Progress DataDirect for JDBC for Microsoft Dynamics 365

User's Guide

Release 6.0.0

Copyright

Table of Contents

1

Welcome to the Progress DataDirect for JDBC for Microsoft Dynamics 365 Driver

The Progress® DataDirect® for JDBC™ for Microsoft Dynamics 365™ driver supports SQL read-write access for JDBC applications to Microsoft Dynamics 365 apps. To support SQL access to Dynamics 365 apps, the driver creates a relational map of the Dynamics 365 data model and translates SQL statements to Dynamics 365 requests. In addition, the driver employs a SQL engine component that provides support to SQL constructs unavailable in Microsoft Dynamics 365. This functionality offers a number of advantages, including support for reporting data and metadata in a form that JDBC applications are ready to use.

The documentation for the driver also includes the *Progress DataDirect for JDBC Drivers Reference*. The reference provides general reference information for all DataDirect drivers for JDBC, including content on troubleshooting, supported SQL escapes, and DataDirect tools.

For the complete documentation set, visit the Progress DataDirect Connectors Documentation Hub: https://docs.progress.com/category/datadirect-microsoft-dynamics-365.

For details, see the following topics:

What's new in this release?

Support and certification

Visit the following web pages for the latest support and certification information.

- Release Notes: https://www.progress.com/datadirect-connectors/whats-new#jdbc
- DataDirect Product Compatibility Guide: https://docs.progress.com/bundle/datadirect-product-compatibility/resource/datadirect-product-compatibility.pdf

Changes Since 6.0.0 GA

- **Enhancements:**
  - The ShowNameColumns on page 89 property has been added to the driver. This property allows you to expose name columns for Choice, Yes/No, and Lookup field types in Dynamics CRM apps.
  - The driver has been enhanced to comply with FIPS standards for data encryption. As part of this enhancement, the driver was tested with FIPS 140-3 enabled using a Red Hat OpenJDK 21 on a Red Hat Universal Base Image 9 instance. See FIPS (Federal Information Processing Standard) on page 57 for details.
  - The driver has been enhanced to support fetching data from all the companies to which the user has access. Earlier, it supported fetching data only from the default company of the user. You can configure this functionality using the new CrossCompany connection property. See CrossCompany on page 71 for more information.
  - The driver has been enhanced to allow you to configure whether requests to web services timeout when the service is unresponsive. You can configure this functionality using the new WSTimeout connection property. See WSTimeout on page 96 for more information.
- **Changed Behavior**
  - The connection property `SpyAttributes` has been updated to exclude the attribute `load=classname`, which was previously used to load the driver specified by the given class name. See SpyAttributes on page 90 for details.

Highlights of 6.0.0 Release

- Supports SQL read-write access to the following Microsoft Dynamics 365 CRM and ERP apps. (See also Supported SQL statements and extensions on page 99.)

CRM apps	ERP apps
Customer Service	Finance
Field Service	Human Resources

Marketing	Retail
Project Service Automation	Supply Chain Management
Sales	

- The driver supports JDBC core functions. For details, refer to "JDBC support" in the *Progress DataDirect for JDBC Drivers Reference*.
- The driver supports the core SQL-92 grammar.
- The driver supports Dynamics 365 data types through data type inference. See Data types on page 20 and getTypeInfo() on page 21 for details.
- The driver supports OAuth 2.0 authentication. See OAuth 2.0 authentication on page 47 for supported grant types and further details.
- The driver supports NTLM authentication. See NTLM authentication on page 56 for details.
- The driver supports the handling of large result sets with paging, and the FetchSize on page 73 connection property.
- The driver supports optimizing insert, update, and delete operations with the BatchUpdateChunkSize on page 67 connection property.
- The driver includes the DataDirect JDBC Driver Configuration Manager for quick configuration and testing of your driver. This Configuration Manager allows you to:
  - Generate and edit connection URLs
  - Test connect connection URLs
  - Execute SQL commands for testing
  - Fetch OAuth tokens and configure OAuth
  - Access connection property descriptions and the full product documentation

Requirements

The driver is compatible with JDBC 2.0, 3.0, and 4.0.

The driver requires a Java Virtual Machine (JVM) that is Java SE 8 or higher, including Oracle JDK, OpenJDK, and IBM SDK (Java) distributions.

Installing and setting up the driver

This section provides you with an overview of the steps required to install and set-up the driver. After completing this procedure, you will be able to begin accessing data with your application.

To begin accessing data with the driver:

1. Install the driver:

a) After downloading the product, unzip the installer files to a temporary directory.

b) From the installer directory, run the appropriate installer file to start the installer.

- **Windows**: `PROGRESS_DATADIRECT_JDBC_INSTALL.exe`
- **Non-Windows**: `PROGRESS_DATADIRECT_JDBC_INSTALL.jar`

c) Follow the prompts to complete installation.

The installer program supports multiple installation methods, including command-line and silent installations. For detailed instructions, refer to the *Progress DataDirect for JDBC Drivers Installation Guide*.

2. Set your system CLASSPATH to include the driver `.jar` file. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate JDBC drivers on your computer. The following examples demonstrate setting the CLASSPATH from a command line using the default installation directory.

   - **Windows Example**

     ```
     CLASSPATH=.;C:\Program Files\Progress\DataDirect\JDBC\lib\60\dynamics365.jar
     ```

   - **UNIX/LINUX Example**

     ```
     CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/dynamics365.jar
     ```

3. Configure your driver using one of the following methods:

   - **Connection URL**: You can begin using the driver immediately by passing a connection URL with your application or tool. The following examples show how to connect using either NTLM authentication or OAuth 2.0 refresh token grant authentication.

     NTLM

     ```
     jdbc:datadirect:dynamics365
     ServiceURL=http://myonpreminstance.sso3.dynamics365.net/api/data/v9.1;
     AuthenticationMethod=NTLM;NTLMDomain=sso3;User=jsmith;Password=secret;
     ```

     Note: See NTLM authentication on page 56 for details.

     OAuth 2.0 refresh token grant

     ```
     jdbc:datadirect:dynamics365:
     ServiceURL=https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/;
     AuthenticationMethod=OAuth2;ClientID=client_id;
     ClientSecret=client_secret;TokenURI=token_uri;
     RefreshToken=refresh_token;
     ```

     Note: See OAuth 2.0 authentication on page 47 for details on this and other supported grant types.

   - **Data sources**: The driver also supports connecting using JDBC data sources. A JDBC data source is a Java object, specifically a DataSource object, that defines connection information required for a JDBC driver to connect to the database. See Connecting using data sources for more information.

Note: For most connections, specifying the minimum required connection properties is sufficient to begin accessing data; however, you can provide values for optional properties to use additional supported features and improve performance.

4. Set the values for any optional properties that you want to configure. For additional information on optional features and functionality, see the following resources:

   - Connection URL Examples provides connection string examples that can be used to configure common functionality and features. You can modify and combine these examples to create a string that best suites your environment.
   - Connection property descriptions provides a complete list of supported properties by functionality.
   - Performance considerations describes connection properties that affect performance, along with recommended settings.

5. Connect to your service and begin accessing data with your applications, BI tools, database tools, and more.

   Important: An initial connection may take a few minutes, depending on network speeds and the amount of metadata the driver must retrieve from the service. Similar delays may occur depending on the CreateMap setting. For example, a delay may be incurred if CreateMap is set to `OnChange` and changes have been made to Dynamics 365 native data. For details, see CreateMap.

   To help you get started, the following resources guide you through accessing data with some common tools:

   - DataDirect Configuration Manager: The DataDirect Configuration Manager allows you to test connect, execute SQL statements, and practice using the JDBC API right out of the box.
   - Interactive SQL for JDBC (JDBCISQL): JDBCISQL supports a command line interface that allows you to connect to a data source, execute SQL statements and retrieve results for display on a terminal. This tool provides a method of quickly testing your driver in an environment that does not support GUIs.
   - Tableau: Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data.
   - DbVisualizer: DB Visualizer is a database tool that allows you to connect and execute SQL statements against your data.
   - Supported SQL statements and extensions: This section describes the syntax used for SQL statements supported by the driver. You can modify and use the provided examples for your application or tool.

This completes the deployment of the driver.

Driver and DataSource classes

The following are the `Driver` and `DataSource` classes used by the driver:

Driver class:

com.ddtek.jdbc.dynamics365.Dynamics365Driver

DataSource class:

com.ddtek.jdbcx.dynamics365.Dynamics365DataSource

Connection URL examples

After setting the CLASSPATH, the connection information needs to be passed in the form of a connection URL. This section provides examples of connection strings configured to use common features and functionality. You can modify and/or combine these examples to create a connection string for your environment.

Note:

- You can also use the DataDirect Configuration Manager tool to generate and test connection URLs. For more information, see "Generating connection URLs with the Configuration Manager."
- Connection property names are case-insensitive. For example, `Password` is the same as `password`.
- For connection properties that support string values, use the following escape sequence to specify values containing leading or trailing spaces and curly brackets: {*value*}. For example: `User={hello }` or `Password={{hello}}`.

See also

Generating connection URLs with the Configuration Manager on page 40

OAuth 2.0 access token flow

The access token flow passes the access token directly from the client to the Dynamics 365 service for authentication.

