This does not apply to delimited identifiers, which are not folded to uppercase, and are therefore case sensitive.

    The routine must be placed in the directory path specified in the CREATE statement, or if no path is given, where DB2 looks for it by default. For UDFs, methods, and fenced procedures, this is: `sqllib/function` (UNIX) or `sqllib\function` (Windows). For unfenced procedures, this is: `sqllib/function/unfenced` (UNIX) or `sqllib\function\unfenced` (Windows).
- The routine is built using the correct calling sequence, precompile (if embedded SQL), compile, and link options.
- The application is bound to the database, except if it is written using DB2 CLI, ODBC, or JDBC. The routine must also be bound if it contains SQL and does not use any of these interfaces.
- The routine accurately returns any error information to the client application.

- All applicable call types are accounted for if the routine was defined with FINAL CALL.
- The system resources used by routines are returned.
- If you attempt to invoke a routine and receive an error (SQLCODE -551, SQLSTATE 42501) indicating that you have insufficient privileges to perform this operation, this is likely because you do not have the EXECUTE privilege on the routine. This privilege can be granted to any invoker of a routine by a user with SECADM authority, ACCESSCTRL authority, or by any user with EXECUTE WITH GRANT OPTION privilege on the routine. The related topic on authorizations and routines provides details on how to effectively manage the use of this privilege.

Routine debugging techniques

To debug a routine, use the following techniques:

- The Development Center provides extensive debugging tools for SQL-bodied and Java procedures.
- It is not possible to write diagnostic data to screen from a routine. If you intend to write diagnostic data to a file, ensure that you write to a globally accessible directory such as `\tmp`. Do not write to directories used by database managers or databases.

  For procedures, a safe alternative is to write diagnostic data to an SQL table. The procedure you are testing must be registered with the MODIFIES SQL DATA clause in order to be able to write to an SQL table. If you need an existing procedure to write data (or no longer write data) to an SQL table, you must drop and re-register the procedure with (or without) the MODIFIES SQL DATA clause. Before dropping and re-registering the procedure, be aware of its dependencies.
- You can debug your routine locally by writing a simple application that invokes the routine entry point directly. Consult your compiler documentation for information on using the supplied debugger.

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 library 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.

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.

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.

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.

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.

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.

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 library 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 reload the routine's library.

Chapter 3. .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 5
- "SQL data type representation in .NET CLR routines" on page 47
- "Parameters in .NET CLR routines" on page 49
- "Returning result sets from .NET CLR procedures" on page 51
- "Restrictions on .NET CLR routines" on page 53
- "Errors related to .NET CLR routines" on page 63

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 Window" on page 55
- "Examples of C# .NET CLR procedures" on page 65
- "Examples of C# .NET CLR functions" on page 99

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.

.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#.

You can develop .NET CLR routines on the following operating systems:

- Windows 2000
- Windows XP (32–bit edition and 64–bit edition)
- Windows Server 2003 (32–bit and 64–bit edition)
- Windows Server 2008 (32–bit and 64–bit edition)

A Version 9 or later data server client must be installed for .NET CLR routine development. The database server must be running DB2 Version 9 or later database products.

A supported version of the Microsoft .NET Framework software must also be installed on the same computer as the DB2 database server. The Microsoft .NET Framework is independently available or available as part of Microsoft .NET Framework Version Software Development Kit.

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 Window

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.

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 routine implementation” on page 6

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

- Chapter 3, “.NET common language runtime (CLR) routines,” on page 45

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, see: "Creating .NET CLR routines" on page 54.

SQL data type representation in .NET CLR routines

.NET CLR routines can reference SQL data type values as routine parameters, parameter values to be used as part of SQL statement execution, and as variables, however the appropriate IBM SQL data type values, IBM Data Server Provider for .NET data type values, and .NET Framework data type values must be used to ensure that there is no truncation or loss of data when accessing or retrieving the values.

For routine parameter specifications within the CREATE PROCEDURE or CREATE FUNCTION statements used to create .NET CLR routines, DB2 SQL data type values are used. Most SQL data types can be specified for routine parameters, however there are some exceptions.

For specifying parameter values to be used as part of an SQL statement to be executed, IBM Data Server Provider for .NET objects must be used. The `DB2Parameter` object is used to represent a parameter to be added to a `DB2Command` object which represents a SQL statement. When specifying the data type value for the parameter, the IBM Data Server Provider for .NET data type values available in the `IBM.Data.DB2Types` namespace must be used. The `IBM.Data.DB2Types` namespace provides classes and structures to represent each of the supported DB2 SQL data types.

For parameters and local variables that might temporarily hold SQL data type values appropriate IBM Data Server Provider for .NET data types, as defined in the IBM.Data.DB2Types Namespace, must be used.

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

The following table shows mappings between DB2Type data types, DB2 data types, Informix® data types, Microsoft .NET Framework types, and DB2Types classes and structures.

Category	DB2Types Classes and Structures	DB2Type Data Type	DB2 Data Type	Informix Data Type	.NET Data Type
Numeric	DB2Int16	SmallInt	SMALLINT	BOOLEAN, SMALLINT	Int16
	DB2Int32	Integer	INT	INTEGER, INT, SERIAL	Int32
	DB2Int64	BigInt	BIGINT	BIGINT, BIGSERIAL, INT8, SERIAL8	Int64
	DB2Real, DB2Real370	Real	REAL	REAL, SMALLFLOAT	Single
	DB2Double	Double	DOUBLE PRECISION	DECIMAL (≤31), DOUBLE PRECISION	Double
	DB2Double	Float	FLOAT	DECIMAL (32), FLOAT	Double
	DB2Decimal	Decimal	DECIMAL	MONEY	Decimal
	DB2DecimalFloat	DecimalFloat	DECFLOAT (16|34)[1,4]		Decimal
	DB2Decimal	Numeric	DECIMAL	DECIMAL (≤31), NUMERIC	Decimal
Date/Time	DB2Date	Date	DATE	DATETIME (date precision)	Datetime
	DB2Time	Time	TIME	DATETIME (time precision)	TimeSpan
	DB2TimeStamp	Timestamp	TIMESTAMP	DATETIME (time and date precision)	DateTime
XML	DB2Xml	Xml[2]	XML		Byte[]
Character data	DB2String	Char	CHAR	CHAR	String
	DB2String	VarChar	VARCHAR	VARCHAR	String
	DB2String	LongVarChar[1]	LONG VARCHAR	LVARCHAR	String
Binary data	DB2Binary	Binary	CHAR FOR BIT DATA		Byte[]
	DB2Binary	Binary[3]	BINARY		Byte[]
	DB2Binary	VarBinary[3]	VARBINARY		Byte[]
	DB2Binary	LongVarBinary[1]	LONG VARCHAR FOR BIT DATA		Byte[]
Graphic data	DB2String	Graphic	GRAPHIC		String
	DB2String	VarGraphic	VARGRAPHIC		String
	DB2String	LongVarGraphic[1]	LONG VARGRAPHIC		String
LOB data	DB2Clob	Clob	CLOB	CLOB, TEXT	String
	DB2Blob	Blob	BLOB	BLOB, BYTE	Byte[]
	DB2Clob	DbClob	DBCLOB		String
Row ID	DB2RowId	RowId	ROWID		Byte[]

1. These data types are not supported as parameters in DB2 .NET common language runtime routines.
2. A DB2ParameterClass.ParameterName property of the type DB2Type.Xml can accept variables of the following types: String, byte[], DB2Xml, and XmlReader.
3. These data types are applicable only to DB2 for z/OS.
4. This data type is only supported for DB2 for z/OS, Version 9 and later releases and for DB2 for Linux, UNIX, and Windows Version 9.5 and later releases.

