IBM Data Virtualization Manager for z/OS
Version 1 Release 1

Developer's Guide

SC27-9302-00

Contents

Figures

Tables

About this information

This information supports IBM Data Virtualization Manager for z/OS (5698-DVM) and contains information about the drivers and APIs that provide connectivity between Data Virtualization Manager and applications.

Purpose of this information

This document provides information about the drivers and APIs that provide connectivity between Data Virtualization Manager and applications.

Who should read this information

This information is intended for application developers and system programmers.

Abstract for IBM Data Virtualization Manager for z/OS Developer's Guide

This information supports IBM Data Virtualization Manager for z/OS (5698-DVM) and contains information about the drivers and APIs that provide connectivity between Data Virtualization Manager and applications.

Purpose of this information

This document provides information about the drivers and APIs that provide connectivity between Data Virtualization Manager and applications.

Who should read this information

This information is intended for application developers and system programmers.

How to send your comments to IBM

We appreciate your input on this documentation. Please provide us with any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

Important: If your comment regards a technical problem, see instead "If you have a technical problem" on page xiii.

Send an email to comments@us.ibm.com.

Include the following information:

- Your name and address
- Your email address
- Your phone or fax number
- The publication title and order number:

  IBM Data Virtualization Manager for z/OS Developer's Guide
  SC27-9302-00
- The topic and page number or URL of the specific information to which your comment relates
- The text of your comment.

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

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are listed for sending comments. Instead, take one or more of the following actions:

- Visit the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

Chapter 1. Overview

To connect between your application and Data Virtualization Manager, you can use the following methods:

- For Java-based applications, you can use the JDBC driver. See Chapter 2, “Spark SQL data access: JDBCThe JDBC driver V3.1,” on page 3.
- For non-Java-based applications, you can use the ODBC driver. See Chapter 3, “The ODBC driver V3.1,” on page 35.
- For high-level language applications, you can use the DS Client high-level API . See Chapter 4, “DS Client high-level API,” on page 55.

What's new in IBM Data Virtualization Manager for z/OS Developer's Guide

This section describes recent technical changes to IBM Data Virtualization Manager for z/OS.

New and changed information is marked like this paragraph, with a vertical bar to the left of a change. Editorial changes that have no technical significance are not marked.

New and changed information is marked like this paragraph, with blue graphics at the beginning and end of the content. Editorial changes that have no technical significance are not marked.

Description	Related APARs
The default value for the JDBC parameter CompressionType and the ODBC parameter Buffer Format (BUFO) is now UNCOMPRESSED. See “JDBC connection properties” on page 5 and “ODBC connection properties” on page 37.	PH03261
When connecting to the Data Virtualization Manager server using the JDBC driver, password phrase authentication is supported. User ID encoding is also supported between the driver and the Data Virtualization Manager server. See “JDBC connection properties” on page 5.	PI92952
The ODBC parameters Connection Timeout Value (CNTM) and Operation Timeout Value (OPTM) are now supported. These parameters are described in “ODBC connection properties” on page 37.	
The DS Client high-level API allows an application running on z/OS to use a call-level interface to communicate with Data Virtualization Manager to process SQL requests and to retrieve result sets. Information about using the API has been added. See Chapter 4, “DS Client high-level API,” on page 55.	

Chapter 2. Spark SQL data access: JDBCThe JDBC driver V3.1

The JDBC driver is a Type 4 driver (written in Java) that is used to implement the network protocol for IBM Data Virtualization Manager for z/OS.

The Java Virtual Machine manages the applications connection to Data Virtualization Manager. Java-based applications and tools use the JDBC driver to access Data Virtualization Manager applications and features. Clients connect directly to Data Virtualization Manager without translation.

Requirements
The driver requires Java 1.7 or higher and it is supplied as a .jar archive file. The components that following are included in the archive file.

The following runtime .jar files are required:

- :`dv-jdbc-[version #].jar`: The driver core implementation file.
- `log4j-api-[version #].jar`: The logging framework API file.
- `log4j-core-[version #].jar`: The logging framework implementation file.

The following sample logging configuration file is included:

- `log4j2.xml`: A sample logging configuration file.

The following command line utilities are included:

Note: Some system-specific environment utilities are available in two formats; Microsoft Windows command script (.cmd) and Bash shell script (.sh).

- `hashpassword.cmd`: Use this utility to generate a hashed format of your text password. You can include the hashed password in your applications connection string or `.ini` file. The script prompts the user for the plain text password. To avoid prompting, the password can be specified as a command-line argument.
- `helpdriver`: Displays the help text, including a detailed list of all supported driver properties.
- `sysinfo.cmd`: Displays local system information.
- `LICENSE.txt`: Provides product licensing information.
- `NOTICE.txt`: Lists notices related to this product.
- `RELEASE-NOTES.txt`: Provides details about optional folders.
- `optional-charsets` folder: Contains additional character sets. Depending on your set up you may need to place these jars either on you classpath or endorsed classpath.
- Java API documentation
- Optional runtime.jars
- `optional-charsets`: Depending on your application, you may need to place these .jars on either your classpath or endorsed classpath.
- `optional-pooling` folder: Contains jar files to pool Data Virtualization Manager data sources. To pool a data source, use:

```
import org.apache.commons.dbcp2.cpdsadapter.DriverAdapterCPDS;
DriverAdapterCPDS driverAdapterCPDS = new DriverAdapterCPDS();
driverAdapterCPDS.setUrl("jdbc:rs:dv:HOST=...");
```

- `optional-logging` folder: Contains optional Log4j jar files for adding additional logging features.

Connecting to a data source using JDBC

A JDBC connection string is used to load the driver and to indicate the settings that are required to establish a connection to the data source. These settings are referred to as *connection properties*. The connection string is a URL with the following format:

```
jdbc:rs:dv://host:port;Key=Value;Key=value;...
```

For example, you can use the following connection string to access virtual tables on your Data Virtualization Manager server, where *host* is the network hostname or IP address:

```
jdbc:rs:dv://host:1200;DatabaseType=DVS;user=userid;password=xxxx
```

If you prefer not to use the `//host:port` syntax, you can use the following string:

```
jdbc:rs:dv:Host=host;Port=1200;DatabaseType=DVS;user=userid;password=xxxx
```

Additional connection properties can be added to influence the behavior of the driver-server communication.

Coding a JDBC application

A JDBC application can establish a connection to the data source using the JDBC `DriverManager` interface, which is part of the `java.sql` package. A connection is created by passing the connection string URL to the `DriverManager.getConnection` method. Alternate forms of this API allow you to specify the user and password as separate parameters, or to specify some or all of the connection properties using the `java.util.Properties` parameter.

Sample Java code fragment:

```
final String url = "jdbc:rs:dv://host:1200; DatabaseType=DVS; user=userid;
password=xxx";
final String sql = "SELECT * FROM MY_VIRTUAL_TABLE";
try (final Connection conn = DriverManager.getConnection(url)) {
   try (final PreparedStatement statement = conn.prepareStatement(sql)) {
      try (final ResultSet rs = statement.executeQuery()) {
         // process the result set
      }
   }
}
```

Coding a Spark application

A Spark application can access a data source using the Spark SQL interface, which is defined in the `org.apache.spark.sql` package namespace. The DataFrameReader interface, obtained via `SparkSession.read`, is used to load data sets from the data source. Spark SQL uses the JDBC driver to connect to the Data Virtualization Manager server and access the virtualized data.

Sample Scala code fragment:

```
val url = "jdbc:rs:dv://host:1200; DatabaseType=DVS; user=userid; password=xxx"
val sql = "MY_VIRTUAL_TABLE"
val spark: SparkSession = ...
val df = spark.read
   .format("jdbc")
   .option("url", url)
   .option("dbtable", sql)
   .load()
// perform data analytics on the dataframe
```

Running on the same sysplex

Applications executing on the same sysplex as the server are not required to specify logon credentials when invoking the JDBC driver. Instead, the application can specify empty string settings for the user and

password to indicate that the current application user ID is to be used. For example, empty values can be specified in the JDBC connection string as follows:

```
…; user=;password=; …
```

When values are specified, the credentials supplied on the driver connection string will be used for server logon instead of the propagated application user ID information.

JDBC connection properties

The JDBC driver supports the following connection properties. If the same property occurs more than once in the connection string, the last entry takes precedence. Property names are not case sensitive.

Table 1. JDBC connection properties

Property names	Description
AdabasColumnNameCorrelationIds Alias: ABCN	Support ADABAS column name correlation IDs. Required: false Default value: false Valid values: [true, false]
ApplicationName Alias: APNA	Application name that is sent to the host as part of logon for connection tracking. Required: false (maximum of 16 characters in length)
AuthenticationMechanism Alias: ATHM	Mechanism for encrypting passwords. Required: false Default value: DEFAULT Valid values: [DEFAULT, AES]
BindInetSocketAddressList Alias: BISAL	A single local name or IP address to bind a socket, or a comma-separated IP address list (used with MapReduce), or `AutoDetect` to detect NIC IP addresses automatically. Required: false Default value: No
CatalogPrefix Alias: CPFX	Database catalog prefix, SYSPROC for DB2, SQLENG for DVS, SDBMAP for the others. Required: false Default value: SYSPROC Valid values: [SYSIBM, SYSPROC, SDBMAP, SQLENG]
CertificateHostName Alias: HostNameInCertificate	Host name for certificate validation when SSL encryption and validation is enabled. Required: false

Table 1. JDBC connection properties (continued)	
Property names	**Description**
Charset Alias: CS, CodePage, CP, Encoding, ENC	The database character encoding. To get a complete list the charsets that are available on a particular JVM, call the Charset.availableCharsets() API, where **charsetName** is one of the available Java character sets. The list of available character sets that are returned depends on the specific version and supplier of Java, as well as the availability of the ICU jar files on the classpath. Names with the 'x-' prefix indicate that a charset is not registered at the Internet Assigned Numbers Authority (IANA). For more information, see https://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html and https://ssl.icu-project.org/icu-bin/convexp. Use the IBM JRE for proper data translation when Charset specifies a Japanese Code page such as; 930, 939, 1390, 1399 or 5026. IBM JRE 1.8 or 1.7 is recommended when accessing IBM-1390 and IBM-1399 mainframe data, and using or exchanging that data in a Unicode environment. IBM JRE 1.8 is recommended when accessing IBM-930 and IBM-939 mainframe data, if you do not experience issues using the IBM-conversion-table-based conversion of the following EBCDIC characters: • X'4260' (Minus sign) • X'444A' (EM Dash) • X'43A1' (Wave Dash) • X'447C' (Double vertical line) • X'426A' (Broken bar) Required: false Default value: IBM037 Valid values: For a list of supported character sets, see “Character sets” on page 18.
CicsTransactionName Alias: TRNA	CICS transaction name. Required: false Default value: (maximum of eight characters in length)
CompressionLevel	Compression level (ZLib only), -1 (default), 1 (best speed) ... 9 (best compression). Required: false Default value: -1 Valid values: [-1, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Table 1. JDBC connection properties (continued)	
Property names	**Description**
CompressionThresholdBytes	Compression threshold in bytes. The driver compresses data for buffers larger than this size. This value can be post-fixed with a unit like KB (K) or MB (M). For example, the following values are all equal: 1048576, 1024 K, 1024 KB, 1 MB. Required: false Default value: 0
CompressionType	The type of compression used. Required: false Default value: UNCOMPRESSED Valid values: [ZLIB_NO_WRAP, CMBU, CMBV, UNCOMPRESSED]
ConnectionType	Connection wire type. Required: false Default value: SOCKET Valid values: [SOCKET, SOCKET_CHANNEL]
CountTraceEnter	Counts JDBC API calls. Required: false Default value: false Valid values: [true, false]
DatabaseRequestModule Alias: DBRM	Database request module. Required: false Default value: OPRXSQ (maximum of eight characters in length)
DatabaseType Alias: DBTY	The database type to connect to after connecting to the Data Virtualization Manager server. Required: false Default value: DRDAorDB2 Valid values: • DRDAorDB2 (the data source type is DB2 on a z/OS subsystem or DB2 LUW database) • DVS (the data source type is determined by the virtual table map definition, and no value is specified for SUBSYS)

Table 1. JDBC connection properties (continued)	
Property names	**Description**
EncodeUserName	When set to true, the user name is encoded when establishing the server connection. Support of this feature can be controlled using server parameters USERIDENCODEREQUIRE and USERIDENCODEALLOW. Required: false Default value: false Valid values: [true, false]
EncryptionMethod	Encryption method. Required: false Default value: NONE Valid values: [NOENCRYPTION, NONE, SSL]
GetTablesSchemaFilter Alias: DP	Filter to use for getTables() DB2 metadata schema. Required: false
HexDumpBytesPerLine	Bytes per line in the hexadecimal dump. Required: false Default value: 16
HexDumpBytesPerWord	Bytes per word in the hexadecimal dump. Required: false Default value: 4
Host	Host name or IP address. Required: true
IniFile Alias: INI	Loads properties from this INI file (overrides IniFileEnvVar). Required: false.
IniFileCharset Alias: INICS	The INI file Charset. Required: false Default value: UTF-8.
IniFileDataSourceName Alias: DSN	Loads properties from a section name to an INI file. Required: false.
IniFileEnvVar Alias: INIEV	Loads properties from the INI file to this environment variable. Required: false. Default value: DV_INI
InitialCurrentDegree Alias: SEDG	Initial current degree (DB2). Required: false Valid values: [ANY, 1]

Table 1. JDBC connection properties (continued)	
Property names	**Description**
InitialCurrentPackageSet SEPK	Initial current package set (DB2). Required: false
InitialCurrentRules Alias: SERL	Initial current rules (DB2). Required: false Valid values: [DB2, STD]
InitializationString	Initialization string, use ';' to separate statements, wrap the string in '(' and ')' when used in a connection string. Required: false
JaasLoginConfigFile Alias: LOGINCFG	Pathname of the JAAS login.conf file (effectively sets the java.security.auth.login.config system property). Required: false
KerberosConfigFile Alias: KRBCFG	Pathname of the krb5.conf file (effectively sets the java.security.krb5.conf system property). Required: false
KerberosKdc Alias: KDC	Effectively sets the java.security.krb5.kdc system property. Required: false
KerberosRealm Alias: KRBREALM	Effectively sets the java.security.krb5.realm system property. Required: false
KerberosServerPrincipal Alias: KSPN	Kerberos Server principal name; required when accessing the Kerberos server to retrieve a token. Required: false
KeyPassword Alias: KP	The SSL key password. Required: false
KeyStore Alias: KS	The SSL keystore. Required: false
KeyStorePassword Alias: KSP	The SSL keystore password. Required: false

Table 1. JDBC connection properties (continued)	
Property names	**Description**
LegacySqlPrepareEnabled Alias: WRPR	This flag controls the behavior of SQLPrepare for non-DB2 data sources such as ADABAS, VSAM, IMSDB, and VSAM CICS. When this keyword is set to true (which is the default), a request is always sent to the host at SQLPrepare time to obtain metadata for the SQL statement. For applications that access non-DB2 data sources and do NOT require metadata after the SQLPrepare, it is recommended to set WRPR to false for better performance since this will eliminate a network roundtrip whenever a SQLPrepare is executed. Required: false Default value: true Valid values: [true, false]
LGID Alias: LanguageID	This setting is for backward compatibility. It is recommended to use Charset instead. Each language code corresponds to a charset used for byte conversion. If a key has no value, the driver will use the default. The mappings are as follows: {ARB=IBM420, CHS=, CHT=, DAN=IBM01142, DEU=IBM01141, DFT=IBM037, ENC=IBM1047, ENG=IBM285, ENU=IBM037, ESN=IBM01145, ESP=IBM284, FIN=IBM01143, FRA=IBM01147, FRC=IBM037, ISL=IBM01149, ITA=IBM01144, JNL=IBM1390, JNX=IBM1399, JPL=IBM5026, JPX=IBM5035, KOR=IBM037, KRN=x-IBM833, MDI=, NGN=IBM01142, NLD=IBM037, NOR=IBM01142, PPS=, PTG=IBM037, SVE=IBM278, SWE=IBM01143, TUR=IBM1026} Required: false Default value: ENU Valid values (31): [ARB, CHS, CHT, DAN, DEU, DFT, ENC, ENG, ENU, ESN, ESP, FIN, FRA, FRC, ISL, ITA, JNL, JNX, JPL, JPX, KOR, KRN, MDI, NGN, NLD, NOR, PPS, PTG, SVE, SWE, TUR]
LogConfiguration Alias: LOGCONFIG	Sets the Log4j 2 configuration file. Required: false
LogThreadsState Alias: LTS	Logs the state of a thread. Required: false Default value: false Valid values: [true, false]
LogThreadsStatePeriodMillis	Logs the state period for a thread in milliseconds. Required: false Default value: 1000

Table 1. JDBC connection properties (continued)	
Property names	**Description**
LoginTimeoutMillis	Login timeout in milliseconds (0 = system default if there is one, or no limit.) Required: false Default value: 0
LoginTimeoutSeconds Alias: LOGINTIMEOUT	Login timeout in seconds. Required: false Default value: 0
MapReduceClient Alias: MRC, MapReduce	Use MapReduceClient (MRC) to read query results in parallel from different connections. When enabled, the driver creates one master connection and **N** worker connections (instances of JDBC connections). Required: false Default value: false Valid values: [true, false, list] To distribute MapReduce on a single server, select from the following methods: • Set MapReduceClient to true (`MRC=true`). The MapReduceConnectionCount defaults to the number of CPU cores discovered (for example: `MRCC=10`) • Set `MRC=(host, port, taskCount)` To distribute MapReduceClient over multiple servers, set MapReduceClient: `MCR=(host1, port1, taskCount1), (host2, port2, taskCount2),...` If you are using MapReduceClient with RDBMS or IMS, you must complete the metadata repository configuration requirements. See "MapReduce" in the *Administrator's Guide*.
MapReduceClientCount Alias: MRCC	The single-connection MapReduceClientCount. This value is only set for use with MRCN. The MRCC default setting is 0. Required: false Default value: 0

Table 1. JDBC connection properties (continued)

Property names	Description

MapReduceClientNumber
Alias: MRCN
	To enable highly-parallel client applications to control concurrent MapReduce connections, from which queries are executed as a single thread, set the MapReduceClientCount (MRCN) value for each MapReduceClientCount (MRCC).
	The following example shows a connection string used to connect to the first of four available connections:
	`jdbc:rs:dv://host:port;DBTY=DVS;`
	`PWD=xxx;UID=xxx;`
	`MXBU=4MB;`
	`MapReduceClientCount=4;`
	`MapReduceClientNumber=1`
	To disable this feature, set MapReduceClientCount to 0 (`MRCC=0`) and set MapReduceClient to false (`MRC=false`).