Note: As opposed to using a third-party application such as Postman, you can use the Progress DataDirect Dynamics 365 Configuration Manager to obtain an access token to support the access token flow. See Obtain access and refresh tokens using the Configuration Manager for details.

Note: Access tokens are temporary and must be replaced to maintain the session without interruption. The life of an access token is typically one hour.

The following string includes the properties used to connect with the OAuth 2.0 access token flow.

```
jdbc:datadirect:dynamics365:ServiceURL=serviceurl;AuthenticationMethod=method;
AccessToken=access_token;[property=value[;...]];
```

where:

serviceurl

specifies the base URL of the Dynamics 365 instance to which you want to issue requests. For example, `https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/`.

method

specifies the authentication method used to connect to the service. The default value is `OAuth2`. Since `OAuth2` is the default, this value does not have to be specified in a connection URL used for OAuth 2.0 implementations.

access_token

specifies the access token required to authenticate to Dynamics 365. This property allows you to set the access token manually.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

The following example connection string includes the properties for connecting with the OAuth 2.0 access token flow.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:dynamics365:
    ServiceURL=https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/;
    AuthenticationMethod=OAuth2;
    AccessToken=C3TQH9zjwek4CgJCU-4Mxb2DxLNfI2LB3a-dNfpWYx;");
```

See also

OAuth 2.0 authentication on page 47
Connection property descriptions on page 59

OAuth 2.0 client credentials grant

The authentication flow for the client credentials grant exchanges client credentials for the access token at the location specified by the TokenURI. Web-based login and consent are not required.

Note: For the client credentials grant, you must create an application user on your Dynamics 365 web instance. See Create an application user for the client credentials grant for details.

```
jdbc:datadirect:dynamics365:ServiceURL=serviceurl;AuthenticationMethod=method;
ClientID=client_id;ClientSecret=client_secret;TokenURI=POST token_uri;
Scope=scope;[property=value[;...]];
```

where:

serviceurl

specifies the base URL of the Dynamics 365 instance to which you want to issue requests. For example, `https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/`.

method

specifies the authentication method used to connect to the service. The default value is `OAuth2`. Since `OAuth2` is the default, this value does not have to be specified in a connection URL used for OAuth 2.0 implementations.

client_id

specifies the client ID key for your application when authenticating with OAuth 2.0.

client_secret

specifies the client secret for your application when authenticating with OAuth 2.0.

Important: The client secret is a confidential value used to authenticate the application to the server. To prevent unauthorized access, this value must be securely maintained.

token_uri

specifies the endpoint used to exchange authentication credentials for access tokens when OAuth 2.0 authentication is enabled. For the client credentials grant, the token URI must be prefaced by the POST command. For example:

```
TokenURI=POST https://login.microsoftonline.com/common/oauth2/v2.0/token
```

scope

specifies an OAuth scope or a space-separated list of OAuth scopes that limit the permissions granted by an access token. Scope is required when using the Microsoft Identity Platform (v2) to provision users and manage application access. The following example shows the scope for a Dynamics CRM instance.

```
Scope=https://mywebinstance.api.crm.dynamics.com/.default
```

Note: The `/.default` scope is embedded in every application. It refers to a static list of permissions configured on the application registration. Refer to Microsoft Identity Plaform documentation for further details.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

The following example connection string includes the properties for connecting with the OAuth 2.0 client credentials grant.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:dynamics365:
    ServiceURL=https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/;
    AuthenticationMethod=OAuth2;ClientID=29453d6f-6789-25gh-gd8g-44tk3c527831;
    ClientSecret=12a3=bCD/EfGh4Ijk+Lm5P67qR8s=//TuV+WXy1Zabcd;
    TokenURI=POST https://login.microsoftonline.com/common/oauth2/v2.0/token;
    Scope=https://mywebinstance.api.crm.dynamics.com/.default;");
```

See also

OAuth 2.0 authentication on page 47
Connection property descriptions on page 59

OAuth 2.0 refresh token grant

The refresh token grant is used to replace expired access tokens with active ones by exchanging the refresh token at the endpoint specified by the TokenURI property.

Note: As opposed to using a third-party application such as Postman, you can use the Progress DataDirect Dynamics 365 Configuration Manager to obtain a refresh token to support the refresh token grant. See Obtain access and refresh tokens using the Configuration Manager for details.

This string includes the properties used to connect with OAuth 2.0 authentication.

```
jdbc:datadirect:dynamics365:ServiceURL=serviceurl;AuthenticationMethod=method;
ClientID=client_id;ClientSecret=client_secret;TokenURI=token_uri;
RefreshToken=refresh_token;[property=value[;...]];
```

where:

serviceurl

specifies the base URL of the Dynamics 365 instance to which you want to issue requests. For example, `https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/`.

method

specifies the authentication method used to connect to the service. The default value is `OAuth2`. Since `OAuth2` is the default, this value does not have to be specified in a connection URL used for OAuth 2.0 implementations.

client_id

specifies the client ID key for your application when authenticating with OAuth 2.0.

client_secret

specifies the client secret for your application when authenticating with OAuth 2.0.

Important: The client secret is a confidential value used to authenticate the application to the server. To prevent unauthorized access, this value must be securely maintained.

token_uri

specifies the endpoint used to exchange authentication credentials for access tokens when OAuth 2.0 authentication is enabled.

refresh_token

specifies the refresh token used to either request a new access token or renew an expired access token.

Important: The refresh token is a confidential value used to authenticate to the server. To prevent unauthorized access, this value must be securely maintained.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

The following example connection string includes the properties required for connecting with the OAuth 2.0 refresh token grant.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:dynamics365:
    ServiceURL=https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/;
    AuthenticationMethod=oauth2;ClientID=29453d6f-6789-25gh-gd8g-44tk3c527831;
    ClientSecret=12a3=bCD/EfGh4Ijk+Lm5P67qR8s=//TuV+WXy1Zabcd;
    TokenURI=https://login.microsoftonline.com/common/oauth2/v2.0/token;
    RefreshToken=12a3=bCD/EfGh4Ijk+Lgd8g-44tk3c527831;");
```

See also

OAuth 2.0 authentication on page 47
Connection property descriptions on page 59

NTLM

This string includes the properties used to connect with NTLM authentication.

```
jdbc:datadirect:dynamics365:ServiceURL=serviceurl;AuthenticationMethod=method;
NTLMDomain=domain;User=username;Password=password;
[property=value[;...]];
```

where:

serviceurl

specifies the base URL of the Dynamics 365 instance to which you want to issue requests. For example, `http://myonpreminstance.sso3.dynamics365.net/api/data/v9.1/`.

method

specifies the authentication method used to connect to the service. AuthenticationMethod should be set to `NTLM` for NTLM authentication.

domain

specifies the NTLM domain used for NTLM authentication.

username

specifies the user name that is used to connect to the service.

password

specifies the password for the user connecting to the service.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

Note: The User and Password properties are not required to be stored in the connection string. They can also be sent separately by the application.

The following example connection string includes the properties required for connecting with NTLM authentication.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:dynamics365
    ServiceURL=http://myonpreminstance.sso3.dynamics365.net/api/data/v9.1;
    AuthenticationMethod=NTLM;NTLMDomain=sso3;User=jsmith;Password=secret;");
```

See also

NTLM authentication on page 56
Connection property descriptions on page 59

Proxy server

This string includes the properties you may need to connect through a proxy server with OAuth 2.0 refresh token grant authentication.

```
jdbc:datadirect:dynamics365:ServiceURL=serviceurl;ProxyHost=proxy_host;ProxyPassword=proxy_password;
ProxyPort=proxy_port;ProxyUser=proxy_user;AuthenticationMethod=method;
ClientID=client_id;ClientSecret=client_secret;TokenURI=token_uri;
RefreshToken=refresh_token;[property=value[;...]];
```

where:

serviceurl

specifies the base URL of the Dynamics 365 instance to which you want to issue requests. For example, `https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/`.

proxy_host

specifies the proxy server to use for the first connection.

proxy_password

specifies the password needed to connect to a proxy server for the first connection.

proxy_port

specifies the port number where the proxy server is listening for requests for the first connection. The default is `0`.

proxy_user

specifies the user name needed to connect to a proxy server for the first connection.

method

specifies the authentication method used to connect to the service. AuthenticationMethod should be set to `OAuth2` for OAuth 2.0 authentication.

client_id

specifies the client ID key for your application when authenticating with OAuth 2.0.

client_secret

specifies the client secret for your application when authenticating with OAuth 2.0.

Important: The client secret is a confidential value used to authenticate the application to the server. To prevent unauthorized access, this value must be securely maintained.

token_uri

specifies the endpoint used to exchange authentication credentials for access tokens when OAuth 2.0 authentication is enabled.

refresh_token

specifies the refresh token used to either request a new access token or renew an expired access token.