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:

- "External routine parameter styles" on page 24

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 keyword 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:

- "External routine parameter styles" on page 24

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 CLR procedures

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)
```

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.

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

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.

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 web page (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.

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 EXECUTION 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
- 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.

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 Chapter 3, ".NET common language runtime (CLR) routines," on page 45.
- 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.

For a list of restrictions associated with CLR routines see:

- "Restrictions on .NET CLR routines" on page 53

Create .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 Window

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

- "Creating .NET CLR routines from DB2 Command Window"

Creating .NET CLR routines from DB2 Command Window

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:
  - Chapter 3, ".NET common language runtime (CLR) routines," on page 45
- The database server must be running a Windows operating system that supports the Microsoft .NET Framework.
- A supported version of the Microsoft .NET Framework software must be installed on the server. The .NET Framework is independently available or as part of the Microsoft .NET Framework Software Development Kit.
- A supported DB2 database product or IBM Data Server Client must be installed. See "Installation requirements for DB2 database products" in *Installing DB2 Servers*.

- 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.

For a list of restrictions associated with CLR routines see:

- "Restrictions on .NET CLR routines" on page 53

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 data types:
     – "SQL data type representation in .NET CLR routines" on page 47
   - 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 49
   - 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 51
   - 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 following topic:
   - "Building common language runtime (CLR) .NET routines" in *Developing ADO.NET and OLE DB Applications*
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.

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 Window as well.

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

Building .NET common language 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 deployment of 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 Window

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. As routines must be built on the same instance where the database resides, no arguments are required for a user ID and password.

Prerequisites

- The required .NET CLR routine operating system and development software prerequisites 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 assembly. 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 Window

Building .NET CLR routine source code is a sub-task of creating .NET CLR routines. This task can be done manually from DB2 Command Window. 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.

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 dynamic link library as a stored procedure assembly dll. **/debug** Use the debugger. **/lib: "%DB2PATH%"\bin\netf20** Use the library path for .NET Framework Version 2.0. There are several supported versions of the .NET framework for applications: version 2.0, version 3.0, and version 3.5. There is a dynamic link library for each. For .NET Framework Version 1.1, use the `"%DB2PATH%"\bin\netf11` sub-directory. For .NET Framework Version 2.0, 3.0, and 3.5, use the `"%DB2PATH%"\bin\netf20` sub-directory. **/reference:IBM.Data.DB2.dll** Use the DB2 dynamic link library for the IBM Data Server Provider for .NET 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 dynamic link library as a stored procedure assembly dll. **/debug** Use the debugger. **/libpath:"%DB2PATH%"\bin\netf20** Use the library path for .NET Framework Version 2.0. There are several supported versions of the .NET framework for applications: version 2.0, version 3.0, and version 3.5. There is a dynamic link library for each. For .NET Framework Version 1.1, use the `"%DB2PATH%"\bin\netf11` sub-directory. For .NET Framework Version 2.0, 3.0, and 3.5, use the `"%DB2PATH%"\bin\netf20` sub-directory. **/reference:IBM.Data.DB2.dll** Use the DB2 dynamic link library for the IBM Data Server Provider for .NET. **/reference:System.dll** Reference the Microsoft Windows System dynamic link library. **/reference:System.Data.dll** Reference the Microsoft Windows System Data dynamic link library. Refer to your compiler documentation for additional compiler options.

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.

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 any user with SECADM authority, ACCESSCTRL authority, or by any user with EXECUTE WITH GRANT OPTION privilege on 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:

- "Errors related to .NET CLR routines" on page 63

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 diagnostic log files.

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 occurred will be written to the db2diag log files.

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 files specifically indicating that the exception occurred due to an execution control violation. The exception stack traceback that indicates where the violation occurred will also be included.

Determine if the assembly of the routine has been compromised or recently modified. If the routine has been validly modified, this problem can be occurring 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 52

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

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 99

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

- Chapter 3, ".NET common language runtime (CLR) routines," on page 45
- "Creating .NET CLR routines from DB2 Command Window" on page 55
- "Building common language runtime (CLR) .NET routines" in *Developing ADO.NET and OLE DB Applications*

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

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

- Example 2: C# parameter style GENERAL WITH NULLS procedure
- Example 3: C# parameter style SQL procedure
- Example 4: C# procedure returning a result set
- Example 5: C# procedure accessing the dbinfo structure
- Example 6: C# procedure in PROGRAM TYPE MAIN style

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 9. 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 10. 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' ;
```

```
 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 11. 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 11. 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 12. 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 12. 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 13. 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 14. 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 sqlState,
                                     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 occurred"
   //
   //  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 15. 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 15. 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();
   }
}
```

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 81

Before working with the CLR UDF examples you may want to read the following concept topics:

- Chapter 3, ".NET common language runtime (CLR) routines," on page 45
- "Creating .NET CLR routines from DB2 Command Window" on page 55
- "External scalar functions" on page 14
- "Building common language runtime (CLR) .NET routines" in *Developing ADO.NET and OLE DB Applications*

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

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
- Example 2: Visual Basic parameter style SQL scalar function

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 16. 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 17. 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 17. 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 17. 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 18. 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 18. 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
```

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 76

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

- Chapter 3, ".NET common language runtime (CLR) routines," on page 45
- "Creating .NET CLR routines from DB2 Command Window" on page 55
- "Benefits of using routines" on page 5
- "Building common language runtime (CLR) .NET routines" in *Developing ADO.NET and OLE DB Applications*

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

Use the following examples as references when making your own Visual Basic CLR procedures:

- The Visual Basic external code file
- Example 1: Visual Basic parameter style GENERAL procedure
- Example 2: Visual Basic parameter style GENERAL WITH NULLS procedure
- Example 3: Visual Basic parameter style SQL procedure
- Example 4: Visual Basic procedure returning a result set
- Example 5: Visual Basic procedure accessing the dbinfo structure
- Example 6: Visual Basic procedure in PROGRAM TYPE MAIN style

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 19. 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 20. 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'
```

Table 20. Code to create a Visual Basic parameter style GENERAL procedure (continued)

```
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 21. 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 22. 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 22. 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 23. 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 24. 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'
```

```
  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 occurred"
    '
    ' 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 25. 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 25. 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
```

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:

- .NET common language runtime (CLR) routines
- Creating .NET CLR routines from DB2 Command Window
- Benefits of using routines
- "Building common language runtime (CLR) .NET routines" in *Developing ADO.NET and OLE DB Applications*

The examples below makes use of a table named `xmlDataTable` that is defined as follows:

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)

INSERT INTO xmlDataTable 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

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 26. 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 `xmlDataTable`. 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 27. 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' ;
```

```
//*************************************************************************
//  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 27. 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 "
                   + "xmlDataTable( 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(\"xmlDataTable.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 xmlDataTable "
                    + "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;
}
```

Example: XML and XQuery support in C procedure

Once the basics of procedures, the essentials of C routines, XQuery and XML are understood, you can start creating and using C procedures with XML features.

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

Prerequisites

Before working with the C procedure example you might want to read the following concept topic:

- Benefits of using routines

The examples below makes use of a table named `xmlDataTable` that is defined as follows:

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)

INSERT INTO xmlDataTable 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 procedures:

- The C external code file
- Example 1: C parameter style SQL procedure with XML features

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.SQC` and has the following format:

Table 28. C external code file format

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sqlda.h>
#include <sqlca.h>
#include <sqludf.h>
#include <sql.h>
#include <memory.h>

// C procedures
               ...
```

The file inclusions are indicated at the top of the file. There are no extra include files required for XML support in embedded SQL routines.

It is important to note the name of the file and the name of the function that corresponds to the procedure implementation. These names are important, because the EXTERNAL clause of the CREATE PROCEDURE statement for each procedure must specify this information so that the DB2 database manager can locate the library and entry point that corresponds to the C procedure.

Example 1: C parameter style SQL procedure with XML features

This example shows the following:

- CREATE PROCEDURE statement for a parameter style SQL procedure
- C code for a parameter style SQL procedure with XML parameters

This procedure receives two input parameters. The first input parameter is named `inNum` and is of type INTEGER. The second input parameters is named `inXML` and is of type XML. The values of the input parameters are used to insert a row into the table `xmlDataTable`. Then an XML value is retrieved using an SQL statement. Another XML value is retrieved using an XQuery expression. The retrieved XML values are respectively assigned to two output parameters, `out1XML` and `out2XML`.

No result sets are returned.