MapReduceFillValueMaximumInitialSize Alias: MRFVMIS	This value specifies the initial capacity of the result row pre-fetch cache for a given buffer that is used after the MapReduce or Parallel IO read queue exceeds the value set for MapReduceFillValueThreshold. Required: false Default value: 20,000 (rows per buffer)
MapReduceFillValueThreshold Alias: MRFVT	When using Parallel IO or MapReduce, this value specifies the number of buffers on the parallel IO thread that must be exceeded before pre-filling the column values (converting mainframe row bytes to Java object representations). Required: false Default value: -1 (off).
MapReducePollTimeOutMillis Alias: MRPTO	The MapReduce and Parallel IO inter-thread poll timeout, in milliseconds. Required: false Default value: 50
MapReduceQueueStats Alias: MRQS	Gathers statistics for MapReduce result sets. Required: false Default value: false Valid values: [true, false]
MaximumBufferSize Alias: MXBU	Maximum server-side communication buffer size in bytes. This value can be post-fixed with a unit like KB (K) or MB (M). For example, the following values are all equal: 1048576, 1024 K, 1024 KB, 1 MB. The minimum value is 40960. Required: false Default value: 262144

Table 1. JDBC connection properties (continued)	
Property names	**Description**
MaximumFieldSize Aliases: MFS, MaxFieldSize	Maximum field size to return. Required: false Default value: 2147483647
MaximumRows Alias: MR	Maximum number of rows to return. Required: false Default value: 0
MetaDataCharset Alias: MDCS	The database metadata character encoding. Available encodings depends on the runtime. See the **Charset** property. Required: false Default value: IBM037
NetworkTimeoutMillis Alias: NTOM	Network timeout in milliseconds. Required: false Default value: 0
NetworkTimeoutSeconds Alias: NETWORKTIMEOUT	Network timeout in seconds. Required: false Default value: 0
ParrallelIoBufferCount Alias: PIOBC	When the value of the Parallel IO buffer count is > 0 (where the buffer size = MXBU), a background thread reads the rows from the server as the main thread consumes them. This memory is re-usable. For example, if MXBU is set to 4 MB and PIOBC is set to 10, the driver uses 40 MB of memory as the read-ahead buffer (10 x 4 MB buffers). Required: false. Default value: 0.
Password Alias: PWD, PassPhrase	Password or password phrase. Required: false
PasswordToUpperCase Alias: UCLC	Converts passwords to uppercase. This property is ignored when a password phrase has been specified. Required: false Default value: true Valid values: [true, false]
Plan	The DB2 plan name used for a DB2 connection. This is used when the subsystem is set to a valid DB2 subsystem name. Required: false Default value: SDBC1010 (maximum of eight characters in length)

Table 1. JDBC connection properties (continued)

Property names	Description
Port	Server port. Required: false Default value: 1200
PrepareMetadataSQL	Determines how the SQL statement metadata calls are handled. Required: false Default value: false Valid values: [true, false] If set to true, the driver prepares the SQL statements for metadata calls. If set to false, the driver builds and executes the SQL statements for metadata calls dynamically, which can be susceptible to SQL injection attacks.
QueryTimeoutMillis Alias: QTOM	Query timeout in milliseconds. Required: false Default value: 0
QueryTimeoutSeconds Alias: QUERYTIMEOUT	Query timeout in seconds. Required: false Default value: 0
SelectForReadOnly Alias: RO	Appends FOR FETCH ONLY to JDBC read-only SQL query statements. Required: false Default value: true Valid values: [true, false]
ServerCertificateStrategy	Server certificate strategy. Required: false Default value: Validate Valid values: [Validate, AcceptSelfSigned, Trust]
ShadowServerCompatibility	Enable compatibility with older Shadow servers. Required: false
SocketKeepAlive Alias: SKA	Socket keep alive. Required: false Default value: false Valid values: [true, false]

Table 1. JDBC connection properties (continued)	
Property names	**Description**
SocketReceiveBufferSize Alias: SRBS	Socket receive buffer size hint in bytes. This value can be post-fixed with a unit like KB (K) or MB (M). For example, the following values are all equal: 1048576, 1024 K, 1024 KB, 1 MB. Required: false Default value: 0
SocketSendBufferSize Alias: SSBS	Socket send buffer size hint in bytes. This value can be post-fixed with a unit like KB (K) or MB (M). For example, the following values are all equal: 1048576, 1024 K, 1024 KB, 1 MB. Required: false Default value: 0
SocketTcpNoDelay Alias: STCPND	Socket TCP NoDelay. Required: false Default value: false Valid values: [true, false]
SqlAuthorizationId Alias: ALUS	SQL authorization ID. Required: false
SslContextProtocol Alias: SSLPROTOCOL	SSL context protocol. Required: false Default value: TLS
StrictJdbcCompliance	Indicates whether the driver complies strictly with the JDBC spec. Required: false Default value: false Valid values: [true, false]
SubSystem Alias: SUBSYS	The database subsystem name. If the data source is DB2 (DatabaseType=DRDAorDB2), enter the DB2 subsystem name as it exists on the mainframe. If the data source is a non-DB2 database (where TYPE must be Member, Group, or LUW), enter the name as it exists in the NAME field of the DEFINE entry in the server configuration member, AVZSIN00. If the data source is DVS, the subsystem name should be set to NONE. Required: false Default value: NONE Valid values: 1 - 4 alphanumeric characters

Table 1. JDBC connection properties (continued)

Property names	Description
ThrowUnsupportedAll	[Development tool] Throws an exception when an API is not supported. Required: false Default value: true Valid values: [true, false]

TraceBrowseAppender	To collect JDBC driver server trace log information, add the name of the collection appender to the log4j configuration file.
	The following example shows the collection name tag:
	`<Appenders>`
	`... other appenders here ...`
	`<Collection name="TB">`
	`<PatternLayout>`
	`<Pattern>%d %-5level [%t][%logger] %msg%n%throwable</Pattern>`
	`</PatternLayout>`
	`</Collection>`
	`</Appenders>`
	`<Loggers>`
	`<Root level="...">`
	`... other appenders here ...`
	`<AppenderRef ref="TB" />`
	`</Root>`
	Where the name of the collection appender (TB in this example) is the name you choose.
	Add the `TraceBrowseAppender` setting to the JDBC connect string to provide the collection appender name in the log4j2.xml config file to the server. For example, `TraceBrowseAppender=TB`.
	The server parameter `TRACE EXTERNAL TRACE DATA (TRACEEXTERNTRACEDATA)` must also be enabled.
	Required: false

TruncateCallLiteral Alias: TRLT	Truncate CALL string. Required: false Default value: true Valid values: [true, false]
TrustStore	The SSL trust store. Required: false
TrustStorePassword	The SSL trust store password. Required: false

Table 1. JDBC connection properties (continued)

Property names	Description
UpperCaseAllCharacters	This field controls if all character data sent to the host should be converted to upper case or not. If this field is set to true, then all character data will be converted to upper case. If this field is set to false, then character data will not be converted to upper case. Required: false Default value: false Valid values: [true, false]
UpperCaseNonLiterals	This flag controls if all non-literal values in SQL statements passed to the driver should be converted to upper case or not. If this flag is set to true, then strings not in single or double quotes will be converted to upper case. Required: false Default value: true Valid values: [true, false]
URL	Use to specify connection properties. Required: false
user Alias: UID	The user name or identifier.
UserParm Alias: UserParm	Sent in the logon information to complete logon to a host security system and/or database. Maximum of 100 characters in length.
ValidateServerCertificate	Validate server certificate or not. Required: false Default value: true Valid values: [true, false]
VpdGroupMemberCount	Virtual Parallel Data group member count. Required: false Default value: 0-255
VpdGroupName	Virtual Parallel Data group name. Maximum of eight characters in length. Required: false Default value:
VpdGroupOpenTimeoutSeconds	Virtual Parallel Data group open timeout in seconds. Required: false Default value: 0

Table 1. JDBC connection properties (continued)	
Property names	**Description**
VpdIoThreadCount	Virtual Parallel Data I/O thread count. Required: false Default value: 0-255
XaEnabled Alias: XAEN	Set to true to enable XA transactions. Required: false Default value: false Valid values: [true, false]
XaTransactionManager Alias: XAOP	The type of transaction manager used for XA operations. Required: false Valid value: JTS

Character sets

The list of available character sets that are returned depends on the specific version and supplier of Java, as well as the availability of the ICU jar files on the classpath.

The following Charsets (CS) are supported:

Charsets
Adobe-Standard-Encoding, Big5, Big5-HKSCS, BOCU-1, CESU-8, cp1363, cp851, EUC-JP, EUC-KR, GB18030, GB2312, GB_2312-80, GBK, hp-roman8, HZ-GB-2312, IBM-Thai, IBM00858, IBM01140, IBM01141, IBM01142, IBM01143, IBM01144, IBM01145, IBM01146, IBM01147, IBM01148, IBM01149, IBM037, IBM1026, IBM1047, IBM273, IBM277, IBM278, IBM280, IBM284, IBM285, IBM290, IBM297, IBM420, IBM424, IBM437, IBM500, IBM775, IBM850, IBM852, IBM855, IBM857, IBM860, IBM861, IBM862, IBM863, IBM864, IBM865, IBM866, IBM868, IBM869, IBM870, IBM871, IBM918, ISO-2022-CN, ISO-2022-CN-EXT, ISO-2022-JP, ISO-2022-JP-1, ISO-2022-JP-2, ISO-2022-KR, ISO-8859-1, ISO-8859-10, ISO-8859-13, ISO-8859-14, ISO-8859-15, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9, JIS_X0201, JIS_X0212-1990, KOI8-R, KOI8-U, KSC_5601, macintosh, SCSU, Shift_JIS, TIS-620, US-ASCII, UTF-16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE, UTF-32LE, UTF-7, UTF-8, windows-1250, windows-1251, windows-1252, windows-1253, windows-1254, windows-1255, windows-1256, windows-1257, windows-1258, windows-31j,

Charsets
x-Big5-HKSCS-2001, x-Big5-Solaris, x-compound-text, x-ebcdic-xml-us, x-euc-jp-linux, x-EUC-TW, x-euc-tw-2014, x-eucJP-Open, x-ibm-1047-s390, x-ibm-1125_P100-1997, x-ibm-1129_P100-1997, x-ibm-1130_P100-1997, x-ibm-1131_P100-1997, x-ibm-1132_P100-1998, x-ibm-1133_P100-1997, x-ibm-1137_P100-1999, x-ibm-1140-s390, x-ibm-1141-s390, x-ibm-1142-s390, x-ibm-1143-s390, x-ibm-1144-s390, x-ibm-1145-s390, x-ibm-1146-s390, x-ibm-1147-s390, x-ibm-1148-s390, x-ibm-1149-s390, x-ibm-1153-s390, x-ibm-1154_P100-1999, x-ibm-1155_P100-1999, x-ibm-1156_P100-1999, x-ibm-1157_P100-1999, x-ibm-1158_P100-1999, x-ibm-1160_P100-1999, x-ibm-1162_P100-1999, x-ibm-1164_P100-1999, x-ibm-1250_P100-1995, x-ibm-1251_P100-1995, x-ibm-1252_P100-2000, x-ibm-1253_P100-1995, x-ibm-1254_P100-1995, x-ibm-1255_P100-1995, x-ibm-1256_P110-1997, x-ibm-1257_P100-1995, x-ibm-1258_P100-1997, x-ibm-12712-s390, x-ibm-12712_P100-1998, x-ibm-1373_P100-2002, x-ibm-1386_P100-2001, x-ibm-16684_P110-2003, x-ibm-16804-s390, x-ibm-16804_X110-1999, x-ibm-25546, x-ibm-33722_P12A_P12A-2009_U2, x-ibm-37-s390, x-ibm-4517_P100-2005, x-ibm-4899_P100-1998, x-ibm-4909_P100-1999, x-ibm-4971_P100-1999, x-ibm-5123_P100-1999, x-ibm-5351_P100-1998, x-ibm-5352_P100-1998, x-ibm-5353_P100-1998, x-ibm-803_P100-1999, x-ibm-813_P100-1995, x-ibm-8482_P100-1999, x-ibm-901_P100-1999, x-ibm-902_P100-1999, x-ibm-9067_X100-2005, x-ibm-916_P100-1995, x-IBM1006, x-IBM1025, x-IBM1046, x-IBM1097, x-IBM1098, x-IBM1112, x-IBM1122, x-IBM1123, x-IBM1124, x-IBM1153, x-IBM1363, x-IBM1364, x-IBM1371, x-IBM1381, x-IBM1383, x-IBM1388, x-IBM1390, x-IBM1399, x-IBM300, x-IBM33722, x-IBM720, x-IBM737, x-IBM833, x-IBM834, x-IBM856, x-IBM867, x-IBM874, x-IBM875, x-IBM921, x-IBM922, x-IBM930, x-IBM933, x-IBM935, x-IBM937, x-IBM939, x-IBM942, x-IBM942C, x-IBM943, x-IBM943C, x-IBM948, x-IBM949, x-IBM949C, x-IBM950, x-IBM954, x-IBM964, x-IBM970, x-IBM971, x-IMAP-mailbox-name, x-iscii-be, x-iscii-gu, x-iscii-ka, x-iscii-ma, x-iscii-or, x-iscii-pa, x-iscii-ta, x-iscii-te, x-ISCII91, x-ISO-2022-CN-CNS, x-ISO-2022-CN-GB, x-iso-8859-11, x-JIS0208, x-JIS7, x-JIS8, x-JISAutoDetect, x-Johab, x-LMBCS-1, x-mac-centraleurroman, x-mac-cyrillic, x-mac-greek, x-mac-turkish, x-MacArabic, x-MacCentralEurope, x-MacCroatian, x-MacCyrillic, x-MacDingbat, x-MacGreek, x-MacHebrew, x-MacIceland, x-MacRoman, x-MacRomania, x-MacSymbol, x-MacThai, x-MacTurkish, x-MacUkraine, x-MS932_0213, x-MS950-HKSCS, x-MS950-HKSCS-XP, x-mswin-936, x-PCK, x-SJIS_0213, x-UnicodeBig, x-UTF-16LE-BOM, X-UTF-32BE-BOM, X-UTF-32LE-BOM, x-UTF16_OppositeEndian, x-UTF16_PlatformEndian, x-UTF32_OppositeEndian, x-UTF32_PlatformEndian, x-windows-50220, x-windows-50221, x-windows-874, x-windows-949, x-windows-950, x-windows-iso2022jp]

Error handling

The JDBC driver reports errors to an application by throwing SQL exceptions, each of which contains the following details.

- Description of the likely cause of the error, prefixed by the component that generated the error.
- Native error code, if applicable.

All errors, including those generated by the driver framework or the driver layer, have the following format:

```
[DV][JDBC Driver][version #]message
```

To fully interpret the error message, it may be necessary to refer to the JDBC documentation for possible resolutions, or refer to the database management system documentation to interpret error codes.

Enable JDBC driver logging in the Apache Log4j configuration file.

About this task

The JDBC driver implements logging by using Apache Log4j2 (https://logging.apache.org/log4j/2.x/).

Note: To include JDBC driver logging information in the Studio **Server Trace** pane, enable the JDBC **`TraceBrowseAppender`** property.

Procedure

Open the Log4j configuration file (https://logging.apache.org/log4j/2.x/manual/configuration.html) and choose one of the following methods to implement logging:

- Add the directory and folder location to the **CLASSPATH**
- Set the **`LogConfiguration`** connection property to a file path or URI that points to a Log4j configuration file

The following is a sample `log4j2.xml` configuration file that is included in the JDBC driver installation ZIP file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN" monitorInterval="5">
    <Appenders>
        <Console name="Console" …>
            <PatternLayout … />
        </Console>
        <RollingRandomAccessFile name="RollingFile" …>
          <PatternLayout>
            <Pattern>…</Pattern>
            </PatternLayout>
            <Policies>
              <SizeBasedTriggeringPolicy … />
        </Policies>
       </RollingRandomAccessFile>
    </Appenders>
    <Loggers>
        <Root level="WARN">
            <AppenderRef ref="Console" />
            <AppenderRef ref="RollingFile" />
        </Root>
        <Logger name="com.rs.jdbc.dv.DvDataSource"
                level="WARN" />
        <Logger name="com.rs.jdbc.dv.log.DvThreadsStateLogger"
                level="WARN" />
        <Logger name="javax.management"
                level="WARN" />
        <Logger name="sun.rmi"
                level="WARN" />
        <Logger name="com.rs.dv.util.LoggerEnvironmentReporter"
                level="OFF" />
    </Loggers>
</Configuration>
```

The sample configuration file does not log anything if the status of the entire configuration and level of the root logger are set to WARN (`<Configuration status=="WARN">;<Root level="WARN">`). Setting the configuration's level to TRACE causes Log4j to log about itself being configured; this is useful when debugging the configuration file is necessary. Setting the root logger level to DEBUG causes the driver to log more detailed information, and setting the level to TRACE will allow the tracing of JDBC API entries and exits. For more information, refer to the following URL:

```
https://logging.apache.org/log4j/2.x/manual/flowtracing.html
```

Each driver implementation class has a logger. The logger name is derived from the fully qualified class name. The logging levels for each of these loggers may be changed independently.

By default, setting the root level to TRACE results in logging hexadecimal dumps of the buffers. To enable API logging while disabling buffer tracing, use a Log4j filter.

The following example shows how to define the filter DV.BUFFER to the JDBC configuration file:

```
<Loggers>
…
    <Filters>
        <MarkerFilter marker="DV.BUFFER"
                      onMatch="DENY"  onMismatch="NEUTRAL" />
    </Filters>
</Loggers>
```

Filtering may also be applied to the communication and SQL operations by setting appropriate values for the marker attribute. You may choose from the following filters:

- DV.BUFFER
  - DV.BUFFER.HEADER
  - DV.BUFFER.PARAM
  - DV.BUFFER.COMPRESSED
  - DV.BUFFER.DECOMPRESSED
  - DV.BUFFER.UNCOMPRESSED
  - DV.BUFFER.DESCRIPTION
  - DV.BUFFER.DESCRIPTION.CONNECT
- DV.COMM
  - DV.COMM.SOCKET
- DV.SQL
  - DV.SQL.PREPARE.USER
  - DV.SQL.BATCH.USER
  - DV.SQL.BATCH.SERVER
  - DV.SQL.QUERY.USER
  - DV.SQL.QUERY.SERVER
  - DV.SQL.QUERY.POST
  - DV.SQL.UPDATE.USER
  - DV.SQL.UPDATE.SERVER

Connecting to a DRDA database server

To connect to a DRDA (DB2 and Oracle) database server, include the application information on the system CLASSPATH.

About this task

To connect to the database, you need add the CLASSPATH to your system. In your application, you need to define the connection string and the connection.

Procedure

1. Add the CLASSPATH.

   To load the JDBC driver, the Java Virtual Machine requires that the location and name of the driver and Log4j implementation files be included in the system CLASSPATH. For example:

   ```
   install_dir/lib/dv-jdbc-[version #].jar
   install_dir/lib/log4j-api-[version #].jar
   ```

```
install_dir/lib/log4j-core-[version #].jar
install_dir/lib/
```

All jar files in the `lib` folder must be in the CLASSPATH, and the folder that contains the `log4j2.xml` file must be in the CLASSPATH.

On the Windows command line, you can choose to include all files in the `install_dir/lib`, or you can choose to list each jar file that you want to include. The following example shows how to include all files:

```
classpath install_dir/lib/*;install_dir/lib;
codeph;
```

On *Nix:

```
classpath install_dir/lib/*:install_dir/lib
```

To list each jar file individually, rather than using `install_dir/lib/*`, pass the database connection information to the driver manager in the form of a connection URL. Using the following URL format as an example, you can substitute values that are specific to your database management system:

```
jdbc:rs:dv://host-name:port-number[;property=value[;…]]
```

Substitute the `host-name` with the IP address or computer name of the server that hosts the database management system. Substitute the `port-number` with the TCP/IP port number on which the Data Virtualization Manager server listens for incoming requests. Multiple connection properties must be separated by semicolons.

Note: On the command line on Windows, if a property value contains a semi-colon, you must enclose the whole connection string in double quotation marks. On *Nix, the same is true for the colon character.

2. Define the connection string and the connection in your application.

   The JDBC driver uses the JDBC Driver Manager to connect to a database server. To connect to the database management system, use the `DriverManager.getConnection()` method to pass the connection URL as the argument, as shown in the following example:

   ```
   Connection conn = DriverManager.getConnection(
   "jdbc:rs:dv://host-name:port-number;
   DatabaseType=DRDAorDB2;
   user=user-id;password=user-password;
   subsystem=my-subsystem;
   Charset=CharsetName");
   ```

   Note: Different database management systems have different required and optional connection properties. You must understand the DBMS to which you want to connect in order to properly configure the JDBC driver.

   By default, the **DatabaseType** property is **DRDAorDB2**, which defaults the **subsystem** value to **NONE**. To specify your subsystem, set the value accordingly.

   The **Charset** must match the **SQLENGDFLTCCSID** parameter setting of the Data Virtualization Manager server, and it must be compatible with the **MCCSID** of the target DB2 subsystem.

   To connect to the Data Virtualization Manager server, use the following syntax:

   ```
   Connection conn = DriverManager.getConnection(
   "jdbc:rs:dv://host-name:port-number;
   DatabaseType=DVS;
   user=user-id;password=user-password;"
   Charset=CharsetName");
   ```

   You can use the `hashpassword.cmd` utility to generate a hashed format of your text password. You can include the hashed password in the connection string or `.ini` file for your application. The script

prompts the user for the plain text password. To avoid prompting, the password can be specified as a command-line argument.

For information on interface methods, see “JDBC driver APIs” on page 29

JDBC performance management

This section describes how you can improve JDBC driver performance when sending and receiving data to and from the client.

To optimize JDBC driver performance, choose from the following options:

Buffering data

When sending large amounts of data to the client or the server, one way to optimize performance is by choosing the appropriate size that the driver uses to divide and send the buffers of data.

The JDBC driver communicates with a server using the Communication Buffer (CMBU) protocol. CMBU specifies that all communications occur with one or more buffers. The maximum size of a buffer is set using the **MaximumBufferSize(MXBU)** JDBC property and the **NETWORKBUFFERSIZE** parameter in the server configuration file. You can use the **MaximumBufferSize** JDBC property as a basic building block for controlling performance.

Network latency that occurs between the client and the server can have a negative impact on performance. When setting the **MaximumBufferSize** value, consider the distance between the client and the server. The buffer size that works best for clients and servers that are closer in proximity (low latency), may not be the buffer size that works best for clients and servers that are not close in proxmity (high latency).

When you execute a large SQL INSERT statement, or a batch of INSERT statements, the statements are divided into multiple data buffers that are no larger than the size you specify for **MaximumBufferSize**. The buffers are then sent to the server. The following illustration shows the buffers of INSERT statements being sent to the server:

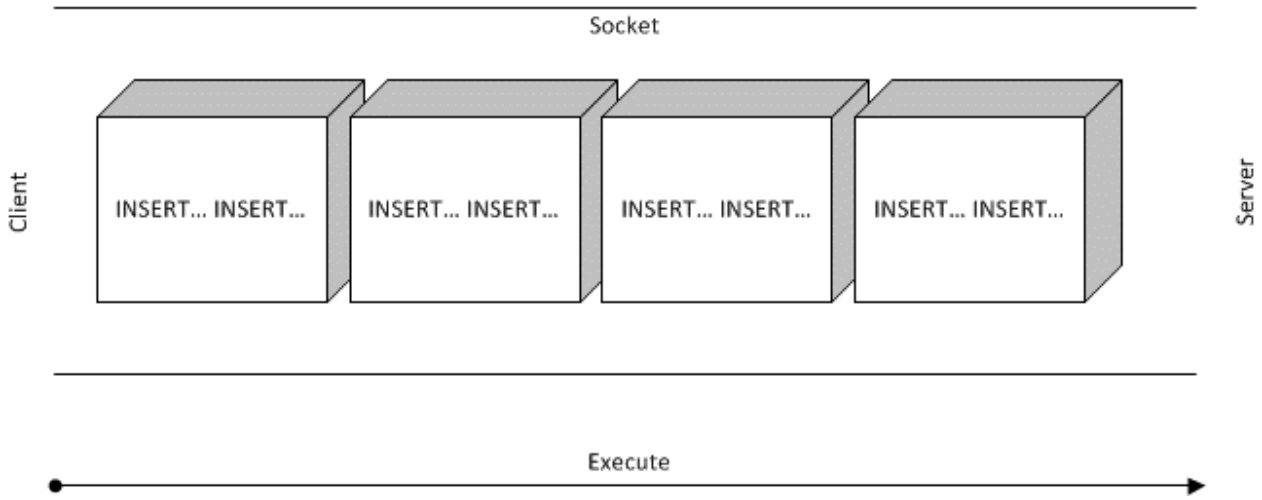


Figure 1. JDBC driver sends data requests using multiple buffers

The actual **MaximumBufferSize** value that the JDBC driver uses is the result of a handshake between the driver and the server. When the driver logs on to the server, it requests that the server support the given **MaximumBufferSize**. In the logon reply, the server tells the driver the actual buffer size value that the driver must use (**NETWORKBUFFERSIZE**). This value may be less than the driver requested value.

After executing a SQL SELECT statement, the server divides and returns the requested rows of data in buffers that have been sized appropriately, one buffer at a time. The following illustration shows the server sending the client rows of data in a single buffer:

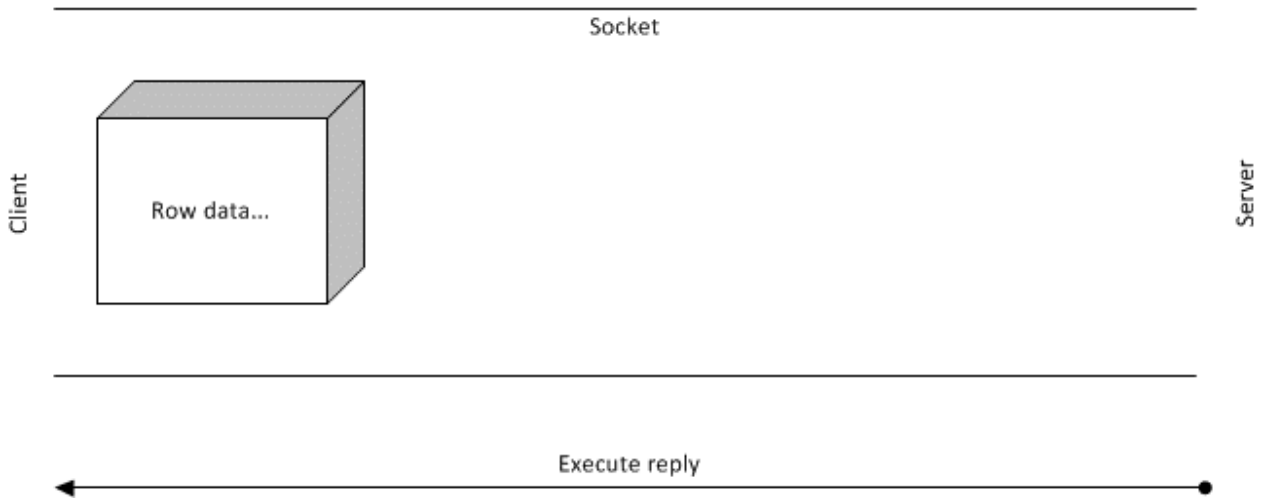


Figure 2. Server returns data one buffer at a time

The client can call **next()** on the result set until all rows are read. When the **next()** call no longer has a row to read from the buffer, the driver issues a request to the server for another row buffer. This cycle continues until the client reads all rows from all buffers. This happens transparently to the JDBC call site. The following code sample shows how this is implemented:

```
Connection con = DriverManager.getConnection(
             "jdbc:rs:dv://host:port;Key=Value;Key=Value;",
                            username,
                            password);
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");

            while (rs.next()) {
              int x = rs.getInt("a");
              String s = rs.getString("b");
              float f = rs.getFloat("c");
            }
```

During certain **next()** API calls, the client can experience a pause while the driver fetches another buffer of row data. To remove this pause, enable parallel IO buffering by setting the ParallelIoBufferCount (PIOBC) JDBC driver property.

Parallel IO

To improve performance, enable Parallel IO.

When Parallel IO is enabled, the JDBC driver creates a separate parallel IO thread from which a given number of buffers that are sent from the server are pre-fetched, read, and placed in a queue. The driver tries to keep the queue as full as possible so that there is always at least one buffer ready for the **next()** API call to use. This eliminates the pause that can occur as a result of certain **next()** calls. The client continues to call the **next()** buffer in the results set, until all rows in all buffers are read and queued.

The **ParallelIoBufferCount** (PIOBC) property determines the number of buffers to pre-fetch on the parallel IO thread. The illustration that follows, shows row data being sent to the client using both the main and the parallel thread.

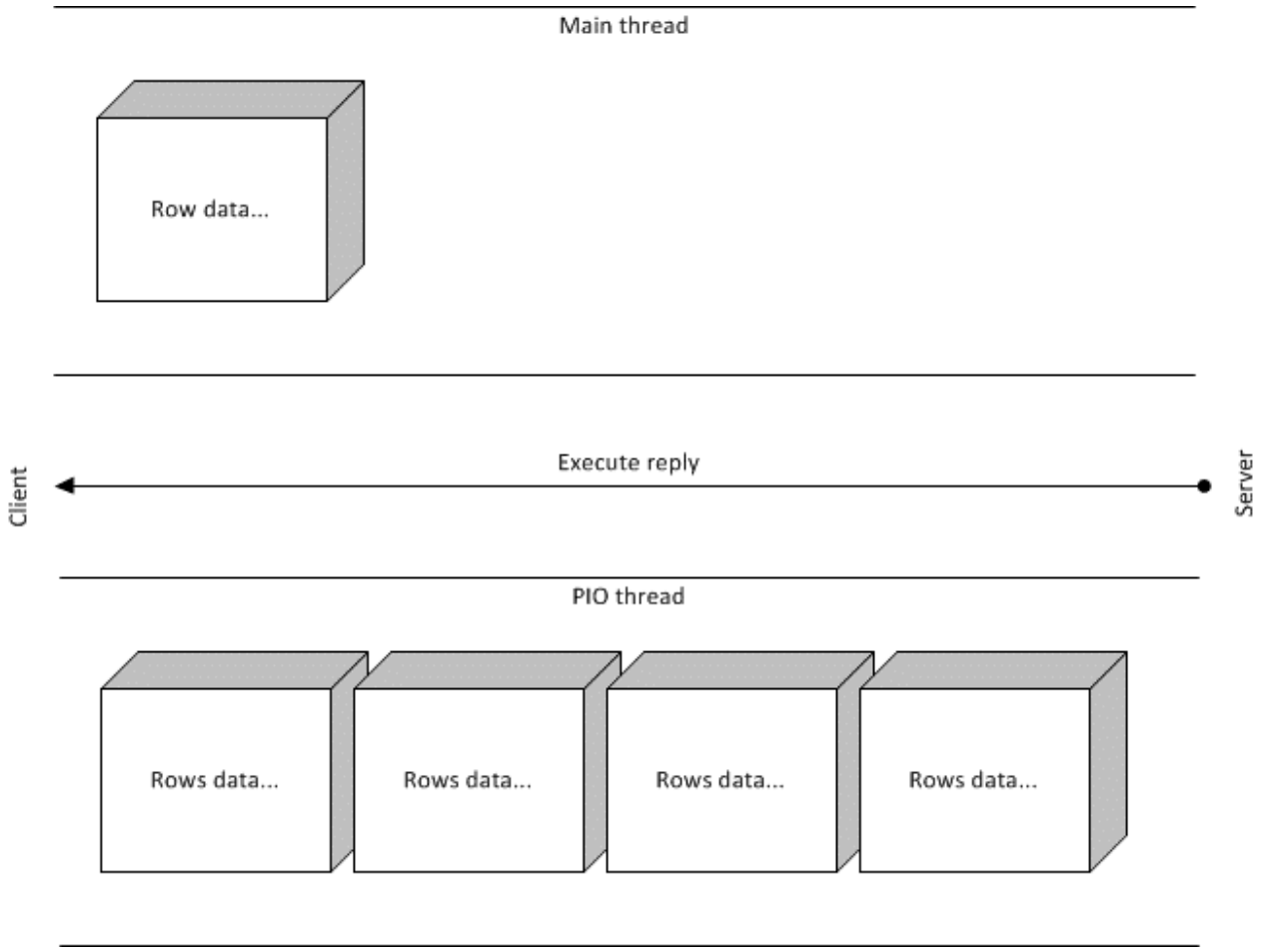


Figure 3. Parallel IO pre-fetches buffers

Materializing row bytes into Java objects on the thread

In addition to prefetching buffers, the Parallel IO thread can be used to materialize row data into their expected Java object representations, as illustrated in the following figure:

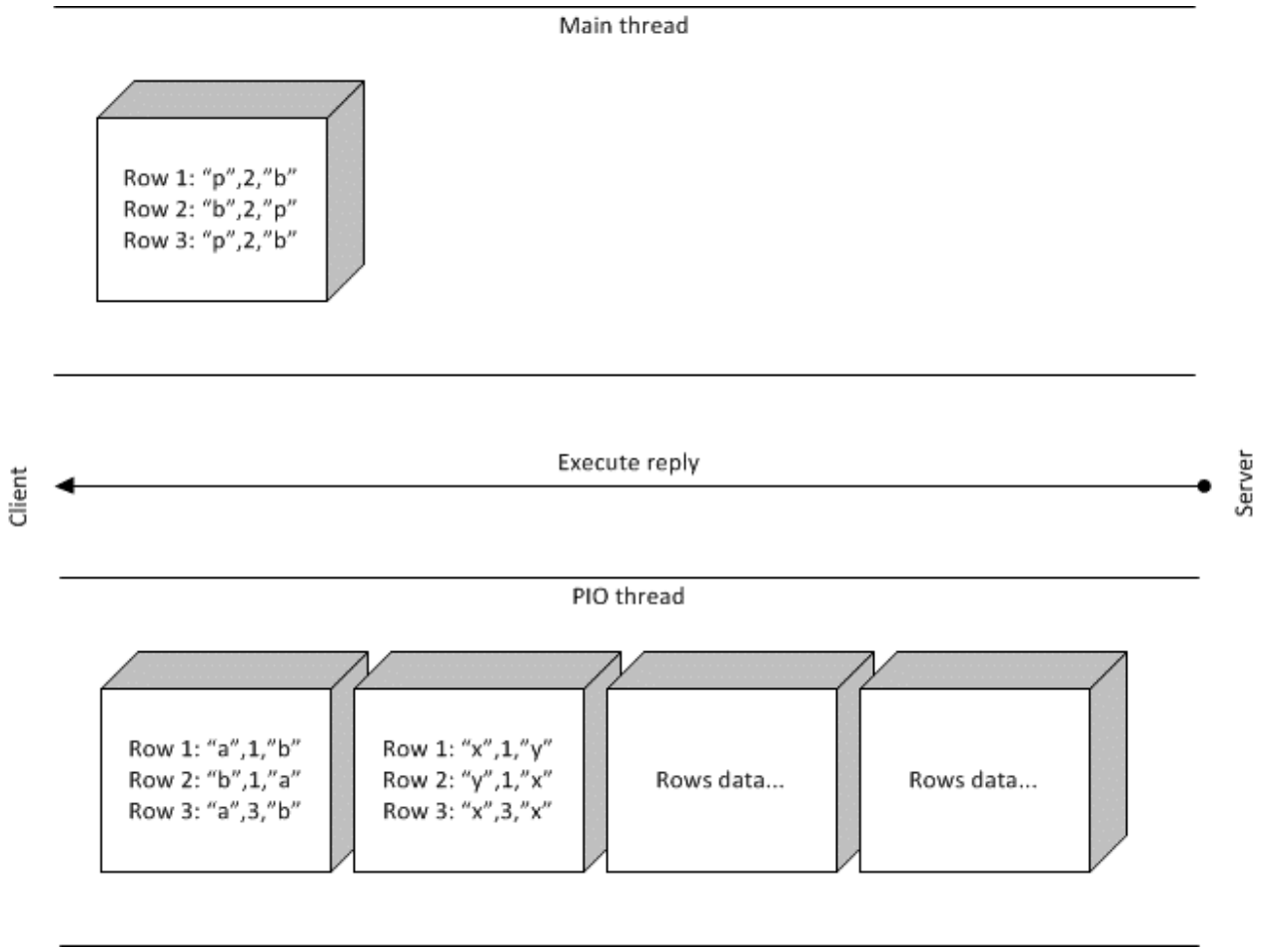


Figure 4. Materializing prefetched row data

The following sample code can be used to convert mainframe bytes to Java objects on the Parallel IO thread. The schema assumes that when the main thread calls a get API on the result set for a given column, that it will use the same get API on subsequent rows; for example **getString()**.

```
Connection con = DriverManager.getConnection(
                         "jdbc:myDriver:myDatabase",
                         username,
                         password);

         Statement stmt = con.createStatement();
         ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");

         while (rs.next()) {
            int x = rs.getInt("a");
            String s = rs.getString("b");
            float f = rs.getFloat("c");
         }
```

The **MapReduceFillValueThreshold** (MRFVT) and **MapReduceFillValueMaximumInitialSize** (MRFVMIS) JDBC property settings govern this behavior. When the MapReduce or Parallel IO read queue fills up beyond the **MapReduceFillValueThreshold** size, the driver materializes the column values. The default is off (-1) and a reasonable experimental value is two buffers.

When the MR or PIO read queue fills up beyond the **MapReduceFillValueThreshold** value, the **MapReduceFillValueMaximumInitialSize** row count is the initial capacity of the result row pre-fetch cache for a given buffer. The arbitrary default is 20,000 rows per buffer and it grows dynamically from there.

MapReduce

MapReduce is a the JDBC driver-controlled feature that is used to improve performance by reading results from the server faster.

Choose from the following MapReduce options:

- Server-controlled MapReduce – MapReduce is performed on the server using the JDBC driver.
- Client-controlled MapReduce – MapReduce is performed on the client and the JDBC driver is used for a single connection.

If you are using MapReduce with RDBMS or IMS, you must complete the meta data repository configuration requirements. See "MapReduce" in the *Administration Guide*.

Server controlled MapReduce

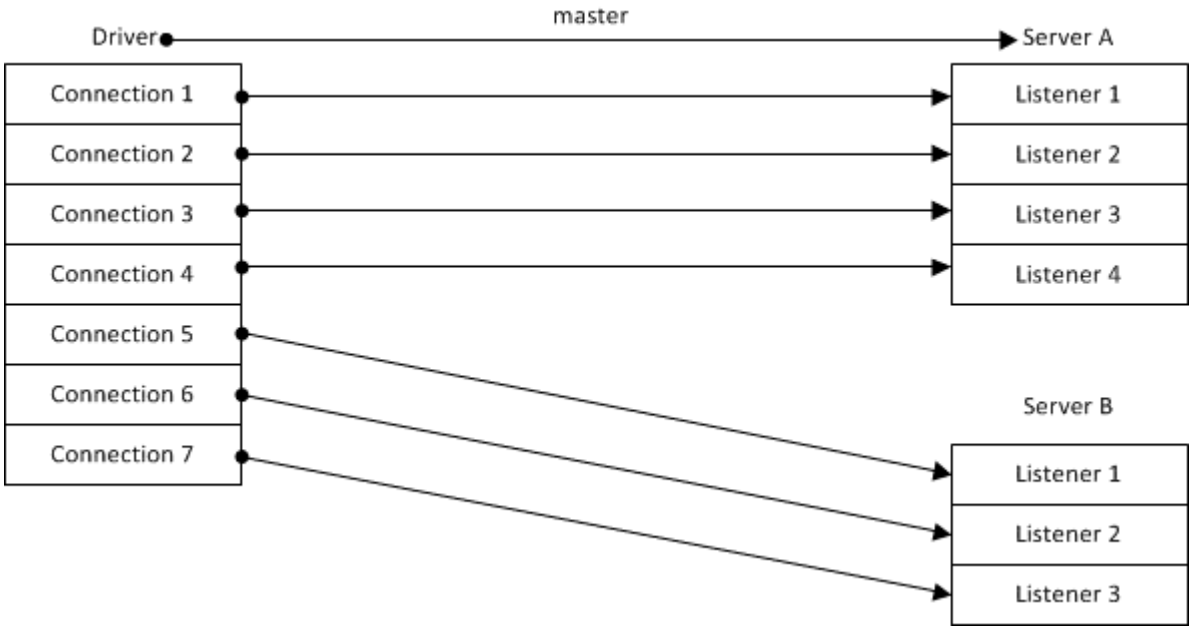
When configuring MapReduce for the server, the mapping step spawns threads that execute a query using one or more server connections. Each thread hosts one connection to one server. The reduce step reads results on each thread in parallel for each connection, and then transparently presents the data to the JDBC call sites.

To configure MapReduce to use a single server connection, set the **MapReduceClient** JDBC property as follows:

```
MapReduceClient = (Hostname, Port, TaskCount)
```

For example: `MapReduceClient= (dvs.example.com, 9999, 7)`

As the following illlustration shows, MapReduce can also be configured to use multiple server connections.



To configure MapReduce to use multiple server connections, set the **MapReduceClient** property as follows:

```
MapReduceClient = (Hostname1, Port, TaskCount1), (Hostname2, Port, TaskCount2),…
```

For example: `MapReduceClient = (dvs1.example.com, 9999, 4), (dvs2.example.com, 1111, 3)`

To further control MapReduce to use a specific range of clients, set the MapReduceClient property as follows:

```
MapReduceClient = (Hostname, Port, maxClientNo, startClientNo, endClientNo)
```

For example, the following specifies MapReduce over two servers for client numbers 1 through 4 and 5 through 7: `MapReduceClient = (host1, 1200, 7, 1, 4), (host2, 1200, 7, 5, 7)`

The **maxClientNo** must be the same for both servers.

Client controlled MapReduce

In some use cases, such as when using Apache Spark, you may want to let the client perform the MapReduce process and still benefit from the server's MapReduce feature. The JDBC driver supports this use case by specifying a single connection as the JDBC MapReduce connection. That single connection is then available for use by a group of specified connections. The JDBC driver only manages the one connection. The client must aggregate the results from each individual connection that is being managed.

To enable client-controlled MapReduce, set the **MapReduceClientCount** and **MapReduceClientNumber** driver properties.

The **MapReduceClientCount** property is used to specify the total number of connections associated with the group of client connections. The **MapReduceClientNumber** property specifies a specific client connection within the group, and has a value between the number 1 and the number specified for the **MapReduceClientCount** property.

The JDBC driver executes queries using the single MapReduce connection for the client connection specified in the **MapReduceClientNumber** property. Only the rows of data for the specified connection are returned, as opposed to using **MapReduceClientCount** over one or more connections to get all rows of data.

To configure client-side MapReduce, set the JDBC driver **MapReduceClientNumber** and **MapReduceClientCount** parameters as follows:

```
MapReduceClientNumber,MapReduceClientCount
```

For example: `MapReduceClientNumber=2;MapReduceClientCount=7;`

In this example, the JDBC driver only creates a single connection, indicated as connection 2 out of 7 available connections. Using a framework like Apache Spark, you must create and manage all remaining connections and aggregate the results from each of those connections.

JDBC driver APIs

The following tables document the methods that are not supported by each of the java interfaces.

The following methods are not supported with the `java.sql.Connection` interface:

Return type	Method name
`Blob`	`createBlob()`
`Clob`	`createClob()`
`NClob`	`createNClob()`
`SQLXML`	`createSQLXML()`
`Struct`	`createStruct (String typeName, Object [] attributes)`
`String`	`getSchema()`
`int`	`getTransactionIsolation()`
`boolean`	`isReadOnly()`
`void`	`releaseSavepoint (Savepoint savepoint)`

Return type	Method name
void	rollback (Savepoint savepoint)
Savepoint	setSavepoint()
Savepoint	setSavepoint (String name)

The following methods are not supported with the `java.sql.PreparedStatement` interface:

Return type	Method name
void	setArray(int parameterIndex, Array x)
void	setAsciiStream(int parameterIndex, InputStream x)
void	setAsciiStream(int parameterIndex, InputStream x, int length)
void	setAsciiStream(int parameterIndex, InputStream x, long length)
void	setBinaryStream(int parameterIndex, InputStream x)
void	setBinaryStream(int parameterIndex, InputStream x, int length)
void	setBinaryStream(int parameterIndex, InputStream x, long length)
void	setBlob(int parameterIndex, Blob x)
void	setBlob(int parameterIndex,InputStream inputStream)
void	setBlob(int parameterIndex,InputStream inputStream, long length)
void	setCharacterStream(int parameterIndex,Reader reader)
void	setCharacterStream(int parameterIndex,Reader reader, int length)
void	setCharacterStream(int parameterIndex,Reader reader, long length)
void	setClob(int parameterIndex, Clob x)
void	setClob(int parameterIndex, Reader reader)
void	setClob(int parameterIndex, Reader reader, long length)
void	setNCharacterStream(int parameterIndex,Reader value)
void	setNCharacterStream(int parameterIndex, Reader value, long length)
void	setNClob(int parameterIndex, NClob value)
void	setNClob(int parameterIndex, Reader reader)

Return type	Method name
void	setNClob(int parameterIndex, Reader reader, long length)
void	setRowId(int parameterIndex, RowId x)
void	setSQLXML(int parameterIndex, SQLXML xmlObject)

The following methods are not supported with the java.sql.Statement interface:

Return type	Method name
ResultSet	getGeneratedKeys()
void	setEscapeProcessing(boolean enable)

The following methods are not supported with the java.sql.ResultSet interface:

Return type	Method name
boolean	absolute(int row)
void	afterLast()
void	beforeFirst()
boolean	first()
Array	getArray(int columnIndex)
Array	getArray(String columnLabel)
NClob	getNClob(int columnIndex)
NClob	getNClob(String columnLabel)
ref	getRef(String columnLabel)
SQLXML	getSQLXML(int columnIndex)
SQLXML	getSQLXML(String columnLabel)
boolean	last()
boolean	previous()
boolean	relative(int rows)
void	updateArray(String columnLabel, Array x)
void	updateAsciiStream(int columnIndex, InputStream x)
void	updateAsciiStream(int columnIndex, InputStream x, int length)
void	updateAsciiStream(int columnIndex, InputStream x, long length)
void	updateAsciiStream(String columnLabel, InputStream x)
void	updateAsciiStream(String columnLabel, InputStream x, int length)
void	updateAsciiStream(String columnLabel, InputStream x, long length)

Return type	Method name
void	updateBigDecimal(int columnIndex, BigDecimal x)
void	updateBigDecimal(String columnLabel, BigDecimal x)
void	updateBinaryStream(int columnIndex, InputStream x)
void	updateBinaryStream(int columnIndex, InputStream x, int length)
void	updateBinaryStream(int columnIndex, InputStream x, long length)
void	updateBinaryStream(String columnLabel, InputStream x)
void	updateBinaryStream(String columnLabel, InputStream x, int length)
void	updateBinaryStream(String columnLabel, InputStream x, long length)
void	updateBlob(int columnIndex, Blob x)
void	updateBlob(int columnIndex, InputStream inputStream)
void	updateBlob(int columnIndex, InputStream inputStream, long length)
void	updateBlob(String columnLabel, Blob x)
void	updateBlob(String columnLabel, InputStream inputStream)
void	updateBlob(String columnLabel, InputStream inputStream, long length)
void	updateBoolean(int columnIndex, boolean x)
void	updateBoolean(String columnLabel, boolean x)
void	updateByte(int columnIndex, byte x)
void	updateByte(String columnLabel, byte x)
void	updateBytes(int columnIndex, byte[] x)
void	updateBytes(String columnLabel, byte[] x)
void	updateCharacterStream(int columnIndex, Reader x)
void	updateCharacterStream(int columnIndex, Reader x, int length)
void	updateCharacterStream(int columnIndex, Reader x, long length)
void	updateCharacterStream(String columnLabel, Reader reader)

Return type	Method name
void	updateCharacterStream(String columnLabel, Reader reader, int length)
void	updateCharacterStream(String columnLabel, Reader reader, long length)
void	updateClob(int columnIndex, Clob x)
void	updateClob(int columnIndex, Reader reader)
void	updateClob(int columnIndex, Reader reader, long length)
void	updateClob(String columnLabel, Clob x)
void	updateClob(String columnLabel, Reader reader)
void	updateClob(String columnLabel, Reader reader, long length)
void	updateDate(int columnIndex, Date x)
void	updateDate(String columnLabel, Date x)
void	updateDouble(int columnIndex, double x)
void	updateDouble(String columnLabel, double x)
void	updateFloat(int columnIndex, float x)
void	updateFloat(String columnLabel, float x)
void	updateLong(int columnIndex, long x)
void	updateLong(String columnLabel, long x)
void	updateNCharacterStream(int columnIndex, Reader x)
void	updateNCharacterStream(int columnIndex, Reader x, long length)
void	updateNCharacterStream(String columnLabel, Reader reader)
void	updateNCharacterStream(String columnLabel, Reader reader, long length)
void	updateNClob(int columnIndex, NClob nClob)
void	updateNClob(int columnIndex, Reader reader)
void	updateNClob(int columnIndex, Reader reader, long length)
void	updateNClob(String columnLabel, NClob nClob)
void	updateNClob(String columnLabel, Reader reader)
void	updateNClob(String columnLabel, Reader reader, long length)
void	updateNString(int columnIndex, String nString)
void	updateNString(String columnLabel, String nString)

Return type	Method name
void	updateNull(int columnIndex)
void	updateNull(String columnLabel)
void	updateObject(int columnIndex, Object x)
void	updateObject(int columnIndex, Object x, int scaleOrLength)
void	updateObject(String columnLabel, Object x)
void	updateObject(String columnLabel, Object x, int scaleOrLength)
void	updateRef(int columnIndex, Ref x)
void	updateRef(String columnLabel, Ref x)
void	updateRowId(int columnIndex, RowId x)
void	updateRowId(String columnLabel, RowId x)
void	updateShort(int columnIndex, short x)updateShort(String columnLabel, short x)
void	updateSQLXML(int columnIndex, SQLXML xmlObject)
void	updateSQLXML(String columnLabel, SQLXML xmlObject)
void	updateString(int columnIndex, String x)
void	updateString(String columnLabel, String x)
void	updateTime(int columnIndex, Time x)
void	updateTime(String columnLabel, Time x)
void	updateTimestamp(int columnIndex, Timestamp x)
void	updateTimestamp(String columnLabel, Timestamp x)

Chapter 3. The ODBC driver V3.1

For non-Java based applications and tools, use the ODBC driver to access data that is made available through Data Virtualization Manager.

The ODBC driver implements the ODBC Direct network protocol that is used to connect to the Data Virtualization Manager server, and uses the ODBC API to execute SQL queries. The driver implements all of the core level 1, and all but one method of the level 2 ODBC functionality.

The driver is available for Windows and Unix/Linux platforms.

Platform	Installation file name
Windows operating system	• DV_ODBC-3.1.24384-win32.exe • DV_ODBC-3.1.24384-win64.exe
Unix • Red Hat Enterprise Linux 5 x86 • Red Hat Enterprise Linux 5 x64 • Red Hat Enterprise Linux 6 x86 • Red Hat Enterprise Linux 6 x64 • Red Hat Enterprise Linux 7	• DV_ODBC-3.1.24501-rhel5-x64.tar • DV_ODBC-3.1.24501-rhel5-x86.tar • DV_ODBC-3.1.24501-rhel6-x64.tar • DV_ODBC-3.1.24501-rhel6-x86.tar • DV_ODBC-3.1.24501-rhel7-x64.tar • DV_ODBC-3.1.24501-slesl0-x64.tar • DV_ODBC-3.1.24501-slesl0-x86.tar • DV_ODBC-3.1.24501-slesl1-x64.tar • DV_ODBC-3.1.24501-slesl1-x86.tar • DV_ODBC-3.1.24501-slesl2-x64.tar • DV_ODBC-3.1.24501-slesl2-x86.tar

The following ODBC drivers are installed:

- IBM Data Virtualization Manager for z/OS driver
- IBM Data Virtualization Manager for z/OS Debug driver

The Debug driver incorporates an advanced tracing component that allows for detailed tracing of ODBC calls. It should be utilized during application development in case problems are encountered and there is a need to contact Customer Support. After applications have been developed and debugged, the code should be modified to use the IBM Data Virtualization Manager for z/OS driver. The Debug driver's performance may be slightly less than the driver.

Connecting an application to a data source using ODBC

Connection details may be specified in a data source name (DSN). On a Windows computer, you configure a DSN based on the [version #] driver with the ODBC Data Source Administrator utility. The **General** tab allows entry of all required connection properties; the **Advanced** tab allows entry of optional connection properties. An identical DSN must be configured on all computers on which applications that are dependent on this DSN run.

Alternatively, an application may utilize a connection string to specify all required and optional connection properties. When you use this approach, a DSN is not required. The connection string consists of a sequence of semicolon separated entries of the form KEYWORD=value. The following table lists the required keywords.

Table 2. Application connection keywords	
KEYWORD	**Value**
DRIVER	Must be: {Data Virtualization Driver 3.1}. This keyword must be provided even if the connection string is being used to add connection properties to a DSN.
UID	User name or identifier
PWD	Password or password phrase for the specified user
PORT	TCP/IP listener port
HOST	Host name or IP address of the server that runs the target data source
SUBSYS	DB2 subsystem if using DB2; otherwise must be set to NONE.
DSN	Data source name. If provided, connection string values override corresponding values in DSN. Set to null (no entry after =) if a DSN is not relevant.

```
obConn.ConnectionString="DRIVER={Data Virtualization
Driver 3.1};
UID=userid;PWD=xxxx;Port=####;HOST=hostname or IPad
dress;
SUBSYS=NONE;DSN=;"
```

Applications executing on the same sysplex as the server are not required to specify logon credentials when invoking the JDBC driver. Instead, the application can specify empty string settings for the user and password to indicate that the current application user ID is to be used. For example, empty values can be specified in the JDBC connection string as follows:

```
…; user=;password=; …
```

When values are specified, the credentials supplied on the driver connection string will be used to override any user ID collected by ioctl when the driver is executing on the same sysplex.

Refer to the list of connection properties for all required and optional KEYWORDS and their interpretation.

Accessing Double Byte Characters

To access double byte characters, do the following:

- Set the parameter **DBCS** mode to **GRAPHIC**.
- Set the parameter **LGID** with appropriate values.
- Set the target variable data type as **SQL_C_WCHAR** in the application.

For example, to display and access double byte characters such as Japanese characters, the target variable data type should be set as **SQL_C_WCHAR** in the application, the **DBCS** mode should be set to **GRAPHIC**, and the **LGID** value should be set to **JPL**, **JNL**, or **JTL**.

If the value of a graphic column is 本社本社本社本社本社 with a length **10** and if it is fetched using the SQLGetData() API with the TargetType set as **SQL_C_WCHAR**, the data is displayed as 本社本社本社本社本社.

If the value is fetched using the SQLGetData() API with the TargetType set as **SQL_C_CHAR** (which is single byte), 20 bytes of data is displayed.

ODBC connection properties

The following table lists the connection parameters that are used to configure the driver.

Table 3. ODBC connection properties

Parameter name	Description
Accessible Tables Alias: AT	This parameter is used to control if table lists returned by the driver should only contain tables that can be accessed by the current user. Select **YES** if you want only those tables that are returned for which you have SELECT authority. The table list includes tables for which the SELECT authority has been granted to PUBLIC or PUBLIC AT ALL LOCATIONS. The table list does not include tables for which you have SELECT authority by virtue of a secondary authorization ID.
ADABAS Correlation Name Support Alias: ABCN	This parameter requests support for ADABAS column correlation names on a connection basis. Use the server parameter ADABASCORRELATIONIDS to control correlation name support for all connections. Setting this parameter to YES may cause errors with older, non-standard SQL that uses syntax that is in conflict with correlation name usage. This parameter is only available if SDEM = YES. Default: NO Valid values: [YES, NO]
Add WITH UR To Queries Alias: WIUR	This parameter controls whether a WITH UR clause should be added to queries or not. The WITH UR clause reduces the amount of CPU time needed for some queries as well as possibly changing the results of these queries by allowing uncommitted data to be read. If this parameter in set to YES, then a WITH UR clause is added to queries. If this parameter is set to NO, then a WITH UR clause is not be added to queries. The default is NO.
Allow DataDirect Licensing Calls	This parameter controls whether the driver should report an error or not, when SQLSetConnectOption is called for attributes SQL_LIC_FILE_NAME and SQL_LIC_FILE_PASSWORD. These attributes are specific to Datadirect OEM client-side licensing and when ADLI is set to NO (default value), SQLSetConnectOption reports an error that “Driver is not capable of supporting this option.” But when ADLI (Allow DataDirect Licensing calls) is set to YES, SQLSetConnectOption bypasses the OEM licensing calls and returns success.
Allow Repeating SQLGetData Alias: RPGT	This parameter when set allows repeating SQLGetData calls for the same unbound column while the cursor is still on that result set row. The exceptions are LOB type columns. They cannot be retrieved repeatedly. The default is NO.
AlternateID Alias: ALUS	This parameter is used to set the host secondary userid for client applications. This parameter must be eight or fewer characters long. If this parameter is set to a non-blank, non-null value a SET CURRENT SQLID statement is issued after DSNALI OPEN processing is completed.
Always Convert Dynamic SQL Alias: ALCD	This parameter controls what happens if dynamic SQL cannot be converted to static SQL. If this parameter is set to YES, an error is reported to the application if dynamic SQL cannot be converted to static. If this parameter is set to NO, the dynamic SQL is sent to the host for processing. This parameter is not even tested unless the primary Dynamic-To-Static SQL parameter is set to YES. This parameter must be set to NO if dynamic SQL, and static SQL are going to be used together.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Application Name Alias: APNA	The Application Name is sent to the host as part of the login information. The Application Name is normally used to group SQL statements within a plan. If the Application Name is not set, then all of the SQL associated with a plan is considered to be part of one large group. If the SQL used with one plan must be divided into subgroups (for conversion to static SQL), then Application Name must be set.
Async Execution Timeout Value Alias: ASTM	This parameter controls the default wait time for an ODBC function that is running asynchronously. This parameter is measured in seconds, and must be an integer. The default value is 2 seconds.
Authentication Method Alias: SECU	This parameter allows the user to choose whether to encrypt logon credential using DEFAULT encryption mechanism or AES encryption mechanism. **DEFAULT** The password is encrypted by using the Data Virtualization Manager encryption mechanism when the logon request is sent to the host. **AES** The password is encrypted by using Diffie-Hellman key exchange (by using AES encryption) when the logon request is sent to the host. When DOMAIN-BASED authentication is selected, the driver will verify that the user ID associated with the current process has been authenticated by a domain-based system. For the Windows platforms, this requires that the user first logs on to a NT domain. For the UNIX platforms, the local machine must be a member of a NIS domain, and the password database used to authenticate the user must be NIS-mapped. Default: DEFAULT Valid values: [DEFAULT, AES]
Binary Passthrough Alias: BIPA	This parameter controls if host binary data should be returned to client applications without being converted to hexadecimal. If this parameter is set to YES, host binary data is passed through to client applications unchanged. If this parameter is set to NO, then host binary data is converted to hexadecimal (in accordance with the ODBC specification). The default is NO.
Bind DOUBLE as DECFLOAT Alias: DADE	This parameter controls how the driver should bind a DOUBLE input value for the target decimal column. When it is set, the driver binds a DOUBLE input value as a DECFLOAT value. Default is YES, which means that the value of target decimal column is rounded, not truncated.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Buffer Format Alias: BUFO	This parameter controls what format of optimization of the buffer is used to transmit and receive data to the server. Values include: • CMBU – If this parameter is set to CMBU, the communication buffers are sent to the server and received from the server in a compressed format. • CMBV – If this parameter is set to CMBV, the communication buffers are sent to the server and received from the server truncated row data. • CMBZ – If this parameter is set to CMBZ, the communication buffers are sent to the server and received from the server using ZLIB library for compression and decompression data. • UNCOMPRESSED – If this field is set to UNCOMPRESSED, the communication buffers are sent to and received from the server in an uncompressed format. Default: UNCOMPRESSED
Bypass Double Quotes Alias: BYDB	This parameter is set to YES if double quotation marks should be left alone. This parameter is provided to fix certain application bugs.
Bypass Optional Output Parameter Alias: BYOP	This parameter controls how the driver should handle the optional output parameter in the escape CALL syntax {?=CALL procname}. Currently, host procedure does not return any value for the optional output parameter, so this parameter should always be set to YES, which is the default.
CALL Lock Value Alias: CALK	This parameter controls the type of lock that is associated with CALL statements on the host. If this parameter is set to NONE, then the host code assumes that CALL statements do not obtain any host database locks. Other possible values are SHARE, UPDATE, and EXCLUSIVE. The default is EXCLUSIVE.
Catalog Prefix Alias: CPFX	This parameter indicates that the table owner (has authorization ID) of the eight DB2 tables that are optimized to support ODBC catalog queries.
Change Char Data Type length Alias: CHLN	This parameter controls if SQLTypeInfo calls should show that variable data is 255 bytes long or not. DB2 variable data is only 254 bytes long. However, some applications do not support this. If this variable is set to YES, then SQLTypeInfo will return 255 as the length of variable data. If this variable is set to NO then SQLTypeInfo will return 254. Note that if this keyword is set to YES, then the behavior of SQLCancel is modified to circumvent vendor implementation errors. The default is NO.
Change Dynamic SQL to Auto-Static Alias: AUST	This parameter controls the conversion of Dynamic SQL to Auto-Static SQL. This functionality is only available when connected to DB2. Dynamic SQL, and it can only be converted to Auto-Static SQL if the host supports it and this parameter is set to YES. If this parameter is set to YES the driver attempts to convert all Dynamic SQL to Auto-Static SQL. If this parameter is set to NO normal Dynamic SQL processing is performed. The default is YES.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Column Order Option Alias: CNMD	"This parameter shows how column names should be returned by the SQLColumns function that don't specify a column name order. If the order of column names aren't specified, this parameter determines the order that the SQLColumns function returns. this parameter controls the connection mode that is used by ODBC applications. The connection mode determines how long each physical connection (session or conversation) lasts and if SQL operations are processed in blocks. The default is to use a permanent connection and to send each SQL operation standalone to the server. In BLOCK mode, the session is permanent. However, SQL operations are blocked and sent together. In TRANSACTION mode each SQL operation is sent standalone but the session is terminated at the end of each Logical Unit Of Work (LUOW). In TRANSBLOCK mode the session is terminated at the end of each LUOW and SQL operations are blocked and sent together. In MESSAGE mode SQL operations are blocked and sent together using messages. In MESSAGE mode, no session is ever maintained.
Connection Mode Alias: CNMD	This parameter controls the connection mode used by ODBC applications. The connection mode determines how long each physical connection (session or conversation) lasts and if SQL operations are processed in blocked. The default is to use a permanent connection and to send each SQL operation standalone to the server. In BLOCK mode, the session is permanent. However, SQL operations are blocked and sent together. In TRANSACTION mode each SQL operation is sent standalone but the session is terminated at the end of each Logical Unit Of Work (LUOW). In TRANSBLOCK mode the session is terminated at the end of each LUOW and SQL operations are blocked and sent together. In MESSAGE mode SQL operations are blocked and sent together using messages. In MESSAGE mode, no session is ever maintained.
Connection Name Alias: CNNA	This parameter is used to specify the connection name. The use of the connection name is application-specific. The connection name can be up to 8 bytes long. However, the application may or may not use all of the bytes. The connection name is padded on the right with blanks.
Connection Timeout Value Alias: CNTM	This parameter controls how long the ODBC client waits for a connection to the host server to complete. If this parameter is set to zero, then the system default value is used. Otherwise, the specified value is used. This value should never be negative, but can be zero. The use of this parameter may cause unpredictable results in many environments. This value is measured in seconds. The default is zero.
Convert DB2 Date to CHAR Alias: DTCH	When this parameter is set, the DB2 date field is converted to SQL_CHAR. The default is NO.
Convert DB2 Time to CHAR Alias: TSCH	When this parameter is set, the DB2 timestamp and time fields are converted to SQL_CHAR. The default is NO.
Convert Dynamic SQL to Static SQL Alias: CD	This parameter is used to control converting dynamic SQL to static SQL. Dynamic SQL can only be converted to static SQL if this parameter is set to YES, and if the .pln file can be located and contains all of the SQL conversion information.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Convert Nulls to Blanks Alias: CVNL	This parameter controls if nulls (zero bytes) in character string data that is returned from the server to the client should be converted to blanks or not. Both fixed length and variable length character data is affected by this parameter. If this parameter is set to YES, then nulls are converted to blanks. If this parameter is set to NO, then nulls are not converted. The default is NO.
Convert Strings to Params Alias: LGPA	This parameter controls if long strings should be converted to LONG VARCHAR types. This conversion is needed because some applications produce literals that are longer than can be handled by the host database. If this parameter is set to NO, then each SQL string is not scanned for long strings. If this parameter is set to YES, then all long strings are converted.
Convert Timestamps Alias: CVTS	This parameter controls if timestamp values should be converted to other types. If this parameter is set to YES, then timestamps that appear to be dates are converted to dates and timestamps that appear to be times are converted to times. These conversions are required to circumvent bugs in several products including MS Access and Crystal Reports. If this parameter is set to NO, then timestamps are not modified. The default is NO.
Convert LONG VAR Data to LOB Alias: LBCB	This parameter when set allows any SQL_LONGVARCHAR or SQL_LONGVARBINARY data to be treated as LOB (either CLOB or BLOB). Default is YES.
COUNT() Fix Type Alias: COFX	This parameter controls how COUNT(column name) SQL functions are fixed. Application tools use the COUNT(column name) function two different ways, both of which are wrong. In some cases, COUNT (column name) means COUNT(DISTINCT column name). In other cases it means COUNT(*). If this parameter is set to DISTINCT, then the DISTINCT keyword will be inserted. If this parameter is set to ASTERISK, then the column name will be replaced with a '*'. If this parameter is set to NONE, then no changes will be made. The default is DISTINCT.
Create Table Index Automatically Alias: CRIN	Tables with a primary key or unique constraint are automatically created with an index to enforce the uniqueness. To disable this option, set this parameter to NO.
Current Degree Alias: SEDG	This parameter is used to set the initial Current Degree value on the host. The only possible values for this parameter are 'ANY' or '1'. No other values are supported currently. If this parameter is set to a non-blank, non-null value a SET CURRENT DEGREE statement is issued after DSNALI OPEN processing is completed.
Current Package Set Alias: SEPK	This parameter is used to set the initial Current Package Set value on the host. This value must be eighteen or fewer or characters long. If this parameter is set to a non-blank, non-null value a SET CURRENT PACKAGESET statement is issued after DSNALI OPEN processing is completed. The SET statement assigns the specified value to the CURRENT PACKAGESET special register.
Current Rules Alias: SERL	This parameter is used to set the initial Current Rules value on the host. The only possible values for this parameter are 'DB2' or 'STD'. No other values are supported now. If this parameter is set to a non-blank, non-null value a SET CURRENT RULES statement is issued after DSNALI OPEN processing is completed.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Current Schema Alias: SESC	This parameter is used to set the initial Current Schema value on the host. If this parameter is set to a non-blank, non-null value a SET CURRENT SCHEMA statement is issued after DSNALI OPEN processing is completed.
Cursor Commit/ Rollback Behavior Alias: CRBH	This parameter controls what value to return for the cursor commit and rollback behavior SQLGetInfo request. DELETE closes cursors and delete prepared statements. To use the cursor again, the application must reprepare and re-execute the statement. CLOSE only closes cursors. For prepared statements, the application can call SQLExecute on the statement without calling SQLPrepare again. PRESERVE will preserve the cursor position and the prepared statement. The default is CLOSE.
Customer Secret Key Alias: SKEY	The secret key value for the user.
Data Source Name Alias: DSN	The developer specified name for this data source.
Data Truncation	This parameter controls whether to truncate the extra characters, display a warning or give a client error message or send a message to the server and return the server error message. If Warning, then it truncates and returns a warning. If Error, a client error message is returned. If Ignore, the query is sent to the server and a server error message is returned. The default is Warning.
Date Format Alias: DTFM	This parameter is used to specify how ODBC dates are converted to character strings. If this field is set to ODBC, then the standard ODBC/ISO format yyyy-mm-dd is used. If this field is set to UK, then the dd-mm-yyyy format is used. If this field is set to EUR, then the dd.mm.yyyy format is used. If this field is set to USA, then the mm/dd/yyyy format is used. If this field is set to USA2, then the mm/dd/yy format will be used. The default is ODBC.
DB2 Default DB Alias: PARM	This is the default DB2 database that is used when issuing a CREATE TABLE. All CREATE TABLE queries are suffixed with “IN DATABASE” and the specified value.
DB2 Plan Name Alias: PLAN	This is the DB2 plan name that is used to create the thread to DB2 at connect time. Default setting is blank.
DB2 Version String Alias: D2VR	This parameter overwrites the host returned DB2 version string data. This string should be in the following format: x.y.z --- where x, y and z are all numeric digits. x is the DB2 version byte. y is the DB2 modification level. z is the DB2 release level. Both the version byte and the modification level must be set. If the release level is not set, then 0 is assumed. For example, 2.3 for DB2 version 2 mod level 3; 4.1.1 for DB2 version 4 mod level 1 and rel 1.
DBCS Mode Alias: DBMD	This parameter is used to specify what type of data will be converted to DBCS data that stored in the column types of GRAPHIC, VARGRAPHIC and LONGVARGRAPHIC. If CHAR is specified, then the DBCS data will be treated as character data. If BINARY is specified, then the DBCS data will be treated as binary data. If GRAPHIC is specified, then the DBCS data will be treated as graphic data. The default is CHAR.

Table 3. ODBC connection properties (continued)

Parameter name	Description
DBCS Remove Blanks Alias: DBQO	This parameter controls if blanks are removed from within quoted strings for double byte languages (such as Chinese, Japanese, and Korean). If this parameter is set to YES, then blanks within quoted strings in SQL being sent to the host is removed. If this parameter is NO, then blanks inside quoted strings in SQL being sent to the host is not changed. The default is YES.
DBMS Type Alias: DBTY	The type of database to be accessed. Select from DRDAorDB2 or DVS.
Decimal as Numeric Alias: DENU	The driver treats DECIMAL and NUMERIC columns the same. This parameter determines whether these columns should be reported as DECIMAL columns or NUMERIC columns. When set to YES, all DECIMAL and NUMERIC columns are reported as NUMERIC. When set to NO, all DECIMAL and NUMERIC columns are reported as DECIMAL. The default is NO.
Decimal Point is Comma Alias: DCCM	This parameter controls if the decimal point is a comma or a period. If this parameter is set to YES, then comma is considered to be the decimal point. If this parameter is set to NO, then period is considered to be the decimal point. This parameter should only be set to YES, if the host uses comma as the decimal point. This parameter causes commas to be converted to periods before data is passed to the Auto-Static SQL conversion facility. The default is NO.
Default Column Names Alias: DFCL	This parameter controls whether the driver assigns default column names (of the form COLnnn, where *nnn* is the column number) when the DBMS does not return names for the columns. The default is NO.
Default Database Alias: DBD	This parameter is added to CREATE TABLE statements that do not specify a database in which to create the table. If the statement includes the 'IN DATABASE' clause, the driver will append this clause using the default database value. This parameter is required for almost all users, since most users do not have authority to create tables in DB2's default database.
Default Schema Alias: DFSC	This parameter controls the default schema for stored procedures with implicit schema name. The default, or value 0, means the procedures are run as IBM Data Virtualization Manager for z/OS RPC. If the value is set to 1, the server uses RPCDEFAULTSCHEMA value as the schema name. If the value is a specific schema name, the schema name is used as the default schema.
Defer Prepare for CALLs Alias: DFPR	This parameter controls if prepare of CALL statements is deferred or not. If this field is set to YES, then prepare is always deferred. If this flag is set to NO, then prepare is only deferred for CALLs that have one or more parameter markers. That is, prepare is always deferred for CALLs that have parameter markers. Prepare is deferred for CALLs that do not have parameter markers, only if this parameter is set to YES. The default is NO.
Description	Optional description of the data source.
Disable Async Option Alias: NOAS	This parameter is used to indicate whether, or not asynchronous execution is allowed. Setting this parameter to YES prevents all asynchronous processing.
Disable All Prompts Alias: NOPM	This parameter when set to YES disables all interactive prompts or informational message boxes. This feature is required for when the driver is being called from an NT service, an UNIX daemon process, or any server type applications, which cannot be interrupted.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Disable List of Catalogs Alias : DILC	This parameter disables obtaining a list of catalogs by the SQLTables ODBC function. Setting this option to YES forces the SQLTables to return RC = -1 and SQLSTATE = 'S1C00' (for ODBC2.0) or SQLSTATE = 'HYC00' (for ODBC3.0). It allows to circumvent a conflict between using special semantics of SQLTables call for obtaining a list of catalogs and processing SQLTables calls by MS Excel to get a list of tables. The default value is YES which means that obtaining a list of catalogs is disabled."
Disable Prepare/Open Alias: DIPO	This parameter sets the prepare/open optimization. Setting this parameter to NO saves a network round trip for SQLPrepare and SQLExecute by ignoring the SQLPrepare until SQLExecute is called or an ODBC function requesting meta-data is called. YES disables this optimization by sending the SQLPrepare and SQLExecute in separate network calls. This parameter affects DB2 SQL only. For non-DB2 SQL, a prepare is always sent to the host at SQLPrepare, and an execute is always sent to the host at SQLExecute. For applications that access non-DB2 data source and do NOT require meta-data at prepare time, it is recommended to set WRPR to NO for better performance since this saves a network roundtrip at prepare time.
Disconnect Timeout Value Alias: DCTM	This parameter controls how many seconds the driver waits for a socket to be ready for disconnect operations. The default value is 30 seconds. Setting the field to zero disables the timeout. This value should never be negative.
Display Leading Zero	This parameter controls whether to display the leading 0 before the decimal point when the user requests a String representation of a Decimal column value with precision is equal to scale. The default is YES, which means the leading zero is displayed.
DTS Plan File Alias: DSFL	This parameter is used to specify the name of the local plan file that is used to convert dynamic SQL to static SQL. The plan file name should end with .pln and can be a full path name, if need be. This parameter should only be used if the local plan file does not have the default name and/or is not located in the working directory of the application. The default name of the local plan file is always planname.pln, where planname is the name of the DB2 plan that is used by the application.
Enable SQLPrepare for ADABAS/VSAM/IMS Alias: WRPR	This flag controls the behavior of SQLPrepare for non-DB2 data sources such as ADABAS, VSAM, IMSDB, and VSAM CICS. When this keyword is set to YES (which is the default), a request is always sent to the host at SQLPrepare time to obtain metadata for the SQL statement. For applications that access non-DB2 data sources and do NOT require metadata after the SQLPrepare, it is recommended to set WRPR to NO for better performance since this will eliminate a network roundtrip whenever a SQLPrepare is executed. This parameter is only available if SDEM = YES. Default: YES Valid values: [YES, NO]
Enable SQL_QUERY_TIMEOUT Support Alias: ESQT	This parameter controls whether the driver supports the option SQL_QUERY_TIMEOUT in the SQLSetStmtOption API or not. The default is NO, which means the driver ignores the SQL_QUERY_TIMEOUT value set by the application. If this parameter is set to YES, the driver honors the setting. If Operation Timeout (OPTM) is also set, the value set in OPTM takes precedence.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Encryption Method Alias: EM	Specifies the method that the driver uses to encrypt data that is exchanged between the driver and the database server. The value NONE means the data that is exchanged is not encrypted. The value TLS1 means the data that is exchanged is encrypted by using the TLS1 version, TLS1.1 (TLS version 1.1), TLS1.2 (TLS version 1.2) of the SSL protocol. The value SSL3 means the data that is exchanged is encrypted by using the SSL3 version of the SSL protocol. The default is NONE.
Extended Cursor Pool Alias: EXCU	This parameter controls if the Extended Cursor Pool should be used or not. If this parameter is set to YES, then the Extended Cursor Pool is used. The Extended Cursor Pool is much larger than the standard cursor pool. The Extended Cursor Pool cannot be used with Auto-Static SQL. If this parameter is set to NO, then the normal cursor pool is used. The default is NO.
Fix Floating Point Rounding Alias: FXFL	Floating point errors are typically seen when a floating point number is converted to a decimal or integer value. If this parameter is set to YES, then floating point values are adjusted before they are converted. If this parameter is set to NO, then no adjustments are made. The default is NO.
Fix INSERT Statements Alias: FXIS	This parameter controls if the VALUES clause of each INSERT statement should be scanned for dates, times, and timestamps without quotes and quotes should be added. If this parameter is set to YES, quotes are added to dates, times, and timestamps as need be. If this parameter is set to NO, then INSERT statements are not modified. The default is NO.
Fix String Length Alias: STFX	This parameter fixes the bug where some products incorrectly set string length to zero when SQL_CHAR field only contains blanks. The default is NO.
Floating to Character Digits Alias: MXDG	This parameter controls the maximum number of digits that should be generated for a floating point to character conversion. This parameter should only be specified if the default value is not acceptable. The default value is 6.
German NLS Support Alias: NLGR	This parameter is set to YES to resolve certain problems handling SQL in the German language environment. This parameter should not be set for any other reason.
GRAPHIC Data Processing Alias: GHDS	This parameter affects the string literals only for the GRAPHIC columns, if keyword is set to YES then all string literals not proceeded by 'G' or 'g' are converted to parameter markers. This parameter is active if the 'Change Dynamic SQL to Auto-Static' (AUST) = 'No' and the language ID (LGID) is a DBCS types.
GRAPHIC Literal Processing Alias: GXSR	This parameter controls if string literals should be converted to parameter markers. This conversion is needed for some applications operating in DBCS modes. If this parameter is set to YES then all string literals not proceeded by 'G' or 'g' will be converted to parameter markers.
Host Name Alias: HOST	The name or IP address of the computer that runs the database to be accessed.
Host User Parm Alias: USERPARM	This parameter is sent to the host as part of the logon information. The host uses this forwarded value to complete the logon to the host security system and/or host databases.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Identifier Quote Option Alias: IDQO	This parameter specifies what identifier quotation mark character should be returned to the application by using SQLGetInfo with SQL_IDENTIFER_QUOTE_CHAR. This parameter is needed to fix certain application bugs. The default is DOUBLE quote characters.
Ignore High Bound Column Errors Alias: IGHI	This parameter controls if SQLFetch should ignore errors for higher-numbered bound columns. If this parameter is set to YES, then SQLFetch ignores column binding errors. If this parameter is set to NO, then column binding errors are handled normally. This parameter must be set to YES to enable ODBC tools (such as Microsoft Excel) to work. The default is NO.
Ignore Underscore Characters Alias: IGUN	This parameters controls if the underscore character should be considered to be a wild card or just a regular character. If this parameter is set to YES, then underscore ('_') are handled as a normal byte. If this parameter is set to NO, then underscore is treated as a wildcard character that matches other single byte. This parameter is used to solve problems where a table or procedure name actually has an underscore in the name and the underscore is misinterpreted as a wildcard. The default is NO.
Keystore Alias: KS	Use this parameter to specify the file system location of the keystore file. The keystore file contains a list of the valid client certificates that are trusted by the server for optional Client Authentication with SSL. **Note:** The keystore and truststore files may be the same file.
Keystore Password Alias: KSP	The password that is required to access the keystore. **Note:** The keystore and truststore files may be the same file and, therefore, have the same password.
Keep Literal Quotes Alias: KEQU	This parameter is used to retain quotation marks around and/or embedded in string literals. If this parameter is set to YES, then quotations around and/or embedded in string literals are retained. This only applies to string literals passed to Stored Procedures, including MDI Stored Procedures. If this parameter is set to NO, then quotation marks are removed from string literals. The default is NO.
Language ID Alias: LGID	This parameter is used to specify the language to be used. The possible values are Arabic (ARB), Simplified Chinese (CHS), Traditional Chinese (CHT), Danish (DAN), Default (DFT), German (DEU), U.S. English Compatibility (ENC), U.K. English (ENG), U.S. English (ENU), Modern Spanish (ESN), Castilian Spanish (ESP), Finish (FIN), French (FRA), Canadian French (FRC), Icelandic (ISL), Italian (ITA), Japanese (JPL), Japanese Latin Extended (JPX), Japanese Katakana-Kanji JIS X213 (JNL), Japanese Latin-Kanji JIS X213 (JNX), Japanese Katakana-Kanji Extended (JTL), Japanese Latin-Kanji Extended (JTX), Korean using US English for SBCS and KSC 5601 for DBCS (KOR), Micro Decisionware (MDI), Dutch (NLD), Norwegian (NOR), PeopleSoft English (PPS), Portuguese (PTG), Swedish (SVE), Turkish (TUR), Korean using Code Page 833 for SBCS and KSC 5601 for DBCS (KRN), and Norwegian with Latin1 for Unix (NGN). The default is ENU.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Long Data Fix Alias: LGFX	This parameters controls if a large number should be returned for the length and precision of LONG VARCHAR fields, rather than the actual length and precision. This fix is needed to resolve certain problems in ODBCDirect GetChunk processing. If this parameter is set to YES, then a large number is returned for the length and precision of LONG VARCHAR fields. If this parameter is set to NO, then the actual values are returned for the length and precision of LONG VARCHAR fields. The default is NO.
Maximum Buffer Size Alias: MXBU	This parameter sets the client-side maximum communication buffer size for all data exchanges. The default sets the maximum buffer size to 256k. Note that the final buffer size used by the driver is a negotiated value based on the server-side setting. The actual run time buffer size could be smaller than the value specified in this parameter, but not larger. The default is 256k.
Maximum Rows Alias: MR	This parameter setting limits the number of rows that are returned from a single query. If no value or zero is specified, no restriction is placed on the number of rows returned.
MDI Text/Keywords Alias: MDBO	This parameter controls if variable text and keywords can be used together with MDI RSPs. If this parameter is set to YES, then variable text can be used with MDI keywords. If this parameter is set to NO, then variable text cannot be used with keywords. The default is NO.
MDI Keep Quotes Alias: MDQO	This parameter controls if quotation marks should be retained around MDI keyword values. If this parameter is set to YES, then quotations are included in keyword values and keyword value lengths. If this parameter is set to NO, then quotations are stripped from MDI keyword values. The default is NO.
MDI Delimited Args Syntax Alias: MDSY	This parameter controls whether argument lists delimited by TSQL's special delimiters are supported. The default is NO.
MDSY Quoting Character Alias: MDQC	This parameter sets a configurable quotation character for use in MDI calls.
MDI DATA Padding Limit Alias: MDDP	This parameter allows the user to configure the total combined width of parameters for MDI calls when used in conjunction with MDSY. The default is 0 (no padding).
No Nulls Alias: NONL	This parameter controls if zero length strings should be returned or not. If this parameter is set to YES, then zero length strings are replaced by one blank. If this parameter is set to NO, then zero length strings are not modified. The default is NO.
Number of Active Statements Alias: ACMT	This parameter controls the maximum number of active statements for a connection. The default value 0 means there can be up to 50 active statements per connection.
ODBC Client Code Page Alias: ODPG	This parameter is used to specify the ODBC client code page. The default is a set of UNIX code pages for UNIX and Windows Latin 1 (ANSI) for all supported Windows operating systems. Windows Latin 1 is also known as ISO 8859. Specifying LATIN1 forces the use of the Windows Latin 1 code page in any environment. Specifying UNIX forces the use of the UNIX code pages in any environment.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Operation Timeout Value Alias: OPTM	This parameter controls the timeout value for all client network operations (that is, query preparation, execution, retrieving result set) AFTER the connection is established. This value should never be negative, but can be zero. The use of this parameter may cause unpredictable results in many environments. This value is measured in seconds. The default is zero.
Optimal Row Count Alias: OPRW	This parameter limits the number of rows that are returned from the host each time a request for rows is made. Since any number of requests for rows can be made for one query, this value has no effect on the total number of rows that are returned by a query. This value can be used to control the size of the buffers that are used to return rows from the host.
Optimized Fetch Alias: RO	This parameter is used to control if Block-Fetch should be used for the current connection. Specify YES for the block fetch capabilities of the Data Virtualization Manager server to be used with your queries. Using optimized fetch greatly speeds up your queries. The only restriction is that cursor-based deletes [DELETE WHERE CURSOR OF] cannot be used when this mechanism is enabled.
Password Alias: PWD	The password for the user ID.
Plan Name Alias: PLAN	The driver uses plan names on the host to establish a connection to DB2.
Port Number Alias: PORT	The TCP/IP port on which to communicate with the database.
Precision Adjustment Alias: PRAD	When this parameter is set, the precision of literal numeric values that are passed in an RPC call is adjusted to an odd value. For example, the value 12.34 is sent with a precision of 5. The default is YES. NO prevents this adjustment.
Procedure Name Filter Alias: PRNF	This parameter is used to filter the procedure names that are returned by the product. This parameter is only used if it is set to a non-blank value and if the host application does not provide a procedure name filter string. The procedure name filter string from either the application or from this parameter restricts the information that is returned by procedure catalog inquiries (SQLProcedures and SQLProcedureColumns). Only rows that match the procedure name pattern that is provided is returned. If this parameter is set to a single percent sign, then all procedures are returned. There is no default value for this field.
Procedure Owner Filter Alias: PROF	This parameter is used to filter the procedure owners that are returned by the product. This parameter is only used if it is set to a non-blank value and if the host application does not provide a procedure owner filter string. The procedure owner filter string from either the application or from this parameter restricts the information that is returned by procedure catalog inquiries (SQLProcedures and SQLProcedureColumns). Only rows with owner ID's that match the procedure owner pattern that is provided are returned. If this parameter is set to a single percent sign, then all procedures are returned. This parameter defaults to the current user ID or alternate user ID.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Procedure Owner Handling Alias: PROW	This parameter is used to control how procedure owner values are returned to ODBC applications. If this parameter is set to NULL, the actual owner value is used as a prefix to the procedure name and the owner field is returned as a null value. If this parameter is set to EMPTY, the actual owner value is used as a prefix to the procedure name and the owner field is returned as an empty string. If this parameter is set to NONE, then the procedure owner value is not changed. The default is NONE.
Process Escapes Alias: PRES	This parameter is used to turn off escape clause processing. PRES=NO sets escape clause processing off and may result in improved performance by reducing CPU overhead. The default is PRES=YES.
Qualifier Name Separator Alias: QUNA	Set this parameter to YES if SQLGetInfo should return a period when it is called to obtain the SQL_QUALIFIER_NAME_SEPARATOR. This parameter is provided to fix certain application bugs. The default is NO.
Read Only Alias: RDON	Set this parameter to YES to make the data source read-only. Setting the data source to read-only does not prevent update operations. However, setting this parameter to read-only prevents index information from being returned to the application. This will generally prevent any updates from being attempted. Note that setting this parameter to YES will greatly improve the performance of some applications using the standard DB2 catalogs. The default is NO.
Remove blanks Alias: RMBL	This parameter handles data that is received from the server. If the parameter is set to YES, single-byte trailing blanks (X'40') are removed from the data strings (in EBCDIC [Host Code]). This does not affect wide blanks (X'0E40400F'). The default is NO.
Remove Equals Alias: RMEQ	This parameter controls if the = character should be removed from MDI keyword names as part of MDI RSP (Remote Stored Procedure) invocation. If this parameter is set to YES, then the = character is removed from each keyword name. If this parameter is set to NO, then the = character is not removed from the keyword name. This flag only applies to MDI RSPs invoked by using TSQL 0/1/2 syntax. It does not apply to MDI RSPs invoked by using the extended ODBC CALL syntax. The default is NO.
Return Code if No Rows Affected Alias: NODA	This parameter controls what the return code should be set to when an INSERT/UPDATE/DELETE does not affect any rows. If this parameter is set to NODATA, the return code from an SQL operation is SQL_NO_DATA_FOUND. If this parameter is set to INFO, the return code is set to SQL_SUCESS_WITH_INFO (in accordance with the ODBC specification). This parameter can also be set to SUCCESS or ERROR. The return code in these cases is SQL_SUCCESS or SQL_ERROR. The default is INFO.
Retain Cursor Alias: HD	This parameter is used to determine if queries should be run with a cursor that maintains cursor positioning across commit operations. This parameter must be set to YES to allow the auto-commit at close cursor time feature to be used. The default is YES.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Return Global Tables Alias: RTGL	This parameter controls if global temporary tables are returned as normal tables. If this parameter is set to YES, then global temporary tables are treated as normal tables and be described as normal tables. If this parameter is set to NO, then global temporary tables are handled as system tables. This parameter is provided to allow standard ODBC tools to be used with global temporary tables. Many of these tools bypass global temporary tables that are described as global temporary tables. However, the same tools work correctly with global temporary tables if they are described as normal tables. The default is YES.
Return Logon Messages Alias: LGMG	The driver should return logon messages to the application by using the ODBC standard SQL_SUCCESS_WITH_INFO mechanism by default. Most ODBC applications are coded to handle SQL_SUCCESS_WITH_INFO return values and displaying a dialog can cause more problems than it helps for most applications and thus this option should be hidden. In the rare cases that an existing application cannot handle the SQL_SUCCESS_WITH_INFO return value they can set this option to NO. The default is YES.
Return System Tables Alias: RTSY	This parameter controls if system tables (SYSIBM) are returned as normal tables. If this parameter is set to YES, then system tables are treated as normal tables and be described as normal tables. If this parameter is set to NO, then system tables are handled as system tables. This parameter is provided to allow standard ODBC tools to be used with system tables. Many of these tools bypass system tables that are described as system tables. However, the same tools work correctly with system tables if they are described as normal tables. The default is YES.
SBCS Mode Alias: SBMD	This parameter is used to specify how SBCS data is handled. SBCS strings are assumed to contain a mixture of single-byte characters and double byte characters. Each set of DBCS characters starts with a SO and end with a SI byte. If BLANK is specified, all SO/SI bytes are converted to blanks. If DELETE is specified, all SO/SI bytes are deleted from each string. If BLANK is specified and RMBL=YES, then the SHIFT-OUT and SHIFT-IN characters are converted to half-width blanks first, and then the RMBL=YES is applied; deleting trailing half-width blanks. DEFAULT is not supported currently.
Shadow Driver Emulation Mode Alias: SDEM	This parameter controls whether the Data Virtualization Manager ODBC driver should work in the mode of compatibility with the Shadow server v7.x. If this flag is set, then Data Virtualization Manager ODBC driver emulates behavior of the Progress Shadow ODBC driver v7.x to be able to communicate with the Shadow server v7.x. The default value is NO.
SQLExecute Connection Check Alias: EXCC	This parameter is used to force a connection check during SQLExecute processing. When this parameter is set to YES, SQLExecute always return an error condition if the connection has been reset or shut down by the server. If this parameter is set to NO and prepare/open optimization is in effect, SQLExecute will bypass the connection check and return SQL_SUCCESS. The default is NO.
Subsystem Alias: SUBSYS	This keyword specifies the DB2 subsystem. This keyword should only be set if using DB2; otherwise this keyword must be set to NONE.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Suppress Decimal Trailing Zeros Alias: STZO	This parameter controls if trailing zeros should be removed from decimal fields. The default is NO.
System Engineering Value Alias: SEVL	This parameter is set to various values to obtain diagnostic and debugging data. This parameter should only be used at the specific request of the Technical Support staff. The default is zero.
Table & Column Name Modification Alias: TCLM	This parameter controls adjustment of table and column names for DB2. NONE implies that the names are considered correct and are not processed. The default is NONE.
Table Filter Alias: DP	This parameter is used to filter the table lists returned by the product. This parameter restricts the information returned by catalog inquiries (SQLTables, SQLColumns, SQLTablePrivileges). Only rows that match the table owner pattern provided are returned. If this parameter is set to a single percent sign, then all tables are returned. This parameter defaults to the current userid or alternate userid.
Table Name Filter Alias: TBFL	This parameter is used to filter the table names that are returned by the product. This parameter is only used if it is set to a non-blank value and if the host application does not provide a table name filter string. The table name filter string from either the application or from this parameter restricts the information that is returned by catalog inquiries (SQLTables, SQLColumns, SQLTablePrivileges). Only rows that match the table name pattern that is provided is returned. If this parameter is set to a single percent sign, then all tables are returned. There is no default value for this parameter.
Table Owner – Synonym Option Alias: SYOP	This parameter is used to show how table owners should be handled for tables that are actually synonyms. Some of these choices are required to make specific desktop productivity tools work with the DVS Direct ODBC driver. The default value is ZERO.
Table Qualifier Option Alias: TQOP	This parameter is used to specify how Table Qualifiers should be returned to ODBC application programs. Some databases use Table Qualifiers as part of the overall name of each table. In other words, two tables can have exactly the same name in all other respects, if their Table Qualifiers are different. This parameter is used to control how Table Qualifier information is returned to ODBC applications. There are three possible values. They are NORMAL, NULL, and ZERO. The default is NULL.
Trace Options Alias: TRACEFLAG	This parameter enables the options defined in the logging file.
Trace Path Alias: TRACEPATH	This parameter specifies the path to the logging file.
Transaction Name Alias: TRNA	This parameter is used to specify the transaction name. The use of the transaction name is application-specific. The transaction name can be up to 8 bytes in length. The transaction name is padded on the right with blanks.
Truncate Literal String Alias: TRLT	This parameter specifies whether literal parameters to stored procedures are changed to parameter markers and sent to the server as bound parameters. The default is YES.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Truststore Alias: TS	When using SSL, this parameter is the path that specifies the location of the truststore file. The truststore file contains a list of the valid Certificate Authorities (CAs) that are trusted by the client machine for SSL server authentication.
Truststore Password Alias: TSP	When using SSL and a PKCS12 encoded truststore, this parameter is the password that is required to access the truststore.
Upcase All Character Data Alias: UPCH	This parameter controls if all character data sent to the host should be converted to uppercase. If this parameter is set to YES, then all character data is converted to uppercase. If this parameter is set to NO, then character data is not converted to uppercase. The default is NO.
Upcase All Non-literals Alias: UPNL	This parameter controls if all non-literal values in SQL statements passed to the driver should be converted to uppercase. If this parameter is set to YES, then strings not in single or double quotation marks are converted to uppercase. The default is NO.
Upcase Double Quote String Alias: UPDB	This parameter controls if strings in double quotation marks should be converted to uppercase. If this parameter is set to YES, then strings in double quotation marks are converted to uppercase. If this parameter is set to NO, then strings in double quotation marks are not modified. Strings must be converted to uppercase in some cases because DB2 treats table names in double quotation marks as literals and table names must be uppercase. The default is NO.
Uppercasing Logon Credentials Alias: UCLC	This parameter is to allow the user to choose whether to uppercase userid/password (and new password if specified) or leave as entered. The default is YES, which means userid/password/new passwords are converted to uppercase before sending to host. NO means the userid/password/new password is left as entered.
User ID Alias: UID	Identifier for user account on target database
Validate Server Certificate Alias: VSC	Set this parameter to YES (the default) to validate the security certificate of the server as part of the SSL authentication handshake.
Wait for Transaction Completion Alias: WATR	This parameter controls if the client should wait for the Server to complete transaction processing when connected in VCF mode (TRANSACTION, TRANSBLOCK or MESSAGE). The default value is NO.
XML Describe Type Alias: XMDT	This parameter determines whether the driver maps XML data type to -1 (SQL_LONGVARCHAR) or -4 (SQL_LONGVARBINARY) data type. The default is -1 (SQL_LONGVARCHAR).
X/OPEN XA Support Alias: XAEN	This parameter is used to set the XA Transaction Manager type. This parameter must be set correctly when participating in a distributed transaction that is coordinated by a monitor. MSDTC sets the manager to the Microsoft Distributed Transaction Manager used mostly by MTS applications. The default is NONE which means that this is not a distributed transaction.

Table 3. ODBC connection properties (continued)

Parameter name	Description
Zero Column Names Alias: ZECL	This parameter is set to YES if column names should be set to binary zeros. This is a performance optimization for production applications. The default is NO.

Connection pooling

For Windows, connection pooling is configured through the ODBC Data Source Administrator.

Procedure

1. Select the **Connection Pooling** tab.
2. Highlight the **IBM Data Virtualization Manager for z/OS Driver** entry.

   Depending on the version of the Windows operating system, either

   - Enable/disable connection pooling and set the retry wait time directly on the controls that are displayed on the **Connection Pooling** tab, or
   - Double-click the **IBM Data Virtualization Manager for z/OS Driver** entry and enable/disable pooling and set the retry wait time in the pop-up window.

   For UNIX/Linux, connection pooling is managed by the ODBC Driver Manager. A driver manager is not included in the driver installation package; a third-party driver manager must be installed.

Optimized fetch

When optimized fetch is enabled, rows ahead of the current row are asynchronously extracted before the client application requests them. This data is then returned to the client application in blocks that may be as large as 32 KB. Enabling optimized fetch helps to minimize network traffic and speeds subsequent fetches as the requested data is likely already in a returned block.

Optimized fetch is enabled either by including the RO=YES connection property in a connection string (a connection string may be used with a DSN) or by appending the FOR FETCH ONLY clause to a SELECT statement.

When optimized fetch is enabled, cursor that is based deletes (DELETE WHERE CURSOR OF) cannot be used.

Chapter 4. DS Client high-level API

Use the DS Client high-level API to process SQL requests between high-level language applications running on z/OS and Data Virtualization Manager.

Data Virtualization Manager offers many ways to access virtualized data. The most popular are JDBC and ODBC. Data Virtualization Manager also provides an API that can be called from within more traditional mainframe languages such as COBOL, Natural or PL/I. This API is named the *DS Client high-level API*.

The DS Client high-level API allows an application running on z/OS to use a call-level interface to communicate with Data Virtualization Manager to process SQL requests and to retrieve result sets.

Result sets are buffered in a 64-bit shared memory object. The API user can choose to process the data in either of the following modes:

- *Move mode*. In move mode, the caller provides a buffer, and data is copied by the API from the memory object to the caller's buffer.
- *Locate mode*. In locate mode, the API is able to peek at the data and count the rows returned.

Load modules

The following table lists the name and location of the DS Client high-level API load modules.

Table 4. DS Client high-level API load modules

Description	**Load module or ALIAS**	***hlq*.SAVZLOAD**	*hlq***SAVZCLOD**
Batch interface routine	AVZCLIEN	Yes	
CICS PLT initialization routine	AVZXMTRI		Yes
CICS stub routine	AVZCLIEN		Yes
CICS task-related user exit	AVZCTRUE		Yes

Configuring access to DS Client for CICS

To use the DS Client API with CICS, you must modify your CICS configuration.

About this task

Configure CICS by modifying the CICS started task JCL, the program list table (PLT), and the DFHCSD file.

Procedure

1. Add the *hlq*.SAVZCLOD library to the DFHRPL concatenation in each CICS region that you want to connect to the server.
2. Update and assemble the CICS program list table/program initialized (PLTPI) list for the DS Client task-related user exit, as follows:
   a) In the PLTPI list, locate the first `DFHDELIM` entry:

   ```
   DFHPLT TYPE=ENTRY, PROGRAM=DFHDELIM
   ```

   b) After the first `DFHDELIM` entry in the PLTPI list, add the following entry for the DS Client task-related user exit:

   ```
   DFHPLT TYPE=ENTRY, PROGRAM=AVZXMTRI
   ```

Note: The entry for the AVZXMTRI program must follow the first DFHDELIM entry in the PLTPI list to ensure that the AVZXMTRI program will be executed in the second stage of the CICS PLTI process.

c) Run the assembly job.

3. Update the DFHCSD file by performing the following steps:
   a) For each CICS region, modify and submit the AVZCICSD job that is in *hlq*.SAVZCNTL data set.
   b) Update LIST(*YOURLIST*) to match the startup group list for the CICS region.
4. Restart CICS and check for the following message in the CICS job log:

```
AVZ4459I CICSE DS Client exit program AVZCTRUE is enabled
```

AVZCLIEN

A DS Client high-level application is implemented by passing a parameter list on a set of calls to module AVZCLIEN. The parameter list consists of the following:

1. **DVCB control block address.** The DVCB address is always required.
2. **Send buffer address**. The send buffer address is required for a SEND call and ignored for other calls.
3. **Receive buffer address**. The receive buffer address is required for a RECV call in move mode. A caller in locate mode should pass a zero in this argument. For other calls, it is ignored.
4. **Message buffer address**. The message buffer address is optional for a RECV call. For other calls, it is ignored.
5. **SQLDA buffer address**. The SQLDA buffer address is optional for a RECV call in move mode. A caller in locate mode should pass a zero in this argument. For other calls, it is ignored.

The application can be linked with SDXMBIS, or LOAD the module and branch to it.

DVCB control block

The DS Client Control Block (DVCB) is the main control structure between the application and DS Client.

All DS Client calls require the address of a DVCB as the first argument in the parameter list.

The calling program needs to have a DVCB defined in its working storage.

DVCB fields

The following table provides a summary of the fields in the DVCB. The “IN/OUT” column tells whether the field is input by the application, output by the API, or both input and output. The code in each column representing an API call tells whether the field is (R)equired, (O)ptional, or (I)gnored.

Name	Description	IN/ OUT	OPEN	SEND	RECV	CLOS
#DVCB-TAG	Eyecatcher = ‘DVCB’	IN	R	R	R	R
#DVCB-TAG2	Eyecatcher = ‘DVCB’	IN	R	R	R	R
#DVCB-VERSION	Control block version = 1	IN	R	R	R	R
#DVCB-SQL-CODE	SQLCODE	OUT	I	I	I	I
#DVCB-DATA-BUFFER-LENGTH	Data buffer length (move mode)	IN	I	I	R	I

Name	Description	IN/ OUT	OPEN	SEND	RECV	CLOS
#DVCB-DATA-RETURNED-LENGTH	Returned (or required) data length	OUT	I	I	I	I
#DVCB-ROWS-RETURNED	Number of rows returned	OUT	I	I	I	I
#DVCB-REQUEST-CODE	Four-character request code	IN	R	R	R	R
#DVCB-SERVER-GROUP	DS Client server group name	IN	O	I	I	I
#DVCB-SSID	Data Virtualization Manager subsystem name	IN	O	I	I	I
#DVCB-USER-PARM	A field for the user	IN	I	I	I	I
#DVCB-OPT-RECV-MODE	Receive mode option	IN	I	I	O	I
#DVCB-OPT-AUTO-COMMIT	Auto-commit option	IN	I	O	I	I
#DVCB-OPT-CLOSE-AFTER	Auto-close after request	IN	I	O	O	I
#DVCB-OPT-SQLDA	Request SQLDA	IN	I	O	I	I
#DVCB-OPT-PRESERVE-ORDER	Preserve row order with MRC	IN	O	I	O	I
#DVCB-BLOCKING-TIMEOUT	Timeout value for blocking RECV	IN	I	I	O	I
#DVCB-SEND-LENGTH	Length of SQL statement string	IN	I	R	I	I
#DVCB-CNID	Connection handle	OUT/IN	I	R	R	R
#DVCB-RETURN-CODE	API return code	OUT	I	I	I	I
#DVCB-ROW-LENGTH	Length of rows returned	OUT	I	I	I	I
#DVCB-SQLDA-LENGTH	SQLDA buffer length	IN	I	I	O **Note:** Required in move mode when the fifth argument (SQLDA buffer address) is non-zero.	I
#DVCB-MESSAGE-LENGTH	Message buffer length	IN/ OUT	I	I	O **Note:** Required in move mode when the fourth argument (message buffer address) is non-zero.	I

Name	Description	IN/ OUT	OPEN	SEND	RECV	CLOS
#DVCB-ROW-RETURNED	Row data is returned	OUT	I	I	I	I
#DVCB-SQLCODE-RETURNED	A SQLCODE is returned in #DVCB-SQL-CODE	OUT	I	I	I	I
#DVCB-MESSAGE-RETURNED	A message is returned	OUT	I	I	I	I
#DVCB-SQLDA-RETURNED	A SQLDA is returned	OUT	I	I	I	I
#DVCB-END-OF-DATA	End of data	OUT	I	I	I	I
#DVCB-ERROR-RETURNED	An API error is returned in #DVCB-RETURN-CODE	OUT	I	I	I	I
#DVCB-PARMS-RETURNED	Variables are returned from a stored procedure	OUT	I	I	I	I
#DVCB-END-OF-RSET	End of result set	OUT	I	I	I	I
#DVCB-ROW-LIMIT	Limit the number of rows	IN	I	O	I	I
#DVCB-USERID	User ID	IN	O	I	I	I
#DVCB-PASSWORD	Password	IN	O	I	I	I
#DVCB-MAPREDUCE-ID	Map Reduce Client id	IN	O	I	I	I
#DVCB-MAPREDUCE-NO	Map Reduce Client number of threads	IN	O	I	I	I

Example

Following is the Natural data declaration of the DVCB.