Important: The refresh token is a confidential value used to authenticate to the server. To prevent unauthorized access, this value must be securely maintained.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

The following example connection string includes the properties required for using a proxy server with OAuth 2.0 refresh token grant authentication.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:dynamics365:
    ServiceURL=https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/;
    ProxyHost=pserver;ProxyPassword=secret;ProxyPort=808;ProxyUser=jsmith;
    AuthenticationMethod=OAuth2;ClientID=29453d6f-6789-25gh-gd8g-44tk3c527831;
    ClientSecret=12a3=bCD/EfGh4Ijk+Lm5P67qR8s=//TuV+WXy1Zabcd;
    TokenURI=https://login.microsoftonline.com/common/oauth2/v2.0/token;
    RefreshToken=12a3=bCD/EfGh4Ijk+Lgd8g-44tk3c527831;");
```

See also

Connection property descriptions on page 59

Data types

The following table lists data types supported by the driver and how they are mapped to JDBC data types.

See "getTypeInfo()" for getTypeInfo() results of data types supported by the driver.

Table 1: Dynamics365 Data Types

Dynamics365 Data Type	JDBC Data Type
BINARY	VARBINARY
BOOLEAN	BOOLEAN
BYTE	SMALLINT
DATE	DATE
DATETIMEOFFSET	TIMESTAMP
DECIMAL	DECIMAL
DOUBLE	DOUBLE
ENUM	VARCHAR
GUID	CHAR

Dynamics365 Data Type	JDBC Data Type
INT16	SMALLINT
INT32	INTEGER
INT64	BIGINT
SBYTE	TINYINT
SINGLE	REAL
STRING	VARCHAR
TIMEOFDAY	TIME

getTypeInfo()

The DatabaseMetaData.getTypeInfo() method returns information about data types. The following table provides getTypeInfo() results for supported data types.

Table 2: getTypeInfo() Results

TYPE_NAME = BINARY AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = -3 (VARBINARY) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = X' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = BINARY MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 3 SQL_DATA_TYPE = 61 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = BOOLEAN AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 16 (BOOLEAN) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = BOOLEAN MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 1 SEARCHABLE = 3 SQL_DATA_TYPE = 16 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = BYTE AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 5 (SMALLINT) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = BYTE MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 5 SEARCHABLE = 3 SQL_DATA_TYPE = 5 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = DATE AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 91 (DATE) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = DATE ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = DATE MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 10 SEARCHABLE = 3 SQL_DATA_TYPE = 91 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = DATETIMEOFFSET AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = TIMESTAMP ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = DATETIMEOFFSET MAXIMUM_SCALE = 6	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 26 SEARCHABLE = 3 SQL_DATA_TYPE = 93 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = DECIMAL AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = DECIMAL MAXIMUM_SCALE = 38	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 38 SEARCHABLE = 3 SQL_DATA_TYPE = 3 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = DOUBLE AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 8 (DOUBLE) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = DOUBLE MAXIMUM_SCALE = 15	MINIMUM_SCALE = -324 NULLABLE = 1 NUM_PREC_RADIX = 2 PRECISION = 15 SEARCHABLE = 3 SQL_DATA_TYPE = 8 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE

TYPE_NAME = ENUM AUTO_INCREMENT = FALSE CASE_SENSITIVE = TRUE CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = ENUM MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 3 SQL_DATA_TYPE = 12 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = GUID AUTO_INCREMENT = FALSE CASE_SENSITIVE = TRUE CREATE_PARAMS = NULL DATA_TYPE = 1 (CHAR) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = GUID MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 36 SEARCHABLE = 3 SQL_DATA_TYPE = 1 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = INT16 AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 5 (SMALLINT) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = INT16 MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 5 SEARCHABLE = 3 SQL_DATA_TYPE = 5 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE

TYPE_NAME = INT32 AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 4 (INTEGER) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = INT32 MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 3 SQL_DATA_TYPE = 4 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = INT64 AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = -5 (BIGINT) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = INT64 MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 19 SEARCHABLE = 3 SQL_DATA_TYPE = 25 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = SBYTE AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = -6 (TINYINT) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = SBYTE MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 3 SEARCHABLE = 3 SQL_DATA_TYPE = -6 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE

TYPE_NAME = SINGLE AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 7 (REAL) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = SINGLE MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 2 PRECISION = 7 SEARCHABLE = 3 SQL_DATA_TYPE = 7 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = STRING AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = STRING MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 3 SQL_DATA_TYPE = 12 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = TIMEOFDAY AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 92 (TIME) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = TIME ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = TIMEOFDAY MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 8 SEARCHABLE = 3 SQL_DATA_TYPE = 92 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

SQL escape sequences

The driver supports the following SQL escape sequences.

- Date, Time, and Timestamp Escape Sequences
- Scalar Functions
- Outer Join Escape Sequences
- LIKE Escape Character Sequence for Wildcards

Refer to "SQL escape sequences" in the *Progress DataDirect for JDBC Drivers Reference* for information about SQL escape sequences.

Supported scalar functions

The driver supports the scalar functions in the following table. Note that your database system may not support all these functions. Refer to the documentation for your database system to find out which functions are supported by your database.

In addition, you can also determine the supported scalar functions by using DatabaseMetaData methods.

You can use scalar functions in SQL statements with the following syntax:

`{fn scalar-function}`

where:

scalar-function

is a scalar function supported by the drivers, as listed in the following table.

Example:

```
SELECT id, name FROM emp WHERE name LIKE {fn UCASE('Smith')}
```

Refer to "Scalar functions" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

Table 3: Supported Scalar Functions

String Functions	Numeric Functions	Timedate Functions	System Functions
ASCII	ABS	CURDATE	CURSESSIONID
BIT_LENGTH	ACOS	CURRENT_DATE	DATABASE
CHAR	ASIN	CURRENT_TIME	IDENTITY
CHAR_LENGTH	ATAN	CURRENT_TIMESTAMP	USER
CHARACTER_LENGTH	ATAN2	CURTIME	
CONCAT	BITAND	DATEDIFF	

String Functions	Numeric Functions	Timedate Functions	System Functions
DIFFERENCE	BITOR	DATE_ADD	
HEXTORAW	BITXOR	DATE_SUB	
INSERT	CEILING	DAY	
LCASE	COS	DAYNAME	
LEFT	COT	DAYOFMONTH	
LENGTH	DEGREES	DAYOFWEEK	
LOCATE	EXP	DAYOFYEAR	
LOCATE_2	FLOOR	EXTRACT	
LOWER	LOG	HOUR	
LTRIM	LOG10	MINUTE	
OCTET_LENGTH	MOD	MONTH	
RAWTOHEX	PI	MONTHNAME	
REPEAT	POWER	NOW	
REPLACE	RADIANS	QUARTER	
RIGHT	RAND	SECOND	
RTRIM	ROUND	SECONDS_SINCE_MIDNIGHT	
SOUNDEX	ROUNDMAGIC	TIMESTAMPADD	
SPACE	SIGN	TIMESTAMPDIFF	
SUBSTR	SIN	TO_CHAR	
SUBSTRING	SQRT	WEEK	
UCASE	TAN	YEAR	
UPPER	TRUNCATE		

DataDirect tools

Progress DataDirect for JDBC drivers install the set of tools described in this section. For detailed instructions on using these tools, refer to the corresponding topics in the *Progress DataDirect for JDBC Drivers Reference*.

- DataDirect Test allows you to test your JDBC driver and learn the JDBC API.
- DataDirect Connection Pool Manager allows you to pool connections when accessing databases. When your applications use connection pooling, connections are reused rather than created each time a connection is requested. Because establishing a connection is among the most costly operations an application may perform, using Connection Pool Manager to implement connection pooling can significantly improve performance.
- Statement Pool Monitor loads statements into and remove statements from the statement pool as well as generate information to help you troubleshoot statement pooling performance.
- DataDirect Spy logs detailed information about calls your driver makes that can be used for troubleshooting.

See also

Generating connection URLs with the Configuration Manager on page 40

Troubleshooting

The *Progress DataDirect for JDBC Drivers Reference* provides information on troubleshooting problems should they occur. Refer to the "Troubleshooting" section in the *Reference* for details.

Additional information

In addition to the content provided in this guide, the documentation set also contains detailed conceptual and reference information that applies to all the drivers. For more information in these topics, refer the *Progress DataDirect for JDBC Drivers Reference* or use the links below to view some common topics:

- "JDBC support" describes support for JDBC interfaces and methods for the Progress DataDirect for JDBC drivers.
- "JDBC extensions" describes the JDBC extensions provided by the com.ddtek.jdbc.extensions package.
- "SQL escape sequences for JDBC" provides an overview of SQL escape sequences for JDBC. In addition, it documents the scalar functions that you use in SQL statements.
- "Security best practices for JDBC applications" describes the security best practices you should employ when developing and deploying your application with the driver.

Contacting Technical Support

Progress DataDirect offers a variety of options to meet your support needs. Please visit our Web site for more details and for contact information:

https://www.progress.com/support

The Progress DataDirect Web site provides the latest support information through our global service network. The SupportLink program provides access to support contact details, tools, patches, and valuable information, including a list of FAQs for each product. In addition, you can search our Knowledgebase for technical bulletins and other information.

When you contact us for assistance, please provide the following information:

- Your number or the serial number that corresponds to the product for which you are seeking support, or a case number if you have been provided one for your issue. If you do not have a SupportLink contract, the SupportLink representative assisting you will connect you with our Sales team.
- Your name, phone number, email address, and organization. For a first-time call, you may be asked for full information, including location.
- The Progress DataDirect product and the version that you are using.
- The type and version of the operating system where you have installed your product.
- Any database, database version, third-party software, or other environment information required to understand the problem.
- A brief description of the problem, including, but not limited to, any error messages you have received, what steps you followed prior to the initial occurrence of the problem, any trace logs capturing the issue, and so on. Depending on the complexity of the problem, you may be asked to submit an example or reproducible application so that the issue can be re-created.
- A description of what you have attempted to resolve the issue. If you have researched your issue on Web search engines, our Knowledgebase, or have tested additional configurations, applications, or other vendor products, you will want to carefully note everything you have already attempted.
- A simple assessment of how the severity of the issue is impacting your organization.

2

Tutorials

The following sections guide you through using the driver to access your data with some common third-party applications. For information on installing your driver and setting the CLASSPATH, see "Installing and setting-up the driver."

Important: When using third-party tools such as Tableau, network speeds and the amount of metadata retrieved from the service can cause timeout and out-of-memory errors. You can work around these issues in the following ways.

1. Increase timeout settings in your third-party tool. For example, Tableau offers the following guidance for timeout errors caused by large results.

   https://kb.tableau.com/articles/issue/error-extract-timeout

   You may also need to increase the Java heap size allotted for your third-party tool. This will allow the tool to manage large metadata results sent by the service. This workaround not only avoids timeout and out-of-memory errors on initial connection, but may also avoid these errors on subsequent connections.

2. Establish your initial connection using the DataDirect JDBC Driver Configuration Manager as described in "Testing connections and queries." In addition to specifying required properties, set the SchemaMap connection property. SchemaMap specifies the location and name of the configuration file where the map of the Dynamics 365 schema is written. Once connected, you can use the connection string generated by the Configuration Manager to connect via your third-party tool. However, it should be noted that timeout and out-of-memory errors may occur on subsequent connections, depending on the CreateMap setting.

For details, see the following topics:

- Interactive SQL for JDBC (JDBCISQL)
- Tableau
- DbVisualizer

Interactive SQL for JDBC (JDBCISQL)

After you have installed your driver and defined it on the CLASSPATH, you can use the driver to access your data with Interactive SQL for JDBC (JDBCISQL). JDBCISQL supports a command line interface that allows you to connect to a data source, execute SQL statements and retrieve results for display on a terminal.

To execute commands with JDBCISQL:

1. Start the ISQL tool. Do one of the following:
   - On Windows, double-click the`jdbcisql.bat` file in the *`install_dir`*`\jdbcisql` folder. Or, from a command prompt, navigate to the *`install_dir`*`\jdbcisql` directory and run the `jdbcisql.bat` file.
   - On Linux and UNIX, change to the *`install_dir`*`\isql` directory and run `jdbcisql.sh`.

   The Interactive SQL prompt appears.
2. Type the driver name class; then, press **Enter**:

   ```
   com.ddtek.jdbc.dynamics365.Dynamics365Driver
   ```

3. Type `connect` followed by the connection URL for the driver; then, press **Enter**. For example:
   - **NTLM**

     ```
     connect jdbc:datadirect:dynamics365
     ServiceURL=http://myonpreminstance.sso3.dynamics365.net/api/data/v9.1;
     AuthenticationMethod=NTLM;NTLMDomain=sso3;User=jsmith;Password=secret;
     ```

     Note: See NTLM authentication on page 56 for details.

   - **OAuth 2.0 refresh token grant**

     ```
     connect jdbc:datadirect:dynamics365:
     ServiceURL=https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/;
     AuthenticationMethod=OAuth2;ClientID=client_id;
     ClientSecret=client_secret;TokenURI=token_uri;
     RefreshToken=refresh_token;
     ```

     Note: See OAuth 2.0 authentication on page 47 for details on this and other supported grant types.

   If successful, the tool will return the time required to connect.
4. At the `ISQL>` prompt, issue a SQL command to query or modify the data source; then, press **Enter**. For example:

   ```
   SELECT * FROM FEATURES;
   ```

   Note: SQL commands must be terminated by a semi-colon.

Note: In addition to SQL commands, JDBCISQL supports a set of proprietary commands. Type `Help` at the prompt for a list of supported commands and syntax.

The results of the command are displayed in the terminal.

5. After you are finished executing queries and commands, you can disconnect from the data source by typing the following; then, pressing **Enter**:

```
DISCONNECT;
```

6. Press any key to end the session.

Tableau

After you have installed your driver and defined it on the CLASSPATH, you can use the driver to access your data with Tableau. Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data. By using the driver with Tableau, you can improve performance when retrieving data while leveraging the driver's relational mapping tools.

To use the driver to access data with Tableau:

1. Navigate to the `\lib\`*`xx`* subdirectory of the Progress DataDirect installation directory; then, locate the `jar` file for your driver:

```
dynamics365.jar
```

2. Copy the `.jar` file for your driver into the following directory:

   Windows: `C:\Program Files\Tableau\Drivers`

   Linux: `/opt/tableau/tableau_driver/jdbc`

3. Open Tableau. From the **Connect** menu, select **Other Databases (JDBC)**.
4. In the **Other Databases (JDBC)** dialog, provide values for the following fields; then, click **Sign In**.

   - **URL**: Copy and paste your connection URL into this field. The following examples show how to connect using either NTLM authentication or OAuth 2.0 refresh token grant authentication.

     NTLM

```
jdbc:datadirect:dynamics365
ServiceURL=http://myonpreminstance.sso3.dynamics365.net/api/data/v9.1;
AuthenticationMethod=NTLM;NTLMDomain=sso3;User=jsmith;Password=secret;
```

     Note: See NTLM authentication on page 56 for details.

     OAuth 2.0 refresh token grant

```
jdbc:datadirect:dynamics365:
ServiceURL=https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/;
AuthenticationMethod=OAuth2;ClientID=client_id;
ClientSecret=client_secret;TokenURI=token_uri;
RefreshToken=refresh_token;
```

Note: See OAuth 2.0 authentication on page 47 for details on this and other supported grant types.

- **Dialect**: Select **SQL92** (the default) from the drop-down box.
- **Username**: If required by the authentication method being used, enter the user name. Alternatively, this value can be specified with the `User` property in the connection string.
- **Password**: If required by the authentication method being used, enter the password. Alternatively, this value can be specified with the `Password` property in the connection string.

5. The **Data Source** window appears. In the **Schema** field, select the schema for the service you want to use.
6. In the **Table** field, the tables stored in the selected schema are now exposed and available for selection.

You have successfully accessed your data and are now ready to create reports with Tableau. For detailed information, refer to the Tableau product documentation at: https://www.tableau.com/support/help.

DbVisualizer

After you have installed your driver and defined it on the CLASSPATH, you can use the driver to access your data with the third-party DbVisualizer tool. The following topics guide you through using DbVisualizer to add your driver, connect, and execute SQL statements.

Adding a driver

To add a driver with DbVisualizer:

1. Open DbVisualizer.
2. From the menu, select **Tools**>**Driver Manager**. The Driver Manager window opens.
3. From the Driver Manager menu, select **Driver**>**Create Driver**.
4. Click the  button to navigate to the location of the driver jar file; then, click **OK**. The following are the default locations for the driver:

   Windows

   ```
   C:\Program Files\Progress\DataDirect\JDBC\lib\60\dynamics365.jar
   ```

   Linux

   ```
   /opt/Progress/DataDirect/JDBC/lib/60/dynamics365.jar
   ```

5. Provide values for the following fields; then, close the Driver Manager window.
   - **Name**: Type an alias for your driver. For example:

     ```
     Dynamics365
     ```

   - **URL Format**: Optionally, specify the format of the connection string for your driver. For example:

     ```
     jdbc:datadirect:dynamics365:ServiceURL=<server>
     ```

- **Driver Class**: From the drop down menu, select the driver class for your driver. For example:

  ```
  com.ddtek.jdbcx.dynamics365.Dynamics365DataSource
  ```

You can now use your driver with DbVisualizer. Proceed to "Connecting and executing SQL statements" for information on connecting and executing SQL statements.

Connecting and executing SQL statements

To use the driver to access data with DbVisualizer:

1. Open DbVisualizer.
2. From the menu, select **Database**>**New Connection**. When prompted to use the Connection Wizard, click **OK**.
3. Provide the following information when prompted; then, click **Next** to proceed:
   - **Connection alias**: Type the name to be used when referring to this connection.
   - **Driver**: Select the alias that you provided for your driver from the drop-down menu.
4. Provide values for the following fields; then, click **Finish**.
   - **Database URL**: Copy and paste your connection URL into this field. The following examples show how to connect using either NTLM authentication or OAuth 2.0 refresh token grant authentication.

     NTLM

     ```
     jdbc:datadirect:dynamics365
     ServiceURL=http://myonpreminstance.sso3.dynamics365.net/api/data/v9.1;
     AuthenticationMethod=NTLM;NTLMDomain=sso3;User=jsmith;Password=secret;
     ```

     Note: See NTLM authentication on page 56 for details.

     OAuth 2.0 refresh token grant

     ```
     jdbc:datadirect:dynamics365:
     ServiceURL=https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/;
     AuthenticationMethod=OAuth2;ClientID=client_id;
     ClientSecret=client_secret;TokenURI=token_uri;
     RefreshToken=refresh_token;
     ```

     Note: See OAuth 2.0 authentication on page 47 for details on this and other supported grant types.

   - **Database Userid**: If required by the authentication method being used, enter the user name. Alternatively, this value can be specified with the `User` property in the connection string.
   - **Database Password**: If required by the authentication method being used, enter the password. Alternatively, this value can be specified with the `Password` property in the connection string.
5. To execute SQL statements, select **SQL Commander**>**New SQL Commander**. A SQL Commander tab opens.
6. Select values for the following fields:

- **Database Connection**: Select connection alias you provided for the connection from the drop-down menu.
- **Schema**: Select the schema you want to execute queries against from the drop-down menu.

7. In the SQL Commander tab, enter SQL commands you want to execute; then select **SQL Commander**>**Execute**. For example:

   To select all of the rows from the BOARDS table:

   ```
   SELECT * FROM BOARDS
   ```

   To select the URLs for a specified issue :

   ```
   SELECT SELF FROM ISSUES WHERE ID = <issue_number>
   ```

   See "Supported SQL statements and extensions" for the supported syntax used to execute SQL statements.

   Note: If you are fetching large sets of data, you may want to limit the results using the Max Rows and Max Chars fields.

You have successfully accessed your data with DbVisualizer.

3

Configuring and connecting

This section provides information on how to connect to your data store using either the JDBC Driver Manager or DataDirect JDBC data sources, as well as information on how to implement and use functionality supported by the driver.

After the driver has been installed and defined on your classpath, you can connect from your application to your data in either of the following ways.

- Using the JDBC `DriverManager` by specifying the connection URL in the `DriverManager.getConnection()` method.
- Creating a JDBC data source that can be accessed through the Java Naming Directory Interface (JNDI).

For details, see the following topics:

Setting the classpath

The driver must be defined on your CLASSPATH before you can connect. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate JDBC drivers on your computer. If the driver is not defined on your CLASSPATH, you will receive a `class not found` exception when trying to load the driver. Set your system CLASSPATH to include the driver jar file as shown, where *`install_dir`* is the path to your product installation directory.

`install_dir``/lib/60/dynamics365.jar`

Windows Example

```
CLASSPATH=.;C:\Program Files\Progress\DataDirect\JDBC\lib\60\dynamics365.jar
```

UNIX Example

```
CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/dynamics365.jar
```

Connecting using the JDBC Driver Manager

One way to connect to a service is through the JDBC DriverManager using the `DriverManager.getConnection()` method. As the following examples show, this method specifies a string containing a connection URL.

NTLM authentication

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:dynamics365:
    ServiceURL=http://mycompany.sso3.dynamics365.net/api/data/;
    AuthenticationMethod=NTLM;NTLMDomain=sso3;User=jsmith;Password=secret;");
```

Note: See NTLM authentication on page 56 for details.

OAuth 2.0 refresh token grant

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:dynamics365:
    ServiceURL=https://mycompany.365.dynamics.net/api/data/;
    AuthenticationMethod=OAuth2;ClientID=29453d6f-6789-25gh-gd8g-44tk3c527831;
    ClientSecret=12a3=bCD/EfGh4Ijk+Lm5P67qR8s=//TuV+WXy1Zabcd;
    TokenURI=https://login.microsoftonline.com/ab123c45-def6-7g89-gh1i-m2345
       no67891/oauth2/token;
    RefreshToken=12a3=bCD/EfGh4Ijk+Lgd8g-44tk3c527831;");
```

Note: See OAuth 2.0 authentication on page 47 for details on this and other supported grant types.

Passing the connection URL

After setting the CLASSPATH, the required connection information needs to be passed in the form of a connection URL. The following example includes the properties required for connecting with OAuth 2.0 refresh token grant authentication.

Connection URL Syntax

The connection URL takes the following form:

```
jdbc:datadirect:dynamics365:ServiceURL=serviceurl;AuthenticationMethod=method;
ClientID=client_id;ClientSecret=client_secret;TokenURI=token_uri;
RefreshToken=refresh_token;[property=value[;...]];
```

where:

serviceurl

specifies the base URL of the Dynamics 365 instance to which you want to issue requests. For example, `https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/`.

method

specifies the authentication method used to connect to the service. AuthenticationMethod should be set to `OAuth2` for OAuth 2.0 authentication.

client_id

specifies the client ID key for your application when authenticating with OAuth 2.0.

client_secret

specifies the client secret for your application when authenticating with OAuth 2.0.

Important: The client secret is a confidential value used to authenticate the application to the server. To prevent unauthorized access, this value must be securely maintained.

uri_token

specifies the endpoint used to exchange authentication credentials for access tokens when OAuth 2.0 authentication is enabled.

refresh_token

specifies the refresh token used to either request a new access token or renew an expired access token.

Important: The refresh token is a confidential value used to authenticate to the server. To prevent unauthorized access, this value must be securely maintained.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

The following example connection string includes the properties required for connecting with the OAuth 2.0 refresh token grant.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:dynamics365:
    ServiceURL=https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/;
    AuthenticationMethod=OAuth2;ClientID=29453d6f-6789-25gh-gd8g-44tk3c527831;
    ClientSecret=12a3=bCD/EfGh4Ijk+Lm5P67qR8s=//TuV+WXy1Zabcd;
    TokenURI=https://login.microsoftonline.com/common/oauth2/v2.0/token;
    RefreshToken=12a3=bCD/EfGh4Ijk+Lgd8g-44tk3c527831;");
```

See also

Connection URL examples on page 14
OAuth 2.0 authentication on page 47
NTLM authentication on page 56
Connection property descriptions on page 59

Generating connection URLs with the Configuration Manager

The driver includes a browser-based tool, Progress DataDirect Dynamics 365 Configuration Manager, that allows you to generate connection URLs, test connections, and execute test queries. This section will guide you through generating and testing a connection URL that can be used by your application.

To generate a connection URL:

1. Open the Dynamics 365 Configuration Manager by double-clicking the driver jar file. Or, in a command line, navigate to the directory containing your driver jar file; then, execute the following command:

   ```
   java -jar dynamics365.jar
   ```

   The Dynamics 365 Configuration Manager opens in your default web browser.

2. From the browser window, provide values of the connection properties you want to configure in the corresponding fields. A connection URL will generate in the Connection String field as you provide settings. To view more properties, select the tabs at the top of the page. See "Connection URL examples" for a list of required properties and properties used for different configurations.

   Note: If you do not specify a value for an optional property, the property will be omitted from the string and the default value will be used.

3. Optionally, you can manually edit your string by clicking the Edit button ().
4. At any point during the process, you can click **Test Connect** to attempt to connect to the service using the string generated in the Connection String field. In the **Test Connection** window:

   Important: An initial connection may take a few minutes, depending on network speeds and the amount of metadata the driver must retrieve from the service. Similar delays may occur depending on the CreateMap setting. For example, a delay may be incurred if CreateMap is set to `OnChange` and changes have been made to Dynamics 365 native data. For details, see CreateMap.

a) Provide values for any fields required by your service.

b) Optionally, in the Test Query field, enter any SQL queries you want to execute during the test. For example:

```
SELECT * FROM ACCOUNTS
```

c) Click **Execute**.

If successful, the window displays a confirmation message and, if a query was specified, the results of the query.

Note: The information you enter in the logon dialog box during a test connect is not saved.

5. To use your string, click the Copy button () and paste the string to a location that can be used by your application.
6. Optionally, click **Save** to store your connection string for later use.

See also

Obtain access and refresh tokens using the Configuration Manager on page 54

Testing connections and queries

You can quickly test a connection string and queries using Progress DataDirect Dynamics 365 Configuration Manager.

To test your connection and query:

1. Open the Dynamics 365 Configuration Manager by double-clicking on the driver jar file. Or, in a command line, navigate to the directory containing your driver jar file; then, execute the following command:

   ```
   java -jar dynamics365.jar
   ```

   The Dynamics 365 Configuration Manager opens in your default web browser.

2. Provide a connection string to test using one of the following methods:

   - Entering a string: Click the Edit button (); then, paste your string into the Connection String field. If you prefer, you can also type a string directly into this field.
   - Generating a string: Provide values of the connection properties you want to configure into the corresponding fields. The Dynamics 365 Configuration Manager will generate a string in the Connection String field based on the values you specify.

3. Click **Test Connect** to attempt to connect to the service using the string specified in the Connection String field. The **Test Connection** window appears.

Important: An initial connection may take a few minutes, depending on network speeds and the amount of metadata the driver must retrieve from the service. Similar delays may occur depending on the CreateMap setting. For example, a delay may be incurred if CreateMap is set to `OnChange` and changes have been made to Dynamics 365 native data. For details, see CreateMap.

4. Provide connection property values for any fields required by your service.
5. To execute a test query with the test connection, enter a SQL query into the Test Query field. For example:

   ```
   SELECT * FROM ACCOUNTS
   ```

6. Click **Execute**.

   If successful, the window displays a confirmation message and, if a query was specified, the results of the query.

Connecting using data sources

A *JDBC data source* is a Java object, specifically a `DataSource` object, that defines connection information required for a JDBC driver to connect to the database. Each JDBC driver vendor provides their own data source implementation for this purpose. A Progress DataDirect data source is Progress DataDirect's implementation of a `DataSource` object that provides the connection information needed for the driver to connect to a database.