Table 29. Code to create a C parameter style SQL 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 C
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
FENCED
THREADSAFE
DETERMINISTIC
NO DBINFO
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc!xmlProc1' ;
```

```
//*************************************************************************
//  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:   out1XML -- XML data returned - value retrieved using XQuery
//          out2XML -- XML data returned - value retrieved using SQL
//*************************************************************************
```

Table 29. Code to create a C parameter style SQL procedure (continued)

```
#ifdef __cplusplus
extern "C"
#endif
SQL_API_RC SQL_API_FN testSecA1(sqlint32* inNum,
                                SQLUDF_CLOB* inXML,
                                SQLUDF_CLOB* out1XML,
                                SQLUDF_CLOB* out2XML,
                                SQLUDF_NULLIND *inNum_ind,
                                SQLUDF_NULLIND *inXML_ind,
                                SQLUDF_NULLIND *out1XML_ind,
                                SQLUDF_NULLIND *out2XML_ind,
                                SQLUDF_TRAIL_ARGS)
{
  char *str;
  FILE *file;

  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
       sqlint32 hvNum1;
       SQL TYPE IS XML AS CLOB(200) hvXML1;
       SQL TYPE IS XML AS CLOB(200) hvXML2;
       SQL TYPE IS XML AS CLOB(200) hvXML3;
  EXEC SQL END DECLARE SECTION;

  /* Check null indicators for input parameters */
  if ((*inNum_ind < 0) ||  (*inXML_ind < 0)) {
     strcpy(sqludf_sqlstate, "38100");
     strcpy(sqludf_msgtext, "Received null input");
     return 0;
  }

  /* Copy input parameters to host variables */
  hvNum1 = *inNum;
  hvXML1.length = inXML->length;
  strncpy(hvXML1.data, inXML->data, inXML->length);

  /* Execute SQL statement */
  EXEC SQL
     INSERT INTO xmlDataTable (num, xdata) VALUES (:hvNum1, :hvXML1);

  /* Execute SQL statement */
  EXEC SQL
     SELECT xdata INTO :hvXML2
        FROM xmlDataTable
           WHERE num = :hvNum1;

  sprintf(stmt5, "SELECT XMLQUERY('for $x in $xmldata/doc
                                   return <carInfo>{$x/model}</carInfo>'
                                   passing by ref xmlDataTable.xdata
                                       as \"xmldata\" returning sequence)
                    FROM xmlDataTable WHERE num = ?");

  EXEC SQL PREPARE selstmt5 FROM :stmt5 ;
  EXEC SQL DECLARE c5 CURSOR FOR selstmt5;
  EXEC SQL OPEN c5 using :hvNum1;
  EXEC SQL FETCH c5 INTO :hvXML3;

  exit:

  /* Set output return code */
  *outReturnCode = sqlca.sqlcode;
  *outReturnCode_ind = 0;

  return 0;
}
```

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 65

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

- Chapter 3, ".NET common language runtime (CLR) routines," on page 45
- "Creating .NET CLR routines from DB2 Command Window" on page 55
- "External scalar functions" on page 14
- "Building common language runtime (CLR) .NET routines" in *Developing ADO.NET and OLE DB Applications*

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

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

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 30. 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 file system. 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 31. 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 31. 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 31. 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 32. 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 CONTROL SAFE
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!CountUp' ;
```

Table 32. 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;
      }
   }
}
```

Chapter 4. IBM Data Server Provider for .NET

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

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

- DB2 Version 9 (or later) for Linux, UNIX, and Windows
- DB2 Universal Database™ Version 8.2 for Windows, UNIX, and Linux
- DB2 for i5/OS® Version 5 Release 4 (or later), through DB2 Connect™
- IBM Informix Dynamic Server, Version 11.10 or later
- IBM UniData®, Version 7.1.11 or later
- IBM UniVerse, Version 10.2 or later

To develop and run applications that use Data Server Provider for .NET you need the .NET Framework

In addition to the IBM Data Server Provider for .NET, the IBM Database Development Add-Ins enable you to quickly and easily develop .NET applications for IBM data servers using Microsoft Visual Studio. You can also use the Add-Ins to create database objects such as indexes and tables, and develop server-side objects, such as stored procedures and user-defined functions.

IBM Data Server Provider for .NET database system requirements

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

- DB2 Version 9 (or later) for Linux, UNIX, and Windows
- DB2 Universal Database Version 8.2 for Windows, UNIX, and Linux
- DB2 for i5/OS Version 5 Release 4 (or later), through DB2 Connect
- IBM Informix Dynamic Server, Version 11.10 or later
- IBM UniData, Version 7.1.11 or later
- IBM UniVerse, Version 10.2 or later

Before using a IBM Data Server client or server installer to install the IBM Data Provider for .NET, you must already have one of the following .NET Framework versions:

- .NET Framwork Version 2.0
- .NET Framwork Version 3.0
- .NET Framwork Version 3.5

Without a .NET framework installed the IBM Data Server Client and driver installer will not install the IBM Data Server Provider for .NET. You will need to install the data provider manually.

For DB2 for i5/OS, the following fix is required on the server: APAR II13348.

32-bit and 64-bit support for ADO.NET applications

IBM Data Server data providers for .NET support both 32-bit and 64-bit .NET applications.

Following are the .NET data providers that are shipped with DB2 Version 9 (or later) clients and servers, and their 32-bit and 64-bit support levels.

Table 33. 32-bit and 64-bit support in IBM Data Server data providers for .NET

.NET data provider Framework	32-bit support	64-bit support
IBM Data Server Provider for .NET, Framework Version 2.0, Version 3.0, and Version 3.5	Yes	Yes

Note: CLR stored procedures and user-defined functions are supported in both 32-bit and 64bit editions of the IBM Data Server Provider for .NET. Prior to that fix pack, support was included for 32bit editions only.

There are 32-bit and 64-bit editions of the IBM Data Server Provider for .NET, each supporting the 32-bit and 64-bit editions of the .NET Framework version 2.0 CLR respectively. During the installation of the DB2 or DB2 Connect client or server software, one of these two IBM Data Server Provider for .NET editions will be installed:

for Windows on 32-bit AMD and Intel® systems (x86)
The 32-bit edition of the IBM Data Server Provider for .NET, Framework Version 2.0, Version 3.0, and Version 3.5 is installed with DB2 Version 9 (or later) or DB2 Connect.

for Windows on AMD64 and Intel EM64T systems (x64)
Only the 64-bit edition of the IBM Data Server Provider for .NET is installed with DB2 Version 9 (or later) or DB2 Connect. The 64-bit edition of the IBM Data Server Provider for .NET do not support the IA-64 architecture.

You can run 32-bit .NET applications on a 64-bit Windows instance, using WOW64, but you will need a 32-bit edition of the IBM Data Server Provider for .NET instead. To get a 32-bit IBM Data Server Provider for .NET on your 64bit computer, you can install the 32bit version of IBM Data Server Driver Package.

Programming applications to use the IBM Data Server Provider for .NET

Generic coding with the ADO.NET common base classes

The .NET Framework, versions 2.0, 3.0, and 3.5, 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 IBM Data Server Provider for .NET, Framework 2.0, 3.0, and 3.5, are inherited from the `System.Data.Common` base classes. As a result, generic ADO.NET applications will work with DB2 and other supported databases through the IBM Data Server Provider for .NET.

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 IBM Data Server Provider for .NET, and results in the initialization of a `DbProviderFactory` instance that creates database provider object instances specific to the IBM Data Server Provider for .NET. 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 IBM Data Server Provider for .NET, and if so, generates a connection string to connect to the SAMPLE database.

Connecting to a database from an application using the IBM Data Server Provider for .NET

When using the IBM Data Server Provider for .NET, a database connection is established through the `DB2Connection` class.

To connect to a database, you must first 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 IBM Data Server Provider for .NET

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 IBM Data Server Provider for .NET 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)

Creating a trusted connection through IBM Data Server Provider for .NET

Starting in Version 9.5 Fix Pack 1, .NET applications support trusted context using connection string keywords.

The following keywords are available in the connection string:

- TrustedContextSystemUserID, or tcsuid, which specifies the trusted context SYSTEM AUTHID to be used with the connection.
- TrustedContextSystemPassword, or tcspwd, which specifies the password corresponding to the trusted context SYSYTEM AUTHID to be used with the connection.

If the TrustedContextSystemPassword keyword is specified without a TrustedContextSystemUserID keyword value, an InvalidArgument exception is thrown. The UserID keyword is also required in a trusted context scenario.

Example

Suppose a trusted context has been established on the server with the following information:

```
CREATE TRUSTED CONTEXT ctxName1
BASED UPON CONNECTION USING SYSTEM AUTHID masteruser
ATTRIBUTES ( PROTOCOL 'TCPIP',
             ADDRESS '9.26.146.201',
             ENCRYPTION 'NONE' )
ENABLE
WITH USE FOR userapp1 WITH AUTHENTICATION, userapp2 WITH AUTHENTICATION;
```

The SYSTEM AUTHID, masteruser, has a corresponding password, masterpassword. Each specific user/application, userapp1 and userapp2, has a corresponding password, passapp1 and passapp2.

In order to use this trusted context, applications would issue connection strings as follows:

- Application 1

```
database=db;server=foobar:446;
UserID=userapp1;Password=passapp1;
TrustedContextSystemUserID=masteruser;TrustedContextSystemPassword=masterpassword
```

- Application 2

```
database=db;server=foobar:446;
UserID=userapp2;Password=passapp2;
TrustedContextSystemUserID=masteruser;TrustedContextSystemPassword=masterpassword
```

Note: The UserID keyword corresponds to the end user of the connection in a trusted context situation, just as in standard applications.

Thus, a simple .NET program could look like the following:

```
[C#]
DB2Connection conn = new DB2Connection();

conn.ConnectionString = "database=db;server=foobar:446;"
     + "UserID=userapp1;Password=passapp1;"
     + "TrustedContextSystemUserID=masteruser;"
     + "TrustedContextSystemPassword=masterpassword;"

conn.Open();

// Do processing as userapp1, such as querying tables

conn.Close();

conn.ConnectionString = "database=db;server=foobar:446;UserID=userapp2;"
     + "Password=passapp2;TrustedContextSystemUserID=masteruser;"
     + "TrustedContextSystemPassword=masterpassword;"

conn.Open();

// Do processing as userapp2

conn.Close();
```

If the trusted context processing fails because no trusted context was set up on the server, or the server does not support trusted contexts, an error with SQLCODE CLI0197E will be thrown. If the TrustedContextSystemUserID keyword value is invalid (too long, for example), an error with SQLCODE CLI0124E will be thrown. The server might report an error with SQLCODE SQL1046N, SQL30082N, or SQL0969N with a native error code of -20361. Any of these errors will cause Open() to fail.

Note: The trusted context processing happens on the next communication with the server.

SQL data type representation in ADO.NET database applications

ADO.NET database applications can reference DB2 SQL data type values as parameter values to be used as part of SQL statement execution and as variables, however the appropriate IBM Data Server Provider for .NET data type values and .NET Framework data type values must be used to ensure that there is no truncation or loss of data when accessing or retrieving the values.

For specifying parameter values to be used as part of a SQL statement to be executed, IBM Data Server Provider for .NET objects must be used. The `DB2Parameter` object is used to represent a parameter to be added to a `DB2Command` object which represents a SQL statement. When specifying the data type value for the parameter, the IBM Data Server Provider for .NET data type values available in the `IBM.Data.DB2Types` namespace must be used. The `IBM.Data.DB2Types` namespace provides classes and structures to represent each of the supported DB2 SQL data types.

For local variables that might temporarily hold SQL data type values, appropriate IBM Data Server Provider for .NET data types, as defined in the IBM.Data.DB2Types Namespace, must be used.

The following table shows mappings between DB2Type data types, DB2 data types, Informix data types, Microsoft .NET Framework types, and DB2Types classes and structures.

Category	DB2Types Classes and Structures	DB2Type Data Type	DB2 Data Type	Informix Data Type	.NET Data Type
Numeric	DB2Int16	SmallInt	SMALLINT	BOOLEAN, SMALLINT	Int16
	DB2Int32	Integer	INT	INTEGER, INT, SERIAL	Int32
	DB2Int64	BigInt	BIGINT	BIGINT, BIGSERIAL, INT8, SERIAL8	Int64
	DB2Real, DB2Real370	Real	REAL	REAL, SMALLFLOAT	Single
	DB2Double	Double	DOUBLE PRECISION	DECIMAL (≤31), DOUBLE PRECISION	Double
	DB2Double	Float	FLOAT	DECIMAL (32), FLOAT	Double
	DB2Decimal	Decimal	DECIMAL	MONEY	Decimal
	DB2DecimalFloat	DecimalFloat	DECFLOAT(16|34)[8]		Decimal
	DB2Decimal	Numeric	DECIMAL	DECIMAL (≤31), NUMERIC	Decimal
Date/Time	DB2Date	Date	DATE	DATETIME (date precision)	Datetime
	DB2Time	Time	TIME	DATETIME (time precision)	TimeSpan
	DB2TimeStamp	Timestamp	TIMESTAMP	DATETIME (time and date precision)	DateTime
XML	DB2Xml	Xml[6]	XML		Byte[]
Character data	DB2String	Char	CHAR	CHAR	String
	DB2String	VarChar	VARCHAR	VARCHAR	String
	DB2String	LongVarChar[5]	LONG VARCHAR	LVARCHAR	String

5. These data types are not supported as parameters in DB2 .NET common language runtime routines.
6. A DB2ParameterClass.ParameterName property of the type DB2Type.Xml can accept variables of the following types: String, byte[], DB2Xml, and XmlReader.
7. These data types are applicable only to DB2 UDB for z/OS.
8. This data type is only supported for DB2 for z/OS Version 9 and later releases and for DB2 for Linux, UNIX, and Windows Version 9.5 and later releases.

Category	DB2Types Classes and Structures	DB2Type Data Type	DB2 Data Type	Informix Data Type	.NET Data Type
Binary data	DB2Binary	Binary	CHAR FOR BIT DATA		Byte[]
	DB2Binary	Binary[7]	BINARY		Byte[]
	DB2Binary	VarBinary[7]	VARBINARY		Byte[]
	DB2Binary	LongVarBinary[5]	LONG VARCHAR FOR BIT DATA		Byte[]
Graphic data	DB2String	Graphic	GRAPHIC		String
	DB2String	VarGraphic	VARGRAPHIC		String
	DB2String	LongVarGraphic[5]	LONG VARGRAPHIC		String
LOB data	DB2Clob	Clob	CLOB	CLOB, TEXT	String
	DB2Blob	Blob	BLOB	BLOB, BYTE	Byte[]
	DB2Clob	DbClob	DBCLOB		String
Row ID	DB2RowId	RowId	ROWID		Byte[]

Executing SQL statements from an application using the IBM Data Server Provider for .NET

When using the IBM Data Server Provider for .NET, 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 Visual Basic.NET:

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

Reading result sets from an application using the IBM Data Server Provider for .NET

When using the IBM Data Server Provider for .NET, 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 IBM Data Server Provider for .NET

When using the IBM Data Server Provider for .NET, 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 conn
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 conn
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 DB2Connection conn
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 DB2Connection conn
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()
```

Optimizing queries in .NET applications using pureQuery

The .NET client drivers can leverage features found in pureQuery technology. These features enables existing .NET application queries to execute as static SQL. Static queries avoid the need to prepare certain statements at runtime. This can lead to improved security and performance in your applications.

Before you begin

In order to leverage the pureQuery technology features you must enable pureQuery for your IBM Data Server Provider for .NET, version 9.5.3 or later, with the IBM Optim pureQuery Runtime 2.1 or later.

About this task

This task will go through the basic steps of discovering and using static queries for your .NET applications.

Procedure

1. Set up your .NET application to capture potential statements:
   a. Locate the part of your code that manages the connection to your database.
   b. Set the captureMode keyword to `on`.
   c. If executionMode keyword is set, ensure the value is set to `dynamic`.
   d. Set collection to the collection name for the package name (collection.rootPkgName).
   e. Set rootPkgName to the root package for the package name (collection.rootPkgName).
   f. Set the pureQueryXML keyword to the path and filename where you want to store the captured statements.
   g. Execute your application to capture statements in the pureQueryXML file you specified.
   h. In a command prompt, run the db2cap utility to bind your capture file with the database. The db2cap command has a few parameters that you will need to pass in.
2. Change your .NET application to run your captured statements statically:
   a. Locate the part of your code that manages the connection to your database.
   b. Remove the captureMode keyword or set it to `off`.
   c. Set the executionMode keyword to `static`.

Results

You should now have a .NET application that executes captured statements statically. Statements that could not be captured continue to be executed dynamically.