```
* ------------------------------------------------------------------ *
*   --- DVCB API CONTROL BLOCK ---
* ------------------------------------------------------------------ *
1 #DVCB (B1/1:256)
1 REDEFINE #DVCB
  2  #DVCB-TAG                  (A4)   /* EYECATCHER              (IN)
  2  #DVCB-VERSION              (I2)   /* MACRO VERSION           (IN)
  2  #DVCB-RESERVED1            (A2)   /* RESERVED
  2  #DVCB-SSID                 (A4)   /* DVM SUBSYSTEM NAME      (IN)
  2  #DVCB-REQUEST-CODE         (A4)   /* REQUEST CODE            (IN)
  2  #DVCB-CNID                 (B16)  /* CONNECTION HANDLE       (OUT)
  2  REDEFINE #DVCB-CNID
     3 #DVCB-CONNECTION         (B12)
     3 #DVCB-CONNECTED-SSID     (A4)
  2  #DVCB-SERVER-GROUP         (A8)   /* DVM SERVER GROUP NAME   (IN)
  2  #DVCB-USER-PARM            (A8)   /* A FIELD FOR THE USER
  2  #DVCB-SQL-CODE             (I4)   /* SQL CODE                (OUT)
  2  #DVCB-DATA-BUFFER-LENGTH   (I4)   /* DATA BUFFER LENGTH      (IN)
  2  #DVCB-DATA-RETURNED-LENGTH (I4)   /* RETURNED DATA LENGTH    (OUT)
  2  #DVCB-RESERVED2            (I4)   /* RESERVED
  2  #DVCB-ROWS-RETURNED        (I4)   /* NUMBER OF ROWS RETURNED  (OUT)
  2  #DVCB-OPTIONS              (A8)   /* OPTION FLAGS             (IN)
  2  REDEFINE #DVCB-OPTIONS
     3  #DVCB-OPT-RECV-MODE     (A1)   /* LOCATE OR MOVE MODE
     3  #DVCB-OPT-AUTO-COMMIT   (A1)   /* SET AUTO-COMMIT
     3  #DVCB-OPT-CLOSE-AFTER   (A1)   /* CLOSE ON NEXT REQUEST
     3  #DVCB-OPT-RESERVED      (A1)   /* RESERVED
     3  #DVCB-OPT-SQLDA         (A1)   /* REQUEST SQLDA
     3  #DVCB-OPT-PRESERVE-ORDER (A1)  /* PRESERVER ROW ORDER WITH MRC
  2  #DVCB-BLOCKING-TIMEOUT     (I4)   /* TIMEOUT FOR BLOCKING RECV (IN)
  2  #DVCB-SEND-LENGTH          (I4)   /* LENGTH OF SEND BUFFER     (IN)
  2  #DVCB-RETURN-CODE          (I4)   /* RETURN CODE              (OUT)
  2  #DVCB-DB2-SUBSYSTEM        (A4)   /* DB2 SUBSYSTEM NAME       (IN)
  2  #DVCB-ROW-LENGTH           (I4)   /* LENGTH OF ROWS RETURNED  (OUT)
```

```
  2  #DVCB-SQLDA-LENGTH         (I4)   /* SQLDA BUFFER LENGTH      (IN)
  2  #DVCB-MESSAGE-LENGTH       (I4)   /* MESSAGE BUFFER LENGTH    (IN)
  2  #DVCB-RESERVED3            (A50)  /* RESERVED
  2  #DVCB-RETURN-FLAGS         (A8)   /* RETURN FLAGS             (OUT)
  2  REDEFINE #DVCB-RETURN-FLAGS
     3 #DVCB-ROW-RETURNED       (A1)   /* ROW DATA IS RETURNED Y/N
     3 #DVCB-SQLCODE-RETURNED   (A1)   /* SQLCODE IS RETURNED Y/N
     3 #DVCB-MESSAGE-RETURNED   (A1)   /* MESSAGE IS RETURNED Y/N
     3 #DVCB-SQLDA-RETURNED     (A1)   /* SQLDA IS RETURNED Y/N
     3 #DVCB-END-OF-DATA        (A1)   /* END OF DATA IS RETURNED Y/N
     3 #DVCB-ERROR-RETURNED     (A1)   /* ERROR IS RETURNED Y/N
     3 #DVCB-PARMS-RETURNED     (A1)    /* DB2 ST PRC PARMS RETURNED Y/N
     3 #DVCB-END-OF-RSET        (A1)   /* END OF RSLTSET IS RETURNED Y/N
  2  #DVCB-RESERVED4            (A2)   /* RESERVED
  2  #DVCB-ROW-LIMIT            (I4)   /* ROW LIMIT                (IN)
  2  #DVCB-USERID               (A8)   /* CLIENT USERID OVERIDE    (IN)
  2  #DVCB-PASSWORD             (A8)   /* CLIENT PASSWORD OVERRIDE (IN)
  2  #DVCB-MAPREDUCE-ID         (I2)   /* MAP REDUCE CLIENT ID     (IN)
  2  #DVCB-MAPREDUCE-NO         (I2)   /* MAP REDUCE THREAD NO     (IN)
  2  #DVCB-RESERVED4            (A64)  /* RESERVED
  2  #DVCB-TAG2                 (A4)   /* STORAGE CHECK EYECATCHER (IN)
*   --- DVCB API EQUATES BLOCK ---
1  #EYECATCHER        (A4) CONST<'DVCB'>
1  #OPEN-REQUEST      (A4) CONST<'OPEN'>
1  #CLOSE-REQUEST     (A4) CONST<'CLOS'>
1  #SEND-REQUEST      (A4) CONST<'SEND'>
1  #RECEIVE-REQUEST   (A4) CONST<'RECV'>
1  #CLOSE-CONNECTION  (A4) CONST<'Y'>
1  #LOCAL-MODE        (A4) CONST<'L'>
1  #MOVE-MODE         (A4) CONST<'M'>
1  #END-OF-RESULT     (I4) CONST<2>
1  #ROW-RETURNED      (L)
1  #SQLCODE-RETURNED  (L)
1  #MESSAGE-RETURNED  (L)
1  #SQLDA-RETURNED    (L)
1  #END-OF-DATA       (L)
1  #ERROR-RETURNED    (L)
1  #CNID              (B16)  /* CONNECTION HANDLE       (OUT)
```

DS Client requests

There are four request types supported by DS Client: OPEN, SEND, RECV, and CLOS.

OPEN

The OPEN request creates a session with the DS Client server, and obtains the shared memory object used to buffer the result set(s). The server can be identified either by subsystem name or by server group name. (The group name is configured in AVZSIN00 using parameter DSCLIENTGROUP. Connections to the same server group are distributed in round-robin fashion.) The API will return a connection handle in #DVCB-CNID; this handle must be maintained and returned on all subsequent API calls.

The OPEN call is optional. If a connection is not open when a request is sent, an implicit OPEN will be performed.

The following options can be presented on the OPEN call:

- The subsystem name or group name that identifies the Data Virtualization Manager server.
- The Db2 subsystem name (or none to use SQL Engine).
- The row limit for the result set(s)
- The auto-commit option
- The request SQLDA option
- The user ID and password to be used for access to the data source. The default is the API application user ID.

Providing alternate credentials

By default, the server will use the API application address space user ID to access the data source. Alternate credentials can be provided in #DVCB-USERID and #DVCB-PASSWORD. If #DVCB-USERID is non-zero, a password must be provided in #DVCB-PASSWORD. The application should clear #DVCB-PASSWORD to zeros after the OPEN call.

Map Reduce Client

Map Reduce Client (MRC) technology provides the ability for a client application to process large requests in parallel on multiple threads. Depending on the data source and the system architecture, significant reductions in elapsed time can be achieved using MRC. The data source must be enabled for Map Reduce in order to benefit from MRC.

To achieve end-to-end parallel processing using MRC, first determine the number of processing threads to use. Start or attach this number of client threads, and open a DS Client connection on each thread. In each DVCB, set #DVCB-MAPREDUCE-NO equal to the total number of threads. Set #DVCB-MAPREDUCE-ID in each thread to a unique value 1 – *n*, where *n* is the value of #DVCB-MAPREDUCE-NO. Send the same request on each client thread. The aggregate of all *n* composite result sets will be the complete result set for the request.

Note: If Map Reduce is not enabled for the data source, the complete result set will be returned to the thread which has specified #DVCB-MAPREDUCE-ID = 1. Threads 2 - *n* will immediately receive SQLCODE 100.

Map Reduce Client can also significantly reduce the elapsed time for large queries when running the application on a single thread, especially for locate-mode requests, and move-mode requests for which the row order is not essential. To use MRC on a single thread, set #DVCB-MAPREDUCE-NO equal to the total number of server threads desired, and set #DVCB-MAPREDUCE-ID to zero. For move-mode requests, set #DVCB-OPT-PRESERVE-ORDER to N if row order is not necessary. (The default is Y to preserve the row order of the result set.)

Note: MRC with #DVCB-OPT-PRESERVE-ORDER = 'Y' is generally slower than the server's normal Map Reduce processing. Do not use this option unless row order is essential.

Example of an OPEN call in Natural

```
ASSIGN #DVCB-REQUEST-CODE = #OPEN-REQUEST
CALL 'AVZCLIEN' #DVCB(*) #SEND-AREA(*) #RECEIVE-AREA(*) #ERROR-AREA
     #SQLDA-AREA(*)
```

Connection handle

An explicit or implicit open creates a 16-byte connection handle, which is returned in field #DVCB-CNID. The application is responsible to preserve this value and present it intact on all future API calls. If the application must save and restore the connection handle, it must save the value after each OPEN or SEND API call.

SEND

The SEND request must be the first request following OPEN. It communicates the SQL request to the server. The SQL statement must be in the buffer pointed to by the second argument of the parameter list. The request is passed to Data Virtualization Manager for processing, and the result set immediately begins to flow into the shared memory object.

The send request places the session in *request received* state. When the request completes, the session is reverted to *open* state, at which time another SEND request may be made, or a CLOS request sent to close the session.

The following options can be presented on the SEND call:

- The Db2 subsystem name (or none to use SQL Engine).
- The row limit for the result set(s)
- The auto-commit option
- The request SQLDA option

Setting the DB2 Subsystem name

The Db2 subsystem name is set by putting the four-character subsystem name in field DVCBDB2S. It can be set on the OPEN call, and changed on the SEND call. If DVCBDB2S is binary zeros or blanks, the request will be processed by z/SQL. To remove a DB2 subsystem name from a previous query, move binary zeros or EBCDIC blank characters to DVCBDB2S.

Requesting auto-commit

To turn auto-commit on, set #DVCB-OPT-AUTO-COMMIT to A before calling SEND. An implicit commit will be performed after each update request. To turn auto-commit off, set #DVCB-OPT-AUTO-COMMIT to N.

Example of a SEND call

```
ASSIGN #DVCB-REQUEST-CODE = #SEND-REQUEST
ASSIGN #DVCB-SEND-LENGTH  = #SEND-LENGTH
CALL 'AVZCLIEN' #DVCB(*) #SEND-AREA(*) #RECEIVE-AREA(*) #ERROR-AREA
     #SQLDA-AREA(*)
```

RECV

The RECV request reads the result set data from the shared memory object. A SQLDA is also optionally returned. In move mode, the data is copied to the buffer pointed to by the third argument of the parameter list.

The session must be in *request received* state when the RECV request is made.

The RECV call will return when the caller's buffer is full, when the current result set is exhausted, or when an error occurs. To reduce the number of RECV calls required, make the data buffer as large as possible.

The buffer must be large enough to hold at least one complete row, or an error (40) will be returned in #DVCB-RETURN-CODE. When this happens, the required buffer size will be returned in #DVCB-DATA-RETURNED-LENGTH. The application should obtain a buffer of at least this size, update #DVCB-DATA-BUFFER-LENGTH with the new length, and retry the request, passing the address of the new buffer. The buffer length can be set to zero to request only a SQLDA and no row data. In this case, the return code will be zero.

The following options can be presented on the RECV call:

- An optional timeout.
- Auto-close after request completion.

Rows are returned in a fixed length, with all variables padded to their maximum length. The row size is returned, along with the number of rows, the first-row address, and the SQLCODE.

SQLDA-format data

A SQLDA is needed to process the rows. SQLDATA and SQLIND in the returned SQLDA hold the offsets into the row of the column data and the null indicator. A null column is indicated by a combination of an odd value in SQLTYPE, and a value of x'FFFF' in the area indicated by SQLIND.

Note: The value of SQLIND is always 0 in the first variable. The SQLTYPE field must be examined to determine the significance of SQLIND.

Return signals

The API returns *signals*, or flags, in the DVCB from each RECV call to indicate the returned data and indicators. All signals should be processed in turn before the next RECV call.

Signal	Description
#DVCB-ROW-RETURNED = 'Y'	One or more rows are returned in the area pointed to by the third argument of the calling parameter list. Use #DVCB-ROWS-RETURNED and #DVCB-ROW-LENGTH to process the rows. **Note:** This flag is not set in locate mode.
#DVCB-MESSAGE-RETURNED = 'Y'	A message is returned from the server in the area pointed to by the fourth argument of the calling parameter list. The length of the message text is in #DVCB-MESSAGE-LENGTH. Print or display the message before continuing.
#DVCB-SQLDA-RETURNED = 'Y'	A SQLDA is returned in the area pointed to by the fifth argument of the calling parameter list.
#DVCB-ERROR-RETURNED = 'Y'	An error is returned from the API. Check the return code in #DVCB-RETURN-CODE. Most errors are permanent; these will be accompanied by the end-of-data signal. Requests that result in transient errors, such as when the buffer is too small (return code 40) can be retried.
#DVCB-PARMS-RETURNED = 'Y'	A stored procedure has returned parameter data. The parameters will be presented on the first RECV call as a single row of data. Flag #DVCB-ROW-RETURNED will also be set to Y. The row will contain all parameters, including IN parameters, in the order they are defined on the CREATE statement. IN parameters will be null.
#DVCB-END-OF-RSET = 'Y'	When the end of a result set is reached, return flag #DVCB-END-OF-RSET will be set toY. If #DVCB-END-OF-DATA is set to N, there are more result sets to follow.
#DVCB-END-OF-DATA = 'Y'	The request is complete and no more data is available. Check #DVCB-ERROR-RETURNED or #DVCB-SQLCODE-RETURNED to determine the reason for the end of data.
#DVCB-SQLCODE-RETURNED = 'Y'	A SQLCODE is available in #DVCB-SQL-CODE.

Note: Multiple signals can be returned together. It is essential, for instance, to check #DVCB-ROW-RETURNED and process the data returned before checking #DVCB-END-OF-DATA.

Example of a move-mode RECV call

```
ASSIGN #DVCB-REQUEST-CODE       = #RECEIVE-REQUEST
ASSIGN #DVCB-DATA-BUFFER-LENGTH = #RECEIVE-LENGTH
CALL 'AVZCLIEN' #DVCB(*) #SEND-AREA(*) #RECEIVE-AREA(*) #ERROR-AREA
     #SQLDA-AREA(*)
```

CLOS

The CLOS request destroys the session and frees all shared resources obtained by the API for the connection.

The CLOS call is optional. If #DVCB-OPT-CLOSE-AFTER is set to Y, the connection will be implicitly closed after completion of the current request.

Example of a CLOS call

```
ASSIGN #DVCB-REQUEST-CODE = #CLOSE-REQUEST
CALL 'AVZCLIEN' #DVCB(*) #SEND-AREA(*) #RECEIVE-AREA(*) #ERROR-AREA
     #SQLDA-AREA(*)
```

Idle timeout

A DS Client connection that is idle for a prolonged period will be timed out by the server. The timeout threshold is configured in AVZSIN00 using the DSCLIENTCONNTIMEOUT parameter.

When a session times out, the shared memory object is detached. Subsequent RECV calls will fail with a no-connection error (return code 32). A new connection can be created with an OPEN or SEND call.

API return codes

The following return codes are returned from DS Client. The return code is moved into DVCBRETC, and is also in general register 15 on return from the DS Client call.

Return code	Description
2	This code is returned if a SQL code is returned from a RECV operation. Check the value in #DVCB-SQL-CODE. This code is returned when the request is ended. The session is reverted to open state, and the next request should be SEND or CLOS. A RECV call following this code will be rejected with a state error.
8	This code is returned if DS Client is not active in the server. DS Client is activated using the DSCLIENTACTIVE parameter in AVZSIN00.
12	This code is returned if storage is not available in the DS Client server to process the request.
16	This code is returned if the DS Client server is not able to obtain or share the memory object buffer.
20	This code is returned if no DS Client server is available to process the request, or if the specified server does not support DS Client.
24	This code is returned if the server timed the session out for inactivity. The timeout time is set with the DSCLIENTCONNTIMEOUT parameter in AVZSIN00.
32	This code is returned if the session indicated by the connection handle is not found in the server.

Return code	Description
36	This code is returned if a request is made while the connection is in the wrong state. The state requirements are discussed under the individual requests.
40	This code is returned if the data to be returned is too large for the buffer. This code can be returned on a RECV call in move mode. The required buffer size will be in #DVCB-DATA-RETURNED-LENGTH. Provide a larger buffer, put its size in #DVCB-DATA-BUFFER-LENGTH, and retry the request.
44	This code is returned if the caller supplied a timeout value, and the RECV call timed out while waiting for data.
56	This code is returned if a shared memory object is required, but a recovery environment cannot be built to ensure the SMO is freed.
60	This code is returned if the required SQL statement length #DVCB-SEND-LENGTH is not provided on a SEND.
64	This code is returned if #DVCB-TAG2 is not DVCB.
68	This code is returned if a RECV request is made in move mode, and the receive buffer address in the parameter list is zero, or the buffer length in #DVCB-DATA-BUFFER-LENGTH is zero.
72	This code is returned if a RECV request is made in locate mode but the connection was opened in move mode.
80	This code is returned if the request code in #DVCB-REQUEST-CODE is unrecognized.
84	This code is returned if the SQLDA to be returned is too large for the buffer. This code can be returned on a RECV call in move mode. If the buffer is at least 16 bytes, the SQLDA prolog will be copied into it. The required SQLDA size will be in field SQLDABC. Obtain a larger buffer, provide its length in #DVCB-SQLDA-LENGTH, and retry the request.
88	This code is returned if the value of #DVCB-TAG is something other than C'DVCB'.
92	This code is returned if the value of #DVCB-VERSION is not supported by the API.
100	This code represents an internal error.
104	This code is returned if a user ID is provided in #DVCB-USERID without a password in #DVCB-PASSWORD.
108	This code is returned if an error is found in the Map Reduce Client fields on the OPEN call: • #DVCB-MAPREDUCE-ID is greater than #DVCB-MAPREDUCE-NO, or • #DVCB-MAPREDUCE-NO is non-zero but #DVCB-MAPREDUCE-ID is zero, or • #DVCB-MAPREDUCE-NO is greater than the value of ACIMAPREDUCECLIMAX in AVZSIN00.
116	This code is returned on an OPEN call if there is an incompatibility between the API and the server.
120	This code is returned when a non-recoverable error occurs while processing the request. A SQL code is returned in XMPLSQLC. The server ends the session. The application should close the connection.

Return code	Description
124	This code is returned if the security environment is invalid.
128	This code is returned when an abnormal termination occurs in the server while processing the request.

DS Client configuration

The following Data Virtualization Manager server configuration options affect DS Client operations. The required parameters are DSCLIENTACTIVE and LICENSECODE. The rest are optional.

Parameter	Description	Values
ACIDVCLIENTMIN	Controls the number of internal ACI servers to keep active for future DS Client requests.	*servers* Default: 2. Minimum: 1 Maximum: 10000
ACIMAPREDUCECLIMAX	Specifies the maximum number of MAP REDUCE tasks that will be started to process a request from a DS Client application. Specify 0 to disable MAP REDUCE CLIENT.	Default: 20 Minimum: 0 Maximum: 255
DSCLIENTACTIVE	Controls whether or not the DS Client server will be activated. This parameter is required.	YES|NO
DSCLIENTBUFFERSIZE	Specifies the default buffer size, in megabytes above the bar, for a DS Client API connection.	*megabytes*
DSCLIENTBUFFERNUMMAX	Specifies the maximum number of buffers to allow for a DS Client connection. The total maximum storage obtained will be = (DSCLIENTBUFFERSIZEMAX * DSCLIENTBUFFERNUMMAX).	*buffers* Default: 1
DSCLIENTBUFFERSIZEMAX	Specifies the maximum buffer size, in megabytes above the bar, for a DS Client connection. The buffer size requested by the application (if any) is negotiated against this value.	*megabytes*
DSCLIENTBUFFERSIZEMIN	Specifies the minimum buffer size, in megabytes above the bar, for a DS Client connection. The buffer size requested by the application (if any) is negotiated against this value.	*megabytes*
DSCLIENTCONNTIMEOUT	Specifies the maximum time, in minutes, a session will remain idle before the server closes it.	*minutes*
DSCLIENTGROUP	Allows the DS Client user to choose a server group to use for DS Client access. One or more DS Client servers can be configured with the same group name, and DS Client will distribute connections in round robin.	*group name*

Parameter	Description	Values
DSCLIENTNUMROWSMAX	Specifies the maximum number of rows that may be requested for each RECV call on a DS Client connection.	*rows*
DSCLIENTSPMDCACHEMAX	Specifies the maximum number of entries to maintain in the stored procedure metadata cache.	*entries* Maximum: 32767
DSCLIENTSPMDCACHEMIN	Specifies the minimum number of entries to maintain in the stored procedure metadata cache. Specify zero to disable metadata caching.	*entries*
DSCLIENTTRACE	Controls whether the DS Client server will trace its internal state changes.	YES|NO
DSCLIENTTRACEDB	Controls whether DS Client processing will trace debugging messages.	YES|NO
DSCLIENTTRACESQL	If set to YES, formats all the related SQL blocks to Trace Browse after each DS Client SQL request.	YES|NO
LICENSECODE	DS Client is a licensed feature. The LICENSECODE must support DS Client. This parameter is required.	*code*

Batch program execution

The method to invoke an existing module that uses Data Virtualization Manager as well does not change, so no special instructions are needed.

More specifically, the invocation of a module will also stay the same for modules running in BMP regions or modules attaching to Db2 for z/OS.

In this example, the module is called using EXEC PGM=, which does not require a BMP region or a connection to Db2 for z/OS.