Because data sources work with the Java Naming Directory Interface (JNDI) naming service, data sources can be created and managed separately from the applications that use them. Because the connection information is defined outside of the application, the effort to reconfigure your infrastructure when a change is made is minimized. For example, if the database is moved to another database server, the administrator need only change the relevant properties of the `DataSource` object. The applications using the database do not need to change because they only refer to the name of the data source.

How data sources are implemented

Data sources are implemented through a `DataSource` class. A data source class implements the following interfaces.

- `javax.sql.DataSource`
- `javax.sql.ConnectionPoolDataSource` (allows applications to use connection pooling)

Refer to "Connection Pool Manager" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

See also

Driver and DataSource classes on page 13

Creating data sources

The following example files provide details on creating and using Progress DataDirect data sources with the Java Naming Directory Interface (JNDI), where *`install_dir`* is the product installation directory.

- *`install_dir`*`/Examples/JNDI/JNDI_LDAP_Example.java` can be used to create a JDBC data source and save it in your LDAP directory using the JNDI Provider for LDAP.
- *`install_dir`*`/Examples/JNDI/JNDI_FILESYSTEM_Example.java` can be used to create a JDBC data source and save it in your local file system using the File System JNDI Provider.

See "Example data source" for an example data source definition for the example files.

To connect using a JNDI data source, the driver needs to access a JNDI data store to persist the data source information. For a JNDI file system implementation, you must download the File System Service Provider from the Oracle Technology Network Java SE Support downloads page, unzip the files to an appropriate location, and add the `fscontext.jar` and `providerutil.jar` files to your CLASSPATH. These steps are not required for LDAP implementations because the LDAP Service Provider is included with supported versions of Java SE.

Example data source

To configure a data source using the example files, you will need to create a data source definition. The content required to create a data source definition is divided into three sections.

First, you will need to import the data source class. For example:

```
import com.ddtek.jdbcx.dynamics365.Dynamics365DataSource;
```

Next, you will need to set the values and define the data source. For example, the following definition contains the minimum properties required for a connection using the OAuth 2.0 refresh token grant.

Note:

- Setting the password using a data source is generally not recommended. The data source persists all properties, including the Password property, in clear text.
- In a JDBC data source, string values must be enclosed in double quotation marks, for example, `setUser("abc@defcorp.com")`.

```
Dynamics365DataSource mds = new Dynamics365DataSource();
mds.setDescription("My Dynamics 365 Data Source");
mds.setServiceURL("https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/");
mds.AuthenticationMethod("OAuth2");
mds.ClientID("29453d6f-6789-25gh-gd8g-44tk3c527831");
mds.ClientSecret("12a3=bCD/EfGh4Ijk+Lm5P67qR8s=//TuV+WXy1Zabcd");
mds.TokenURI("https://login.microsoftonline.com/common/oauth2/v2.0/token");
mds.RefreshToken("12a3=bCD/EfGh4Ijk+Lgd8g-44tk3c527831");
```

Finally, you will need to configure the example application to print out the data source attributes. Note that this code is specific to the driver and should only be used in the example application. For example, you would add the following section for the minimum properties required to establish a connection:

```
if (ds instanceof Dynamics365DataSource)
{
Dynamics365DataSource jmds = (Dynamics365DataSource) ds;
System.out.println("description=" + jmds.getDescription());
System.out.println("serviceurl=" + jmds.getServiceURL());
...
System.out.println();
}
```

Calling a data source in an application

Applications can call a Progress DataDirect data source using a logical name to retrieve the `javax.sql.DataSource` object. This object loads the specified driver and can be used to establish a connection to the database.

Once the data source has been registered with JNDI, it can be used by your JDBC application as shown in the following code example.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("EmployeeDB");
Connection con = ds.getConnection("domino", "spark");
```

In this example, the JNDI environment is first initialized. Next, the initial naming context is used to find the logical name of the data source (`EmployeeDB`). The `Context.lookup()` method returns a reference to a Java object, which is narrowed to a `javax.sql.DataSource` object. Then, the `DataSource.getConnection()` method is called to establish a connection.

Testing a data source connection

You can use DataDirect Test™ to establish and test a data source connection. The screen shots in this section were taken on a Windows system.

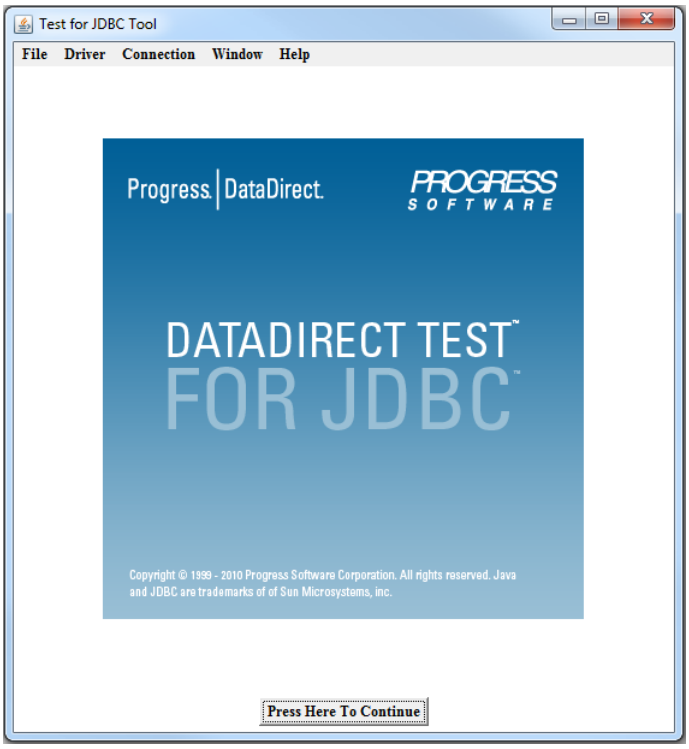
Take the following steps to establish a connection.

1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`
   - UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC/testforjdbc`

   Note: For UNIX/Linux, if you do not have access to `/opt`, your home directory will be used in its place.

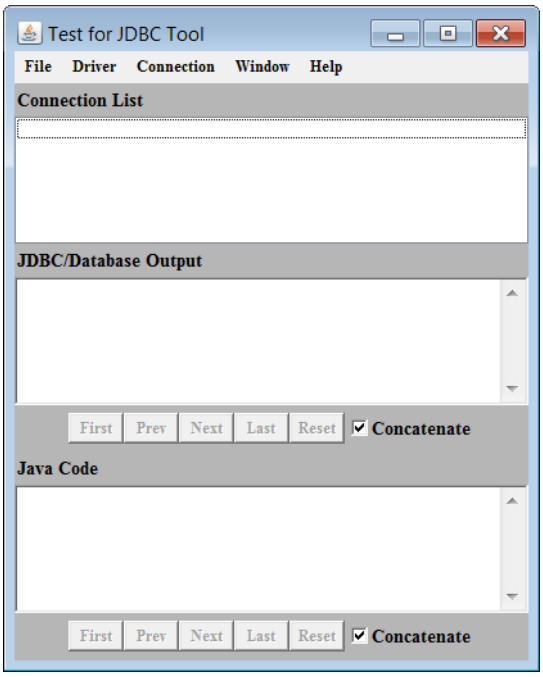
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

   The **Test for JDBC Tool** window appears:



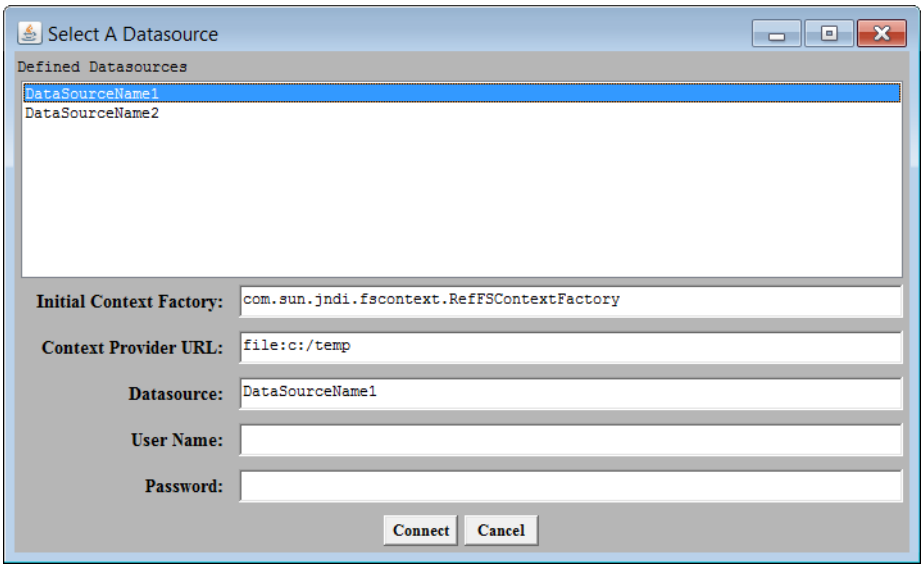
3. Click **Press Here to Continue**.

   The main dialog appears:



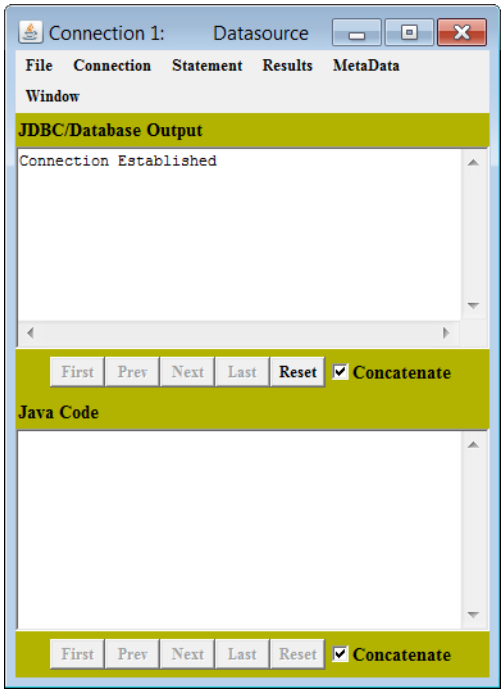
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   The **Select A Database** dialog appears:



5. Select a datasource template from the **Defined Datasources** field.
6. Provide the following information:
   a) In the **Initial Context Factory**, specify the location of the initial context provider for your application.
   b) In the **Context Provider URL**, specify the location of the context provider for your application.
   c) In the **Datasource** field, specify the name of your datasource.
7. If you are using user ID/password authentication, enter your user ID and password in the corresponding fields.
8. Click **Connect**.

If the connection information is entered correctly, the **JDBC/Database Output** window reports that a connection has been established. If a connection is not established, the window reports an error.



OAuth 2.0 authentication

The driver supports the following OAuth 2.0 grant types for accessing web instances of Dynamics 365 services.

- Access token flow on page 47
- Client credentials grant on page 48
- Refresh token grant on page 49

If it is not already registered, you will need to register your application as an Azure Active Directory application to implement an OAuth 2.0 grant flow and run an application against a Dynamics 365 web instance. This process is detailed in Register your application on the Azure portal.

To configure the driver for an OAuth 2.0 implementation, you will need client credentials and endpoints related to your registered application, as well as the service URL for your Dynamics 365 web instance. In addition, for the client credentials grant, you must create an application user on your Dynamics 365 web instance. Consult the following topics for details.

- Obtain the service URL
- Obtain application client information and endpoints
- Determine the scope (only for Microsoft Identity Platform v2)
- Obtain access and refresh tokens using the Configuration Manager
- Create an application user for the client credentials grant

Access token flow

The access token flow passes the access token directly from the client to the Dynamics 365 service for authentication.

Note: As opposed to using a third-party application such as Postman, you can use the Progress DataDirect Dynamics 365 Configuration Manager to obtain an access token to support the access token flow. See Obtain access and refresh tokens using the Configuration Manager for details.

Note: Access tokens are temporary and must be replaced to maintain the session without interruption. The life of an access token is typically one hour.

To configure the driver to use an access token flow:

- Set the ServiceURL property to the base URL of the Dynamics 365 instance to which you want to issue requests. For example, `https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/`.
- Set the AuthenticationMethod property to `OAuth2`. Since `OAuth2` is the default, this value does not have to be specified in a connection URL used for OAuth 2.0 implementations.
- Set the AccessToken property to the value of the access token obtained from external sources.

The following examples show the connection information required to establish a session using the access token flow.

Connection URL

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:dynamics365:
    ServiceURL=https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/;
    AuthenticationMethod=OAuth2;
    AccessToken=C3TQH9zjwek4CgJCU-4Mxb2DxLNfI2LB3a-dNfpWYx;");
```