Example

The following is an example of setting up a connection string to capture executed statements.

```
[C#]
string myConnectionString =
    "Database=Sample;captureMode=ON;pureQueryXML=c:\temp\capfile.xml";
DB2Connection myConn = new DB2Connection(myConnectionString);
string myInsertQuery = "INSERT INTO STAFF (ID, NAME) Values(...)";
DB2Command myDB2Command = new DB2Command(myInsertQuery);
myDB2Command.Connection = myConn;
myConn.Open();
myDB2Command.ExecuteNonQuery();
myConn.Close();

[Visual Basic]
Dim myConnectionString As String = _
    "Database=Sample;captureMode=ON;pureQueryXML=c:\temp\capfile.xml"
Dim myConn As New DB2Connection(myConnectionString)
Dim myInsertQuery As String = "INSERT INTO STAFF (ID, NAME) Values(...)"
Dim myDB2Command As New DB2Command(myInsertQuery)
myDB2Command.Connection = myConn
myConn.Open()
myDB2Command.ExecuteNonQuery()
myConn.Close()
```

The next example changes the connection string to run the captured statements statically.

```
[C#]
string myConnectionString =
    "Database=Sample;executionMode=STATIC;pureQueryXML=c:\temp\capfile.xml";
DB2Connection myConn = new DB2Connection(myConnectionString);
string myInsertQuery = "INSERT INTO STAFF (ID, NAME) Values(...)";
DB2Command myDB2Command = new DB2Command(myInsertQuery);
myDB2Command.Connection = myConn;
myConn.Open();
myDB2Command.ExecuteNonQuery();
myConn.Close();

[Visual Basic]
Dim myConnectionString As String = _
    "Database=Sample;captureMode=ON;pureQueryXML=c:\temp\capfile.xml"
Dim myConn As New DB2Connection(myConnectionString)
Dim myInsertQuery As String = "INSERT INTO STAFF (ID, NAME) Values(...)"
Dim myDB2Command As New DB2Command(myInsertQuery)
myDB2Command.Connection = myConn
myConn.Open()
myDB2Command.ExecuteNonQuery()
myConn.Close()
```

Enabling pureQuery for .NET applications

Optim pureQuery is a licensed feature that must be purchased. You can obtain pureQuery features by purchasing the IBM Optim pureQuery Runtime product.

Before you begin

Before attempting to enable pureQuery, you must have installed the IBM Optim pureQuery Runtime 2.1 or later. You must also have installed at least one of the following products that include the IBM Data Server Provider for .NET:

- IBM Data Server Driver for ODBC, CLI, and .NET, Version 9.5.3, or IBM Data Server Driver Package, Version 9.5.4 (or later)
- IBM Data Server Runtime Client, Version 9.5.3 (or later)
- IBM Data Server Client, Version 9.5.3 (or later)
- DB2 for Linux, UNIX, and Windows, Version 9.5 Fix pack 3 (or later)

About this task

This task will enable pureQuery for .NET applications. After completing this task you will be able to use features of pureQuery technology in your .NET applications.

Restrictions

none.

Procedure

1. Activate the pureQuery license file for .NET.
   - For the IBM Data Server Driver Package:
     a. Locate the correct version of the license file in the pureQuery installation directory. For example, `C:\Program Files\IBM\purequery\license\clientv97\`.

        Note: You can use the `db2level` command provided in the driver package to determine the version of your driver package.

b. Copy the license file `dspq_rt.lic` to the IBM Data Server Driver Package license directory. You can find the license directory in the directory where the IBM Data Server Driver Package is installed. For example, `C:\Program Files\IBM\IBM DATA SERVER DRIVER\license\`.

- For the other DB2 clients or data servers:
  - In a command prompt, execute the following command:`db2licm -a C:\pqRuntime21\pureQuery\dspq_rt.lic` where `C:\pqRuntime21\` is the directory of your pureQuery Runtime installation.

2. For Microsoft Windows Vista operating systems, to complete the activation of the pureQuery license file you must run your first capture under an administrator account. This step needs to be run only once. Attempts to use pureQuery features without taking this step will result in an error stating that a valid license key could not be found. After a successful first capture using an administrator account pureQuery features will be enabled for users on the system.

Results

The pureQuery license file is activated for your drivers. Your .NET applications can now take advantage of client optimization features.

What to do next

After activating the pureQuery license file, you can begin using pureQuery to optimize the data provider for your .NET applications.

Provider support for Microsoft Entity Framework

The IBM Data Server Provider for .NET enables users to take advantage of the Microsoft Entity Framework with IBM data servers. Users can generate EDM schemas, as well as write and execute EntitySQL and LINQ statements to Entities applications with the supported server versions.

System requirements

The provider will work with the following IBM data server products:

- DB2 for Linux, UNIX, and Windows, Version 8 (or later)
- IBM Data Server Client, Version 9.5.3 (or later)
- IBM Data Server Runtime Client, Version 9.5.3 (or later)
- IBM Data Server Driver for ODBC, CLI, and .NET, Version 9.5.3, or IBM Data Server Driver Package, Version 9.5.4 (or later)

You will need to have Microsoft .NET Framework 3.5 SP1 with the Microsoft ADO.NET Entity Framework. Microsoft Visual Studio 2008 will also be required to manipulate Entity Data Models using Microsoft's Entity Data Model Wizard or ADO.NET Entity Designer.

Supported IBM data servers include the following IBM data servers:

- DB2 for Linux, UNIX, and Windows, Version 8 (or later)
- DB2 for IBM i5/OS V5R3
- DB2 for IBM i 5.4
- DB2 for z/OS V7, V8, and V9
- IBM Informix Dynamic Server, Version 11.10 (or later)

- UniVerse 10.2 (or later) and UniData 7.1 (or later)

Known limitations

For information on the current list of limitations, go to http://www.ibm.com/developerworks/wikis/display/DB2/IBM%20Data%20Server%20LINQ%20Entity%20Framework%20Limitations

Using the Enterprise Library data access module

The Enterprise Library is a collection of application blocks designed to assist developers with common development challenges. Application blocks are provided as source code that can be used as is or modified for development projects.

The Enterprise Library data access module for IBM data servers can be obtained along with other modules at http://codeplex.com/entlibcontrib/SourceControl/PatchList.aspx.

For information on how to install and use the Enterprise Library data access module with IBM data servers (DB2, IDS, and U2), see the readme file found in the download package.

Resources

Below are several online resources that describe how to use the data access modules:

- EntLib Contrib Project Homepage: http://www.codeplex.com/entlibcontrib
- patterns & practices for Enterprise Library: http://www.codeplex.com/entlib
- Microsoft Enterprise Library Homepage: http://msdn.microsoft.com/en-us/library/cc467894.aspx
- IBM DB2 for .NET: http://www.ibm.com/software/data/db2/windows/dotnet.html

Building .NET applications

Building Visual Basic .NET applications

DB2 products provide a bldapp.bat batch file for compiling and linking DB2 Visual Basic .NET applications. This file is 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).

This task will take you through the basic steps of building a Visual Basic .NET application using bldapp.bat with the DbAuth sample file.

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`.

Building C# .NET applications

DB2 products provide a bldapp.bat batch file for compiling and linking DB2 C# .NET applications. This batch file is 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).

This task will take you through the basic steps of building a C# .NET application using bldapp.bat with the DbAuth sample file.

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`.

Visual Basic .NET application compile and link options

This topic describes the various options available when compiling and linking Visual Basic .NET applications.

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 dynamic link library for the .NET framework version you are using. **%VERSION%** There are several supported versions of the .NET framework for applications. DB2 has a dynamic link library for each. For .NET Framework Version 2.0, 3.0, and 3.5, `%VERSION%` points to the `netf20\` sub-directory.

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 dynamic link library, used by the LCTrans application, from the RootCOM.vb source file,

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

/target:library %1.cs
Create the dynamic 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 dynamic link library for the .NET framework version you are using.

%VERSION%
There are several supported versions of the .NET framework for applications. DB2 has a dynamic link library for each. For .NET Framework Version 2.0 and 3.0, %VERSION% points to the netf20\ sub-directory.

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

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

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

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

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

Refer to your compiler documentation for additional compiler options.

C# .NET application compile and link options

This topic describes the various options available when compiling and linking Visual Basic .NET applications.