```
//STEP1 EXEC PGM=module,REGION=4096K
//STEPLIB DD DSN=APPL.LOADLIB,DISP=SHR
// DD DSN=DVM.SAVZLOAD,DISP=SHR
```

Example: Using Data Virtualization Manager in a COBOL program

This example shows how a non-relational data source can be accessed via SQL from a COBOL program using the DS Client high-level API.

About this example

Assume that you have a VSAM data set that has been virtualized as a virtual table STAFFVS, and you need to access it from a program that is written in a high-level language such as COBOL.

Data Virtualization Manager offers a DS Client high-level language API through the program AVZCLIEN that is utilized with this sequence of requests:

1. Open a connection to the Data Virtualization Manager server
2. Send the SQL command to the Data Virtualization Manager server

3. Receive the data back from the Data Virtualization Manager server until all data of this SQL command is received
4. Close the connection to the Data Virtualization Manager server

When multiple SQL commands are needed, each of them must be completely received before the next one can be sent. For more information, see “Multiple SQL commands” on page 70.

The program’s data structures

The COBOL program needs to have a DS Client Control Block (DVCB) defined in its working storage and a copy book can be used for that. See “Example: DS Client control block (DVCB)” on page 71.

Various other variables are also needed and they are discussed when they are referenced. See “Other useful variables” on page 71.

Program preparation

The program should be compiled without any specific options.

The program should be linked as follows:

```
//SYSLIB DD DSN=DVM.SAVZLOAD,DISP=SHR
//SYSLMOD DD DSN=APPL.LOADLIB,DISP=SHR
//SYSIN DD *
INCLUDE SYSLIB(AVZCLIEN)
NAME module(R)
```

AVZCLIEN is to be found in the Data Virtualization Manager SAVZLOAD library.

Batch program execution

The method to invoke an existing module that uses Data Virtualization Manager as well does not change, so no special instructions are needed.

More specifically, the invocation of a module will also stay the same for modules running in BMP regions or modules attaching to Db2 for z/OS.

In this example, the module is called using EXEC PGM=, which does not require a BMP region or a connection to Db2 for z/OS.

```
//STEP1 EXEC PGM=module,REGION=4096K
//STEPLIB DD DSN=APPL.LOADLIB,DISP=SHR
// DD DSN=DVM.SAVZLOAD,DISP=SHR
```

OPEN request

The following example shows an OPEN request:

```
MOVE 'AVZC' TO DVCB-SSID.
MOVE MESSAGE-LENGTH TO DVCB-MESSAGE-LENGTH.
MOVE FCT-OPEN TO DVCB-REQUEST-CODE.
MOVE 'Y' TO DVCB-OPT-SQLDA.
MOVE SQLDA-LENGTH TO DVCB-SQLDA-LENGTH.
CALL 'AVZCLIEN' USING DVCB SEND-AREA RECEIVE-AREA
ERROR-AREA SQLDA-AREA.
PERFORM 5000-CHECK-RESPONSE.
```

Notes:

- The first MOVE identifies the name of the Data Virtualization Manager server that you will work with.
- The second MOVE initializes the message length that will be passed to the server
- The third MOVE will put 0PEN in the request.
- The fourth MOVE tells the server that, upon RECV time, metadata information (mostly for debugging) must be returned in the area that is identified by the fifth parameter of the CALL. Metadata is returned

for each column in the result set. See “Example: DS Client control block (DVCB)” on page 71 for the structure of this area.

- The fifth MOVE sets the length of this area of metadata information.

The OPEN request will initiate the communication with the Data Virtualization Manager server through the API called AVZCLIEN.

After the CALL statement is executed, the various one-byte fields in the DVCB-RETURN-FLAGS group of the DVCB can be tested for a Y value. See the contents of a sample paragraph `5000-CHECK-RESPONSE` in “Typical error checking” on page 72.

Note that a Y for such a flag is not necessarily an error (they are informational), except for DVCB-ERROR-RETURNED which might be a reason to stop the execution of the program.

The following points are about the metadata information in SQLDA-AREA (the fifth parameter of the CALL):

- The header is 16 bytes long, where SQLDAID is an eye-catcher with the value SQLDA and SQLDABC contains the total length of SQLDA as returned by the server.
- Each repeating SQLVAR group (44 bytes long) has the name of the column in SQLNAME and the length of the name in SQLNAME-LENGTH.

SEND request

The following example shows a SEND request:

```
MOVE FCT-SEND TO DVCB-REQUEST-CODE.
MOVE SEND-LENGTH TO DVCB-SEND-LENGTH.
CALL 'AVZCLIEN' USING DVCB SEND-AREA RECEIVE-AREA
ERROR-AREA SQLDA-AREA.
PERFORM 5000-CHECK-RESPONSE.
```

Notes:

- The first MOVE will put SEND in the request.
- The second MOVE initializes the length of SEND-AREA into the DVCB, in practice the length of the SQL command. SEND-AREA is a placeholder for the SQL command that you want to pass to the Data Virtualization Manager server.

Again, check for eventual return information with `5000-CHECK-RESPONSE` in “Typical error checking” on page 72.

You must build the SQL command yourself in your COBOL program, maybe using character functions in the COBOL language. An example is shown as follows:

```
STRING
"SELECT STAFFVS_KEY_ID,"
DELIMITED BY SIZE
" STAFFVS_DATA_NAME,"
DELIMITED BY SIZE
" STAFFVS_DATA_DEPT,"
DELIMITED BY SIZE
" STAFFVS_DATA_JOB,"
DELIMITED BY SIZE
" STAFFVS_DATA_YRS"
DELIMITED BY SIZE
" FROM STAFFVS"
DELIMITED BY SIZE
INTO SQL-TEXT.
```

This example results in the following SQL command being sent:

```
SELECT STAFFVS_KEY_ID, STAFFVS_DATA_NAME, STAFFVS_DATA_DEPT, STAFFVS_DATA_JOB,
       STAFFVS_DATA_YRS FROM STAFFVS
```

where STAFFVS is the virtualized table.

For this example, the length of SQLDA-AREA as returned by the server in SQLDABC is 236 (hex EC): a fixed header of 16 bytes and five groups of 44 bytes. The following example shows a hexadecimal representation of SQLDA-AREA (first repeating group only):

```
=COLS> ----+----1----+----2----+----3----+----4----+----5----+----6
SQLDA \ 4 STAFFVS_KEY_ID
EDDCC444000E00000F000000000000EECCCEE6DCE6CC4444444444444444
28341000000C05051402000000000E2316652D258D940000000000000000
```

RECEIVE request

Data Virtualization Manager stores the results of the SQL command in the storage of the Data Virtualization Manager started task. Those results must be transferred into the program's working storage. As the program's working storage normally is much smaller than the server storage, you must request reception of the server's results in a repetitive process until Data Virtualization Manager tells you that all data has been received. This concept can be compared to using multi-row fetch from an application with Db2 for z/OS.

The following example shows a way to receive the results repetitively:

```
PERFORM 4000-PROCESS-RESULTS
UNTIL DVCB-END-OF-DATA = 'Y'
OR DVCB-ERROR-RETURNED = 'Y'.
```

UNTIL forces at least one execution of 4000-PROCESS-RESULTS.

The following example shows a code snippet to process the result set:

```
*****************************************************************
4000-PROCESS-RESULTS.
*****************************************************************
* UNCOMMENT THE COMMENTS TO SHOW ONLY FIRST RESULT SET *
*****************************************************************
MOVE 0 TO DVCB-ROWS-RETURNED.
PERFORM 3000-GET-RESULTS.
DISPLAY 'RESULT-SET #' RESULT-SET-NUMBER
PERFORM TEST AFTER VARYING W1 FROM 1 BY 1
UNTIL W1 = DVCB-ROWS-RETURNED
DISPLAY DVC-ID (W1) ' '
DVC-NAME (W1) ' '
DVC-DEPT (W1) ' '
DVC-JOB (W1) ' '
DVC-YEARS (W1) ' '
END-PERFORM
ADD 1 TO RESULT-SET-NUMBER.
```

Paragraph 4000-PROCESS-RESULTS is executed until Data Virtualization Manager communicates the end of the Data Virtualization Manager result set or the presence of an error.

Inside 4000-PROCESS-RESULTS, some of the coding is for data processing after data has been received. The data is received with the execution of paragraph 3000-GET-RESULTS.

```
*****************************************************************
3000-GET-RESULTS.
*****************************************************************
MOVE FCT-RECEIVE TO DVCB-REQUEST-CODE.
MOVE RECEIVE-LENGTH TO DVCB-DATA-BUFFER-LENGTH.
CALL 'AVZCLIEN' USING DVCB SEND-AREA RECEIVE-AREA
ERROR-AREA SQLDA-AREA.
PERFORM 5000-CHECK-RESPONSE.
DISPLAY 'NUMBER OF ROWS IN BUFFER:' DVCB-ROWS-RETURNED.
DISPLAY 'ROW LENGTH:' DVCB-ROW-LENGTH.
ADD DVCB-ROWS-RETURNED TO TOTAL-ROWS.
```

The first MOVE will put RECV in the request and the second MOVE initializes the length of RECEIVE-AREA into the DVCB. This length should be greater than the largest row returned.

RECEIVE-AREA will receive the data back. Note that RECEIVE-AREA can receive many rows of the Data Virtualization Manager result set.

In this example, the RECEIVE-AREA has 200 positions that can hold 10 rows of the Data Virtualization Manager result set (one row is 20 positions). The first execution of 3000-GET-RESULTS will return the rows 1 to 10, and the fourth execution will return rows 31 to 35 (as there are 35 rows in the result set).

The number of rows returned is in DVCB-ROWS-RETURNED.

CLOSE request

The CLOSE request is not required when the field DVCB-OPT-CLOSE-AFTER has been set to Y.

This can be achieved with the following code example:

```
MOVE OPT-AUTOCLOSE TO DVCB-OPT-CLOSE-AFTER.
```

This is an indication that the connection to the Data Virtualization Manager server is automatically closed when the RECV request has delivered all rows of the server's result set.

The following example shows the CLOSE request if AUTOCLOSE is not used:

```
*****************************************************************
* CLOSE CONNECTION TO DV SERVER *
*****************************************************************
9000-CLOSE.
MOVE MESSAGE-LENGTH TO DVCB-MESSAGE-LENGTH.
MOVE FCT-CLOSE TO DVCB-REQUEST-CODE.
CALL 'AVZCLIEN' USING DVCB SEND-AREA RECEIVE-AREA
ERROR-AREA SQLDA-AREA.
PERFORM 5000-CHECK-RESPONSE.
```

Multiple SQL commands

Assume that an application needs to select some clients and read the orders for these clients.

Technically, the program must RECEIVE all these clients. For each client in the RECEIVE-AREA1 it needs to issue another SQL command statement in the order database and to process these orders as they come into the RECEIVE-AREA2.

Note that the API of this Data Virtualization Manager client uses a stateful protocol, and that you cannot send another SQL statement on the same open connection until you have completed receiving the result set from the current SQL statement.

However, one can have as many open connections as needed. The application must manage two connections, each with its own DVCB: one for clients, another for orders.

Processing logic is then as follows:

1. Obtain two DVCBs: CLIENTS-DVCB and ORDERS-DVCB
2. Call OPEN twice (once for CLIENTS-DVCB and once for ORDERS-DVCB)
3. Call SEND for the CLIENTS fetch, using CLIENTS-DVCB
4. Call RECV, pointing to CLIENTS-DVCB
5. Scan the clients in the CLIENTS buffer and for each client:
    a. Call SEND for the ORDERS fetch, using ORDERS-DVCB
    b. Call RECV, using ORDERS-DVCB
    c. Process the orders returned in the ORDERS receive buffer
    d. Loop to 5.b until EndOfData is returned in the ORDERS-DVCB
    e. Locate the next client in the CLIENTS buffer
    f. Return to step 5a.
6. When the CLIENTS buffer is exhausted, return to step 4
7. When EndOfData is returned in the CLIENTS-DVCB, call CLOS twice (once for CLIENTS-DVCB and once for ORDERS-DVCB)

Example: DS Client control block (DVCB)

```
000100*************************************************************************00010000
000200* *00020000
000300* COPYRIGHT ROCKET SOFTWARE, INC. 1991, 2016. *00030001
000400* *00040000
000500*************************************************************************00050000
000600 01 DVCB. 00060000
000700 03 DVCB-TAG PIC X(4) VALUE 'DVCB'. 00070000
000800 03 DVCB-VERSION PIC X(2) VALUE X'0001'. 00080000
000900 03 DVCB-RESERVED1 PIC X(2). 00090000
001000 03 DVCB-SSID PIC X(4). 00100000
001100 03 DVCB-REQUEST-CODE PIC X(4). 00110000
001200 03 DVCB-CNID. 00120000
001300 05 DVCB-CONNECTION PIC X(12). 00130000
001400 05 DVCB-CONNECTED-SSID PIC X(4). 00140000
001500 03 DVCB-SERVER-GROUP PIC X(8). 00150000
001600 03 DVCB-USER-PARM PIC X(8). 00160000
001700 03 DVCB-SQL-CODE PIC S9(5) COMP. 00170000
001800 03 DVCB-DATA-BUFFER-LENGTH PIC S9(5) COMP. 00180000
001900 03 DVCB-DATA-RETURNED-LENGTH PIC S9(5) COMP. 00190000
002000 03 DVCB-RESERVED2 PIC S9(5) COMP. 00200000
002100 03 DVCB-ROWS-RETURNED PIC S9(5) COMP. 00210000
002200 03 DVCB-OPTIONS. 00220000
002300 05 DVCB-OPT-RECV-MODE PIC X(1). 00230000
002400 05 DVCB-OPT-AUTO-COMMIT PIC X(1). 00240000
002500 05 DVCB-OPT-CLOSE-AFTER PIC X(1). 00250000
002600 05 DVCB-OPT-REFORMAT PIC X(1). 00260000
002700 05 DVCB-OPT-SQLDA PIC X(1). 00270000
002800 05 DVCB-OPT-RESERVED PIC X(3). 00280000
002900 03 DVCB-BLOCKING-TIMEOUT PIC S9(5) COMP. 00290000
003000 03 DVCB-SEND-LENGTH PIC S9(5) COMP. 00300000
003100 03 DVCB-RETURN-CODE PIC S9(5) COMP. 00310000
003200 03 DVCB-DB2-SUBSYSTEM PIC X(4). 00320000
003300 03 DVCB-ROW-LENGTH PIC S9(5) COMP. 00330000
003400 03 DVCB-SQLDA-LENGTH PIC S9(5) COMP. 00340000
003500 03 DVCB-MESSAGE-LENGTH PIC S9(5) COMP. 00350000
003600 03 DVCB-MAP-NAME PIC X(50). 00360000
003700 03 DVCB-RETURN-FLAGS. 00370000
003800 05 DVCB-ROW-RETURNED PIC X(1). 00380000
003900 05 DVCB-SQLCODE-RETURNED PIC X(1). 00390000
004000 05 DVCB-MESSAGE-RETURNED PIC X(1). 00400000
004100 05 DVCB-SQLDA-RETURNED PIC X(1). 00410000
004200 05 DVCB-END-OF-DATA PIC X(1). 00420000
004300 05 DVCB-ERROR-RETURNED PIC X(1). 00430000
004400 05 DVCB-PARMS-RETURNED PIC X(1). 00440000
004500 05 DVCB-END-OF-RSET PIC X(1). 00450000
004600 03 DVCB-RESERVED3 PIC X(2). 00460000
004700 03 DVCB-ROW-LIMIT PIC S9(5) COMP. 00470000
004800 03 DVCB-USERID PIC X(8). 00480000
004900 03 DVCB-PASSWORD PIC X(8). 00490000
005000 03 DVCB-MAPREDUCE-ID PIC X(2). 00500000
005100 03 DVCB-MAPREDUCE-NO PIC X(2). 00510000
005200 03 DVCB-RESERVED4 PIC X(64). 00520000
005300 03 DVCB-TAG2 PIC X(4) VALUE 'DVCB'. 00530000
005400 01 FCT-SEND PIC X(4) VALUE 'SEND'. 00540000
005500 01 FCT-RECEIVE PIC X(4) VALUE 'RECV'. 00550000
005600 01 FCT-OPEN PIC X(4) VALUE 'OPEN'. 00560000
005700 01 FCT-CLOSE PIC X(4) VALUE 'CLOS'. 00570000
005800 01 OPT-AUTOCLOSE PIC X VALUE 'Y'. 00580000
005900 01 OPT-LOCAL-MODE PIC X VALUE 'L'. 00590000
006000 01 OPT-MOVE-MODE PIC X VALUE 'M'. 00600000
```

Other useful variables

```
003300 01 SEND-LENGTH PIC S9(5) COMP VALUE 400. 00330000
003400 01 RECEIVE-LENGTH PIC S9(5) COMP VALUE 200. 00340013
003500 01 MESSAGE-LENGTH PIC S9(5) COMP VALUE 40. 00350000
003600 01 SQLDA-LENGTH PIC S9(5) COMP VALUE 1000. 00360000
003700 01 ERROR-AREA. 00370000
003800 02 ERROR-AREA1 PIC X(40) VALUE IS SPACES. 00380000
003900 01 SEND-AREA. 00390000
004000 02 SQL-TEXT PIC X(400) VALUE IS SPACES. 00400000
004100 01 SQLDA-AREA. 00410000
004200 02 SQLDAID PIC X(8). 00420000
004300 02 SQLDABC PIC S9(5) COMP. 00430000
004400 02 SQLN PIC S9(4) COMP. 00440000
004500 02 SQLD PIC S9(4) COMP. 00450000
```

```
004600 02 SQLVAR OCCURS 100. 00460000
004700 05 SQLTYPE PIC S9(4) COMP. 00470000
004800 05 SQLLEN PIC S9(4) COMP. 00480000
004900 05 SQLDATA PIC S9(5) COMP. 00490000
005000 05 SQLIND PIC S9(5) COMP. 00500000
005100 05 SQLNAME-LENGTH PIC S9(4) COMP. 00510000
005200 05 SQLNAME PIC X(30). 00520000
005300 01 RECEIVE-AREA. 00530000
005400 02 RECORD-SET PIC X(200). 00540013
005500 02 RECORD-OUT REDEFINES RECORD-SET OCCURS 10 TIMES. 00550013
005600 05 DVC-ID PIC S9(4) COMP. 00560000
005700 05 DVC-NAME PIC X(9). 00570000
005800 05 DVC-DEPT PIC S9(4) COMP. 00580000
005900 05 DVC-JOB PIC X(5). 00590000
006000 05 DVC-YEARS PIC S9(4) COMP. 00600000
```

Typical error checking

```
014400*****************************************************************01540000
014500 5000-CHECK-RESPONSE. 01550000
014600*****************************************************************01560000
014700 EVALUATE TRUE 01570000
014800 WHEN DVCB-ROW-RETURNED = 'Y' 01580000
014900 DISPLAY 'ROW RETURNED: TRUE' 01590000
015000 WHEN DVCB-SQLCODE-RETURNED = 'Y' 01600000
015100 DISPLAY 'SQLCODE RETURNED: TRUE' 01610000
015200 DISPLAY 'SQLCODE:' DVCB-SQL-CODE 01620000
015300 WHEN DVCB-MESSAGE-RETURNED = 'Y' 01630000
015400 DISPLAY 'MESSAGE RETURNED: TRUE' 01640000
015500 DISPLAY 'MESSAGE:' ERROR-AREA 01650000
015600 WHEN DVCB-SQLDA-RETURNED = 'Y' 01660000
015700 DISPLAY 'SQLDA RETURNED: TRUE' 01670000
015800 WHEN DVCB-END-OF-DATA = 'Y' 01680000
015900 DISPLAY 'END-OF-DATA RETURNED: TRUE' 01690000
016000 WHEN DVCB-ERROR-RETURNED = 'Y' 01700000
016100 DISPLAY 'ERROR RETURNED: TRUE' 01710000
016200 PERFORM 6000-WRITE-ERROR 01720000
016300 WHEN DVCB-PARMS-RETURNED = 'Y' 01730000
016400 DISPLAY 'IN/OUT PARMS RETURNED: TRUE' 01740000
016500 WHEN DVCB-END-OF-RSET = 'Y' 01750000
016600 DISPLAY 'END OF RESULT-SET RETURNED: TRUE' 01760000
016700 WHEN OTHER 01770000
016800 DISPLAY 'NO RESPONSE FLAGS SET' 01780011
016900 END-EVALUATE. 01790000
017000*****************************************************************01800000
017100 6000-WRITE-ERROR. 01810000
017200*****************************************************************01820000
017300 EVALUATE DVCB-RETURN-CODE 01830000
017400 WHEN 20 01840000
017500 DISPLAY 'DV SERVER NOT AVAILABLE' 01850000
017600 WHEN 32 01860000
017700 DISPLAY 'CONNECTION HANDLE NOT FOUND ON THIS SERVER' 01870000
017800 WHEN 36 01880000
017900 DISPLAY 'REQUEST MADE WHILE CONNECTION IN WRONG STATE' 01890000
018000 WHEN 40 01900000
018100 DISPLAY 'ROW DOES NOT FIT INTO RECEIVE BUFFER' 01910000
018200 WHEN 44 01920000
018300 DISPLAY 'RECV REQUEST TIMEOUT WAITING FOR DATA' 01930000
018400 WHEN 60 01940000
018500 DISPLAY 'SQL STATEMENT SEND LENGTH NOT SET' 01950000
```

Index

SC27-9302-00