Data Source

```
Dynamics365DataSource mds = new Dynamics365DataSource();
mds.setDescription("My Dynamics 365 Data Source");
mds.setServiceURL("https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/");
mds.setAuthenticationMethod("OAuth2");
mds.setAccessToken("C3TQH9zjweek4CgJCU-4Mxb2DxPLNfI2LB3a-dNfpWnYx");
```

See also

OAuth 2.0 authentication on page 47
Connection property descriptions on page 59

Client credentials grant

The authentication flow for the client credentials grant exchanges client credentials for the access token at the location specified by the TokenURI. Web-based login and consent are not required.

Note: For the client credentials grant, you must create an application user on your Dynamics 365 web instance. See Create an application user for the client credentials grant for details.

To configure the driver to use a client credentials grant:

- Set the ServiceURL property to the base URL of the Dynamics 365 instance to which you want to issue requests. For example, `https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/`.
- Set the AuthenticationMethod property to `OAuth2`. Since `OAuth2` is the default, this value does not have to be specified in a connection URL used for OAuth 2.0 implementations.
- Set the ClientID property to specify the client ID key for your application.
- Set the ClientSecret property to specify client secret for your application.

  Important: The client secret is a confidential value used to authenticate the application to the server. To prevent unauthorized access, this value must be securely maintained.

- Set the TokenURI property. The value of the TokenURI property must begin with the `POST` command followed by the token URI or the endpoint used to exchange authentication credentials for access tokens. For example:

  ```
  TokenURI=POST https://login.microsoftonline.com/common/oauth2/v2.0/token
  ```

- Set the Scope property when using the Microsoft Identity Platform (v2) to provision users and manage application access. Scope specifies an OAuth scope or a space-separated list of OAuth scopes that limit the permissions granted by an access token. The following example shows the scope for a Dynamics CRM instance.

  ```
  Scope=https://mywebinstance.api.crm.dynamics.com/.default
  ```

Note: The `/.default` scope is embedded in every application. It refers to a static list of permissions configured on the application registration. Refer to Microsoft Identity Platform documentation for further details.

The following examples show the connection information required to establish a session using the client credentials grant.

Connection URL

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:dynamics365:
    ServiceURL=https://mycompany.365.dynamics.net/api/data/;
    AuthenticationMethod=OAuth2;ClientID=29453d6f-6789-25gh-gd8g-44tk3c527831;
    ClientID=29453d6f-6789-25gh-gd8g-44tk3c527831;
    ClientSecret=12a3=bCD/EfGh4Ijk+Lm5P67qR8s=//TuV+WXy1Zabcd;
    TokenURI=POST https://login.microsoftonline.com/common/oauth2/v2.0/token;
    Scope=https://mywebinstance.api.crm.dynamics.com/.default;");
```

Data Source

```
Dynamics365DataSource mds = new Dynamics365DataSource();
mds.setDescription("My Dynamics 365 Data Source");
mds.setServiceURL("http://mycompany.365.dynamics.net/api/data/");
mds.AuthenticationMethod("OAuth2");
mds.ClientID("29453d6f-6789-25gh-gd8g-44tk3c527831");
mds.ClientSecret("12a3=bCD/EfGh4Ijk+Lm5P67qR8s=//TuV+WXy1Zabcd");
mds.TokenURI("POST https://login.microsoftonline.com/common/oauth2/v2.0/token");
mds.Scope("https://mywebinstance.api.crm.dynamics.com/.default");
```

See also

OAuth 2.0 authentication on page 47
Connection property descriptions on page 59

Refresh token grant

The refresh token grant is used to replace expired access tokens with active ones by exchanging the refresh token at the endpoint specified by the TokenURI property.

Note: As opposed to using a third-party application such as Postman, you can use the Progress DataDirect Dynamics 365 Configuration Manager to obtain a refresh token to support the refresh token grant. See Obtain access and refresh tokens using the Configuration Manager for details.

To configure the driver to use the refresh token grant:

- Set the ServiceURL property to the base URL of the Dynamics 365 instance to which you want to issue requests. For example, `https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/`.
- Set the AuthenticationMethod property to `OAuth2`. Since `OAuth2` is the default, this value does not have to be specified in a connection URL used for OAuth 2.0 implementations.
- Set the ClientID property to specify the client ID key for your application.
- Set the ClientSecret property to specify the client secret for your application.

Important: The client secret is a confidential value used to authenticate the application to the server. To prevent unauthorized access, this value must be securely maintained.

- Set the TokenURI property to specify the endpoint from which the driver fetches access tokens.
- Set the RefreshToken property to specify the refresh token used to request a new access token or renew an expired one.

Important: The refresh token is a confidential value used to authenticate to the server. To prevent unauthorized access, this value must be securely maintained.

The following examples show the connection information required to establish a session using the refresh token grant.

Connection URL

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:dynamics365:
    ServiceURL=https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/;
    AuthenticationMethod=OAuth2;ClientID=29453d6f-6789-25gh-gd8g-44tk3c527831;
    ClientSecret=12a3=bCD/EfGh4Ijk+Lm5P67qR8s=//TuV+WXy1Zabcd;
    TokenURI=https://login.microsoftonline.com/common/oauth2/v2.0/token;
    RefreshToken=12a3=bCD/EfGh4Ijk+Lgd8g-44tk3c527831;");
```

Data Source

```
Dynamics365DataSource mds = new Dynamics365DataSource();
mds.setDescription("My Dynamics 365 Data Source");
mds.setServiceURL("https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/");
mds.AuthenticationMethod("OAuth2");
mds.setClientID("29453d6f-6789-25gh-gd8g-44tk3c527831");
mds.setClientSecret("12a3=bCD/EfGh4Ijk+Lm5P67qR8s=//TuV+WXy1Zabcd");
mds.setTokenURI("https://login.microsoftonline.com/common/oauth2/v2.0/token");
mds.setRefreshToken("12a3=bCD/EfGh4Ijk+Lgd8g-44tk3c527831");
```

See also

OAuth 2.0 authentication on page 47
Connection property descriptions on page 59

Register your application on the Azure portal

Take the following steps to register your application as an Azure Active Directory application.