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 dynamic link library for the .NET framework version you are using.

%VERSION%
There are several supported versions of the .NET framework for applications. DB2 has a dynamic link library for each. For .NET Framework Version 2.0, 3.0, and 3.5, `%VERSION%` points to the `netf20\` 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` dynamic link library, used by the `LCTrans` application, from the `RootCOM.cs` source file,

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

/target:library %1.cs
Create the dynamic 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 dynamic link library for the .NET framework version you are using.

%VERSION%
There are several supported versions of the .NET framework for applications. DB2 has a dynamic link library for each. For .NET Framework Version 2.0, 3.0, and 3.5, `%VERSION%` points to the `netf20\` sub-directory.

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

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

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

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

/r:RootCOM.dll
Reference the RootCOM dynamic link library (for `LCTrans.cs`).

Refer to your compiler documentation for additional compiler options.

Chapter 5. IBM OLE DB Provider for DB2

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.

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, 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 or later of the Microsoft OLE DB specification.

System Requirements

Refer to the announcement letter for the IBM OLE DB Provider for DB2 data 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).

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

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 34. 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	

Table 34. Schema Rowsets Supported by the IBM OLE DB Provider for DB2 (continued)

Supported GUIDs	Supported Restrictions	Supported Columns	Notes
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.
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	

Table 34. Schema Rowsets Supported by the IBM OLE DB Provider for DB2 (continued)

Supported GUIDs	Supported Restrictions	Supported Columns	Notes
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 35. 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

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 36. 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"
LONG VARGRAPHIC	DBTYPE_WSTR	"DBTYPE_WLONGVARCHAR"	"LONG VARGRAPHIC"

Table 36. 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
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 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.

The following table shows data conversions 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 37. 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	DATALINK
DBTYPE_EMPTY																						

Table 37. Data conversions from OLE DB types to DB2 types (continued)

OLE DB Type Indicator	DB2 Data Types																						
																		For Bit Data					
	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_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												
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																							

Table 37. Data conversions from OLE DB types to DB2 types (continued)

OLE DB Type Indicator	DB2 Data Types																						
																		For Bit Data					
	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_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												

Table 37. Data conversions from OLE DB types to DB2 types (continued)

OLE DB Type Indicator	DB2 Data Types																						
																		For Bit Data					
	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_DBTIME								X	X	X	X												
DBTYPE_DBTIMESTAMP							X	X	X	X	X												
DBTYPE_FILETIME																							
DBTYPE_PROP_VARIANT																							
DBTYPE_HCHAPTER																							
DBTYPE_VARNUMERIC																							

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.

The following table shows supported 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 38. Data conversions from DB2 types to OLE DB types

OLE DB Type Indicator	DB2 Data Types																					
																		For Bit Data				
	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_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

Table 38. Data conversions from DB2 types to OLE DB types (continued)

OLE DB Type Indicator	DB2 Data Types																					
																		For Bit Data				
	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_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																						
DBTYPE_ARRAY																						
DBTYPE_VECTOR																						
DBTYPE_UDT																						
DBTYPE_DBDATE							X	X	X	X	X	X		X	X	X		X	X	X		X

Table 38. Data conversions from DB2 types to OLE DB types (continued)

OLE DB Type Indicator	DB2 Data Types																					
																		For Bit Data				
	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_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.

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 tables list 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 39. Blobs

Interface	DB2	ODBC Provider
ISequentialStream	Yes	Yes

Table 40. Commands

Interface	DB2	ODBC Provider
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

Table 41. DataSources

Interface	DB2	ODBC Provider
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

Table 42. Enumerators

Interface	DB2	ODBC Provider
IDBInitialize	Yes	Yes
IDBProperties	Yes	Yes
IParseDisplayName	Yes	No

Table 42. Enumerators (continued)

Interface	DB2	ODBC Provider
ISourcesRowset	Yes	Yes
ISupportErrorInfo	Yes	Yes

Table 43. Error Lookup Service

Interface	DB2	ODBC Provider
IErrorLookUp	Yes	Yes

Table 44. Error Objects

Interface	DB2	ODBC Provider
IErrorInfo	Yes	No
ISQLErrorInfo (custom)	Yes	No

Table 45. Multiple Results

Interface	DB2	ODBC Provider
IMultipleResults	Yes	Yes
ISupportErrorInfo	Yes	Yes

Table 46. Rowsets

Interface	DB2	ODBC Provider
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

Table 46. Rowsets (continued)

Interface	DB2	ODBC Provider
IRowsetView	No	No
ISupportErrorInfo	Yes	Yes
Note: 1. The values to be returned are approximations. Deleted rows will not be skipped.		

Table 47. Sessions

Interface	DB2	ODBC Provider
IAlterIndex	No	No
IAlterTable	No	No
IDBCreateCommand	Yes	Yes
IDBSchemaRowset	Yes	Yes
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

Table 48. View Objects

Interface	DB2	ODBC Provider
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 49. Properties Supported by the IBM OLE DB Provider for DB2: Data Source (DBPROPSET_DATASOURCE)

Properties	Default Value	R/W
DBPROP_MULTIPLECONNECTIONS	VARIANT_FALSE	R
DBPROP_RESETDATASOURCE	DBPROPVAL_RD_RESETALL	R/W

Table 50. Properties Supported by the IBM OLE DB Provider for DB2: DB2 Data Source (DBPROPSET_DB2DATASOURCE)

Properties	Default Value	R/W
DB2PROP_REPORTISLONGFORLONGTYPES	VARIANT_FALSE	R/W
DB2PROP_RETURNCHARASWCHAR	VARIANT_TRUE	R/W
DB2PROP_SORTBYORDINAL	VARIANT_FALSE	R/W

Table 51. Properties Supported by the IBM OLE DB Provider for DB2: Data Source Information (DBPROPSET_DATASOURCEINFO)

Properties	Default Value	R/W
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
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

Table 51. Properties Supported by the IBM OLE DB Provider for DB2: Data Source Information (DBPROPSET_DATASOURCEINFO) (continued)

Properties	Default Value	R/W
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

Table 52. Properties Supported by the IBM OLE DB Provider for DB2: Initialization (DBPROPSET_DBINIT)

Properties	Default Value	R/W
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

Table 52. Properties Supported by the IBM OLE DB Provider for DB2: Initialization (DBPROPSET_DBINIT) (continued)

Properties	Default Value	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

Table 53. Properties Supported by the IBM OLE DB Provider for DB2: Rowset (DBPROPSET_ROWSET)

Properties	Default Value	R/W
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
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

Table 53. Properties Supported by the IBM OLE DB Provider for DB2: Rowset (DBPROPSET_ROWSET) (continued)

Properties	Default Value	R/W
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

Table 54. Properties Supported by the IBM OLE DB Provider for DB2: DB2 Rowset (DBPROPSET_DB2ROWSET)

Properties	Default Value	R/W
DBPROP_ISLONGMINLENGTH	32000	R/W

Table 55. Properties Supported by the IBM OLE DB Provider for DB2: Session (DBPROPSET_SESSION)

Properties	Default Value	R/W
DBPROP_SESS_AUTOCOMMITISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY	R/W

Note:

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 reroute is enabled, the connection timeout behavior is dictated by client reroute.

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 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

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 56. 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.

Table 56. Keywords supported by the IBM OLE DB Provider for DB2 (continued)

Keyword	Value	Meaning
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.

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.

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 or LOB data 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 57. Command Methods

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
Cancel	ICommand	Yes
CreateParameter		Yes
Execute		Yes

Table 58. Command Properties

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
ActiveConnection	(ADO specific)	
Command Text	ICommandText	Yes
Command Timeout	ICommandProperties::SetProperties DBPROP_COMMANDTIMEOUT	Yes
CommandType	(ADO specific)	
Prepared	ICommandPrepare	Yes
State	(ADO specific)	

Table 59. Command Collections

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
Parameters	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	Yes
Properties	ICommandProperties IDBProperties	Yes

Table 60. Connection Methods

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
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 61. Connection Properties

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
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)	

Table 62. Connection Collection

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
Errors	IErrorRecords	Yes
Properties	IDBProperties	Yes

Table 63. Error Properties

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
	IErrorRecords	
Description		Yes
NativeError		Yes
Number		Yes
Source		Yes
SQLState		Yes
HelpContext		No
HelpFile		No

Table 64. Field Methods

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
AppendChunk	ISequentialStream	Yes
GetChunk		Yes

Table 65. Field Properties

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
Actual Size	IAccessor IRowset	Yes
Attributes	IColumnInfo	
DataFormat		Yes
DefinedSize		Yes
Name		Yes
NumericScale		Yes
Precision		Yes
Type		Yes
OriginalValue	IRowsetUpdate	Yes (Cursor Service)
UnderlyingValue	IRowsetRefresh	Yes (Cursor Service)
	IRowsetResynch	Yes (Cursor Service)
Value	IAccessor IRowset	Yes

Table 66. Field Collection

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
Properties	IDBProperties IRowsetInfo	Yes

Table 67. Parameter Methods

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
AppendChunk	ISequentialStream	Yes

Table 67. Parameter Methods (continued)

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
Attributes Direction Name NumericScale Precision Scale Size Type	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	 Yes No Yes Yes Yes Yes Yes
Value	IAccessor ICommand	Yes

Table 68. Parameter Collection

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
Properties		Yes

Table 69. RecordSet Methods

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
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

Table 69. RecordSet Methods (continued)

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
Open	ICommand IOpenRowset	Yes
Requery	ICommand IOpenRowset	Yes
Resync	IRowsetRefresh	Yes (Cursor Service)
Supports	IRowsetInfo	Yes
Update UpdateBatch	IRowsetChange IRowsetUpdate	Yes Yes (Cursor Service)

Table 70. RecordSet Properties

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
AbsolutePage	IRowsetLocate IRowsetScroll	Yes Yes[1]
AbsolutePosition	IRowsetLocate IRowsetScroll	Yes Yes[1]
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

Table 70. RecordSet Properties (continued)

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
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)
Note: 1. The values to be returned are approximations. Deleted rows will not be skipped.		

Table 71. RecordSet Collection

Method/Property	OLE DB Interface/Property	IBM OLE DB Support
Fields	IColumnInfo	Yes
Properties	IDBProperties IRowsetInfo::GetProperties	Yes

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.

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 System i® 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

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 `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.

Chapter 6. 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. This provider is no longer tested. Users are recommended to use the IBM Data Server Provider for .NET.

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 *Developing ADO.NET and OLE DB Applications*.

In order to use the OLE DB .NET Data Provider, you must have the .NET Framework Version 2.0, 3.0, or 3.5 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 72. Useful 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 data server
PWD	password	The password for the user ID used to connect to the DB2 data server

Note: For the full list of ConnectionString keywords, see the Microsoft documentation.

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 OLE DB .NET Data Provider is no longer tested. Users are recommended to use the IBM Data Server Provider for .NET.

The following table identifies usage restrictions for the IBM OLE DB .NET Data Provider:

Table 73. 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"`	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.	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 73. 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 **Note:** 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.	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
`OleDbCommandBuilder. DeriveParameters`	The `OleDbCommandBuilder.DeriveParameters()` method does not include the `ReturnValue` parameter in the generated `OleDbParameterCollection`. SqlClient and the IBM Data Server Provider for .NET by default adds the parameter with `ParameterDirection.ReturnValue` to the generated `ParameterCollection`.	All

Table 73. IBM OLE DB .NET Data Provider restrictions (continued)

Class or feature	Restriction description	DB2 servers affected
`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" });`
		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
`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®, V7 or lower DB2 for OS/400 DB2 for VM/VSE

Table 73. IBM OLE DB .NET Data Provider restrictions (continued)

Class or feature	Restriction description	DB2 servers affected
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

Hints and tips

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 74. 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 7. 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. This provider is no longer tested. Users are recommended to use the IBM Data Server Provider for .NET.

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 *Developing ADO.NET and OLE DB Applications*.

In order to use the ODBC .NET Data Provider, you must have the .NET Framework Version 2.0, 3.0, or 3.5 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 75. Useful **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

Note: For the full list of **ConnectionString** keywords, see the Microsoft documentation.

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 ODBC .NET Data Provider is no longer tested. Users are recommended to use the IBM Data Server Provider for .NET.

The following table identifies usage restrictions for the IBM ODBC .NET Data Provider:

Table 76. 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"`	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";`
	
	`// This ends up re-executing the first statement`
	`command.ExecuteReader();`
		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 76. 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`
		DB2 for VM/VSE

Table 76. 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`
		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

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.
		All

Table 76. IBM ODBC .NET Data Provider restrictions (continued)

Class or feature	Restriction description	DB2 servers affected
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

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 OdbcCommandBuilderclass:
	`[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

Table 76. IBM ODBC .NET Data Provider restrictions (continued)

Class or feature	Restriction description	DB2 servers affected
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
`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. Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics (Task, concept and reference topics)
  - Help for DB2 tools
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF DVD)
  - printed books
- Command line help
  - Command help
  - Message help

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hardcopy 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 IBM Redbooks® publications 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 to 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.

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. English and translated DB2 Version 9.7 manuals in PDF format can be downloaded from www.ibm.com/support/docview.wss?rs=71&uid=swg2700947.

Although the tables identify books available in print, the books might not be available in your country or region.

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

Note: The *DB2 Information Center* is updated more frequently than either the PDF or the hard-copy books.

Table 77. DB2 technical information

Name	Form Number	Available in print	Last updated
Administrative API Reference	SC27-2435-01	Yes	November, 2009
Administrative Routines and Views	SC27-2436-01	No	November, 2009
Call Level Interface Guide and Reference, Volume 1	SC27-2437-01	Yes	November, 2009
Call Level Interface Guide and Reference, Volume 2	SC27-2438-01	Yes	November, 2009
Command Reference	SC27-2439-01	Yes	November, 2009
Data Movement Utilities Guide and Reference	SC27-2440-00	Yes	August, 2009
Data Recovery and High Availability Guide and Reference	SC27-2441-01	Yes	November, 2009
Database Administration Concepts and Configuration Reference	SC27-2442-01	Yes	November, 2009
Database Monitoring Guide and Reference	SC27-2458-01	Yes	August, 2009
Database Security Guide	SC27-2443-01	Yes	November, 2009
DB2 Text Search Guide	SC27-2459-01	Yes	November, 2009
Developing ADO.NET and OLE DB Applications	SC27-2444-01	Yes	August, 2009
Developing Embedded SQL Applications	SC27-2445-01	Yes	November, 2009
Developing Java Applications	SC27-2446-01	Yes	November, 2009
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-2447-00	No	August, 2009
Developing User-defined Routines (SQL and External)	SC27-2448-01	Yes	November, 2009
Getting Started with Database Application Development	GI11-9410-01	Yes	November, 2009
Getting Started with DB2 Installation and Administration on Linux and Windows	GI11-9411-00	Yes	August, 2009

Table 77. DB2 technical information (continued)

Name	Form Number	Available in print	Last updated
Globalization Guide	SC27-2449-00	Yes	August, 2009
Installing DB2 Servers	GC27-2455-01	Yes	November, 2009
Installing IBM Data Server Clients	GC27-2454-00	No	August, 2009
Message Reference Volume 1	SC27-2450-00	No	November, 2009
Message Reference Volume 2	SC27-2451-00	No	November, 2009
Net Search Extender Administration and User's Guide	SC27-2469-01	No	November, 2009
Partitioning and Clustering Guide	SC27-2453-01	Yes	November, 2009
pureXML Guide	SC27-2465-01	Yes	November, 2009
Query Patroller Administration and User's Guide	SC27-2467-00	No	August, 2009
Spatial Extender and Geodetic Data Management Feature User's Guide and Reference	SC27-2468-00	No	August, 2009
SQL Procedural Languages: Application Enablement and Support	SC27-2470-01	Yes	August, 2009
SQL Reference, Volume 1	SC27-2456-01	Yes	November, 2009
SQL Reference, Volume 2	SC27-2457-01	Yes	November, 2009
Troubleshooting and Tuning Database Performance	SC27-2461-01	Yes	November, 2009
Upgrading to DB2 Version 9.7	SC27-2452-01	Yes	November, 2009
Visual Explain Tutorial	SC27-2462-00	No	August, 2009
What's New for DB2 Version 9.7	SC27-2463-01	Yes	November, 2009
Workload Manager Guide and Reference	SC27-2464-01	Yes	August, 2009
XQuery Reference	SC27-2466-01	No	November, 2009

Table 78. DB2 Connect-specific technical information

Name	Form Number	Available in print	Last updated
Installing and Configuring DB2 Connect Personal Edition	SC27-2432-01	Yes	November, 2009
Installing and Configuring DB2 Connect Servers	SC27-2433-01	Yes	November, 2009

Table 78. DB2 Connect-specific technical information (continued)

Name	Form Number	Available in print	Last updated
DB2 Connect User's Guide	SC27-2434-01	Yes	November, 2009

Table 79. Information Integration technical information

Name	Form Number	Available in print	Last updated
Information Integration: Administration Guide for Federated Systems	SC19-1020-02	Yes	August, 2009
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-04	Yes	August, 2009
Information Integration: Configuration Guide for Federated Data Sources	SC19-1034-02	No	August, 2009
Information Integration: SQL Replication Guide and Reference	SC19-1030-02	Yes	August, 2009
Information Integration: Introduction to Replication and Event Publishing	GC19-1028-02	Yes	August, 2009

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* DVD 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 DVD 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 DVD 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/db2luw/v9r7.

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:

1. 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.
2. When you call, specify that you want to order a DB2 publication.
3. Provide your representative with the titles and form numbers of the books that you want to order. For titles and form numbers, see "DB2 technical library in hardcopy or PDF format" on page 163.

Displaying SQL state help from the command line processor

DB2 products return an SQLSTATE value for conditions that can be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

To start 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.

Accessing different versions of the DB2 Information Center

For DB2 Version 9.7 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/.

For DB2 Version 9.5 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

For DB2 Version 9.1 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/.

For DB2 Version 8 topics, go to the *DB2 Information Center* URL at: http://publib.boulder.ibm.com/infocenter/db2luw/v8/.

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.

- 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. 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 button in the **Languages** section of the **Tools** —> **Options** —> **Advanced** dialog. 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. Refresh the page to display the DB2 Information Center in your preferred language.

On some browser and operating system combinations, you must also change the regional settings of your operating system to the locale and language of your choice.

Updating the DB2 Information Center installed on your computer or intranet server

A locally installed DB2 Information Center must be updated periodically.

Before you begin

A DB2 Version 9.7 Information Center must already be installed. For details, see the "Installing the DB2 Information Center using the DB2 Setup wizard" topic in *Installing DB2 Servers*. All prerequisites and restrictions that applied to installing the Information Center also apply to updating the Information Center.

About this task

An existing DB2 Information Center can be updated automatically or manually:

- Automatic updates - updates existing Information Center features and languages. An additional benefit of automatic updates is that the Information Center is unavailable for a minimal period of time during the update. In addition, automatic updates can be set to run as part of other batch jobs that run periodically.
- Manual updates - should be used when you want to add features or languages during the update process. For example, a local Information Center was originally installed with both English and French languages, and now you want to also install the German language; a manual update will install German, as well as, update the existing Information Center features and languages. However, a manual update requires you to manually stop, update, and restart the Information Center. The Information Center is unavailable during the entire update process.

Procedure

This topic details the process for automatic updates. For manual update instructions, see the "Manually updating the DB2 Information Center installed on your computer or intranet server" topic.

To automatically update the DB2 Information Center installed on your computer or intranet server:

1. On Linux operating systems,
   a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `/opt/ibm/db2ic/V9.7` directory.
   b. Navigate from the installation directory to the `doc/bin` directory.
   c. Run the `ic-update` script:
      ```
      ic-update
      ```
2. On Windows operating systems,
   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 `<Program Files>\IBM\DB2 Information Center\Version 9.7` directory, where `<Program Files>` represents the location of the Program Files directory.
   c. Navigate from the installation directory to the `doc\bin` directory.
   d. Run the `ic-update.bat` file:
      ```
      ic-update.bat
      ```

Results

The DB2 Information Center restarts automatically. If updates were available, the Information Center displays the new and updated topics. If Information Center updates were not available, a message is added to the log. The log file is located in `doc\eclipse\configuration` directory. The log file name is a randomly generated number. For example, `1239053440785.log`.

Manually updating the DB2 Information Center installed on your computer or intranet server

If you have installed the DB2 Information Center locally, you can obtain and install documentation updates from IBM.

About this task

Updating your locally-installed *DB2 Information Center* manually 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 apply updates. The Workstation version of the DB2 Information Center always runs in stand-alone mode. .
2. Use the Update feature to see what updates are available. If there are updates that you must install, you can use the Update feature to obtain and install them

   Note: If your environment requires installing the *DB2 Information Center* updates on a machine that is not connected to the internet, mirror the update site to a local file system using a machine that is connected to the internet and has the *DB2 Information Center* installed. If many users on your network will be

installing the documentation updates, you can reduce the time required for individuals to perform the updates by also mirroring the update site locally and creating a proxy for the update site.
If update packages are available, use the Update feature to get the packages. However, the Update feature is only available in stand-alone mode.

3. Stop the stand-alone Information Center, and restart the *DB2 Information Center* on your computer.

Note: On Windows 2008, Windows Vista (and higher), the commands listed later in this section must be run as an administrator. To open a command prompt or graphical tool with full administrator privileges, right-click the shortcut and then select **Run as administrator**.

Procedure

To update the *DB2 Information Center* installed on your computer or intranet server:

1. Stop the *DB2 Information Center*.
   - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click **DB2 Information Center** service and select **Stop**.
   - On Linux, enter the following command:
     ```
     /etc/init.d/db2icdv97 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 *Program_Files*`\IBM\DB2 Information Center\Version 9.7` directory, where *Program_Files* represents the location of the `Program Files` directory.
     c. Navigate from the installation directory to the `doc\bin` directory.
     d. Run the `help_start.bat` file:
        ```
        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/V9.7` directory.
     b. Navigate from the installation directory to the `doc/bin` directory.
     c. Run the `help_start` script:
        ```
        help_start
        ```

   The systems default Web browser opens to display the stand-alone Information Center.
3. Click the **Update** button ( ). (JavaScript™ must be enabled in your browser.) On the right panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the installation process, check the selections you want to install, then click **Install Updates**.
5. After the installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center:
   - On Windows, navigate to the installation directory's `doc\bin` directory, and run the `help_end.bat` file:

```
help_end.bat
```

Note: The `help_end` batch file contains the commands required to safely stop the processes that were started with the `help_start` batch file. Do not use `Ctrl-C` or any other method to stop `help_start.bat`.

- On Linux, navigate to the installation directory's `doc/bin` directory, and run the `help_end` script:

```
help_end
```

Note: The `help_end` script contains the commands required to safely stop the processes that were started with the `help_start` script. Do not use any other method to stop the `help_start` script.

7. Restart the *DB2 Information Center*.
   - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click **DB2 Information Center** service and select **Start**.
   - On Linux, enter the following command:

```
/etc/init.d/db2icdv97 start
```

Results

The updated *DB2 Information Center* displays the new and updated topics.

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 the title.

"pureXML®" in *pureXML Guide*
Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

"Visual Explain" in *Visual Explain Tutorial*
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 database products.

DB2 documentation
Troubleshooting information can be found in the *DB2 Troubleshooting Guide* or the Database fundamentals section of the *DB2 Information Center*. There you will find information about 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 database 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/support/db2_9/

Terms and Conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal use: You may reproduce these Publications for your personal, non commercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these Publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the Publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. Information about non-IBM products is based on information available at the time of first publication of this document and is subject to change.

IBM may not offer the products, services, or features discussed in this document in other 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 grant 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 character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711 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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Such information may be available, subject to appropriate terms and conditions, including, in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application

programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided ″AS IS″, without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel, Intel logo, Intel Inside®, Intel Inside logo, Intel® Centrino®, Intel Centrino logo, Celeron®, Intel® Xeon®, Intel SpeedStep®, Itanium®, and Pentium® are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft, Windows, Windows NT®, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

Special characters

Numerics

A

B

C

D

E

G

H

I

J

L

M

N

O

P

R

S

T

U

V

X

Printed in USA

SC27-2444-01

Spine information:

IBM DB2 9.7 for Linux, UNIX, and Windows **Version 9 Release 7** **Developing ADO.NET and OLE DB Applications**