1. Go to the Azure portal.

   https://portal.azure.com

2. Select **Azure Active Directory**.
3. Select **App registrations** on the left.
4. Select **New registration**.
5. Enter an application name.
6. Select the account types your application supports.
7. Enter a **Redirect URI**.
   - **Option 1**. If you are using the Configuration Manager to obtain OAuth tokens, select **Web** from the dropdown and enter `http://localhost` as the redirect URI.

- **Option 2**. If you are using a 3rd party application to obtain OAuth tokens, select **Web** or **Public client/native (mobile & desktop)** as required for your application. Then, enter the redirect URI associated with the application.

8. Press **Register**.

   You are directed to the application overview page.

9. Add permissions associated with the Microsoft APIs your application uses.
   a) Select **API permissions** from the navigation menu.
   b) Select the API your application uses.
   c) As appropriate, select **Delegated permissions** or **Application permissions**.
   d) Check the box of each permission you want to grant the application.
   e) Press **Add permissions**.

Results:

You have successfully registered your application on Azure Active Directory, as well as added permissions required for application access to Microsoft APIs.

See also

OAuth 2.0 authentication on page 47

Obtain the service URL

The service URL is the REST endpoint of the Dynamics 365 service to which you are connecting. The service URL is specified with the ServiceURL property. For example:

```
ServiceURL=https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/
```

Note: The service URL contains the resource URI `https://mywebinstance.api.crm.dynamics.com`. Depending on whether you are using Azure Active Directory (v1) or Microsoft Identity Platform (v2) to manage user and application permissions, the resource URI is used in either the authorization URI or the scope.

Take the following steps to obtain the service URL.

1. Logon to the instance of Dynamics 365 to which you are connecting.
2. From the application dropdown, select **Settings** and then **Customizations**.
3. From **Customizations**, click **Developer Resources**.
4. From **Developer Resources**, the service URL is the URL in the **Service Root URL** field under the heading **Instance Web API**.

See also

OAuth 2.0 authentication on page 47

Obtain application client information and endpoints

You must obtain the following client information and endpoints in order to retrieve the OAuth tokens necessary for connecting to a Dynamics 365 service.

- Client ID - the client ID for your application
- Client Secret - the client secret for your application
- Authorization URI - the endpoint for obtaining an authorization code from the Azure authorization service
- Token URI - the endpoint used to exchange authentication credentials for access tokens

Take the following steps to obtain application client information and endpoints.

1. Go to the Azure portal.

   https://portal.azure.com

2. Select **Azure Active Directory**.
3. Select **App registrations** on the left.
4. Select the app that you have created.
5. From the app overview page, record the ClientID.
6. Obtain your ClientSecret.
   a) Select **Certificates & secrets**.
   b) Click **New client secret**.
   c) Enter description and choose an expiry option.
   d) Click **Add**.
   e) Copy the ClientSecret to a secure location.
7. Obtain authorization URI and token URI endpoints.
   a) Return to the **Overview** page for your application.
   b) Click **Endpoints** to display authorization and token endpoints associated with the application.

      Dynamics 365 supports two different APIs to manage user and application permissions: Azure Active Directory (v1) and Microsoft Identity Platform (v2). As shown in the following examples, authorization and token endpoints have different formats, depending on which API you are using.

      - Azure Active Directory (v1) authorization endpoint

        ```
        https://login.microsoftonline.com/common/oauth2/authorize
        ```

      - Microsoft Identity Platform (v2) authorization endpoint

        ```
        https://login.microsoftonline.com/common/oauth2/v2.0/authorize
        ```

   c) Record the AuthURI.

      - For v1, the AuthURI consists of the **OAuth 2.0 authorization endpoint (v1)** appended with `?resource=<resource_uri>` where the `<resource_uri>` is the URI for your Dynamics 365 service. For example:

        ```
        https://login.microsoftonline.com/common/oauth2/authorize?resource
           =https://mywebinstance.api.crm.dynamics.com
        ```

- For v2, the AuthURI is the **OAuth 2.0 authorization endpoint (v2)**. However, for v2, you must provide permissions via the Scope property. See Determine the Scope for details.

d) Record the TokenURI.

- For v1, the TokenURI is the **OAuth 2.0 token endpoint (v1)**.
- For v2, the TokenURI is the **OAuth 2.0 token endpoint (v2)**.

See also

OAuth 2.0 authentication on page 47

Determine the scope

Scope refers to the permissions associated with an application. Scope is determined by your Dynamics 365 administrator. If the Microsoft Identity Platform (v2) is being used to provision users and manage application access, then scope must be specified using the Scope property when retrieving the OAuth tokens necessary for connecting to a Dynamics 365 service.

The value of the Scope property may consist of an OAuth scope or a space-separated list of OAuth scopes. The following syntax applies to each scope specified by the scope property.

resource_uri`/`*scope_name* `offline_access`

where:

resource_uri

is the URI for your Dynamics 365 instance and is found at the start of the ServiceURL. For example, `https://mywebinstance.api.crm.dynamics.com` is the resource URI for the Service URL `https://mywebinstance.api.crm.dynamics.com/api/data/v9.1/`.

scope_name

is the name of a scope being enforced against the Dynamics 365 service.

`offline_access`

is a scope that enables prolonged access to resources on behalf of a user. This scope must be included if you are retrieving a refresh token.

Example

The following example shows a scope for a Dynamics CRM instance with the `user_impersonation` and `offline_access` scopes.

```
Scope=https://mywebinstance.api.crm.dynamics.com/user_impersonation offline_access
```

Note: The `user_impersonation` scope is a default scope for Dynamics CRM when using the v2 API. Refer to Microsoft Identity Plaform documentation for further details.

See also

OAuth 2.0 authentication on page 47

Obtain access and refresh tokens using the Configuration Manager

You need the following information before you begin.

- Service URL - the REST endpoint of the Dynamics 365 service to which you are connecting
- Authorization URI - the endpoint for obtaining an authorization code from the Azure authorization service
- Token URI - the endpoint used to exchange authentication credentials for access tokens
- Client ID - the client ID for your application
- Client Secret - the client secret for your application
- Scope - an OAuth scope or a space-separated list of OAuth scopes (only for Microsoft Identity Platform v2)

The following steps describe how you can use the Progress DataDirect Dynamics 365 Configuration Manager to obtain access and refresh tokens for either the access token flow or the refresh token grant. In addition, the Configuration Manager produces a connection URL that you can use in your application.

Note: You must allow popups in your browser to obtain access and refresh tokens with the Configuration Manager.

---

1. Open the Dynamics 365 Configuration Manager by double-clicking the driver jar file. Or, in a command line, navigate to the directory containing your driver jar file; then, execute the following command:

   ```
   java -jar dynamics365.jar
   ```

   The Dynamics 365 Configuration Manager opens in your default web browser.

2. Set **Authentication Method** to `OAuth2`.
3. Provide the following information in the fields provided.
   - **Service URL**
   - **Authorization URI**
   - **Token URI**
   - **Client ID**
   - **Client Secret**
   - **Scope** (only for Microsoft Identity Platform v2)

     Note: If you are using the Microsoft Identity Platform (v2) to provision users and manage application access, the scope must include the `offline_access` permission to retrieve a refresh token. This permission allows the Configuration Manager to retrieve a refresh token, and, by way of the refresh token grant, provides the application with prolonged access to the Dynamics 365 service. See also Determine the scope.

     ---

4. Retrieve access and refresh tokens.
   a) Click **Fetch OAuth Token**.
   b) If logon popup appears, enter Azure Active Directory credentials. (This popup may not appear if you previously logged on.)

c) If consent popup appears, provide consent, allowing the Configuration Manager to retrieve the tokens. (This popup may not appear if you previously provided consent to the Configuration Manager.)

d) The **Access Token** and **Refresh Token** fields populate with values retrieved from the OAuth authorization server.

5. Click **Test Connect** to verify connectivity and run SQL queries against the service.

Results:

The **Access Token** and **Refresh Token** fields include access and refresh tokens that you can use to implement OAuth 2.0.

The connection string in the **Connection String** field may be copied and used in your JDBC application to connect with your Dynamics 365 service.

Note:

Not all the values in the resulting connection string are required. However, the connection string can be copied directly into your JDBC application. The driver ignores any values that do not apply to your OAuth implementation.

For example, the refresh token grant connection string, derived from the Configuration Manager, might include the following properties.

```
jdbc:datadirect:dynamics365:ServiceURL=serviceurl;
AuthURI=auth_uri;TokenURI=token_uri;
ClientID=client_id;ClientSecret=client_secret;
AccessToken=access_token;RefreshToken=refresh_token;
Scope=scope;
```

However, only the following properties are required for a refresh token grant connection string.

```
jdbc:datadirect:dynamics365:ServiceURL=serviceurl;
TokenURI=token_uri;ClientID=client_id;
ClientSecret=client_secret;RefreshToken=refresh_token;
```

See also

OAuth 2.0 authentication on page 47

Create an application user for the client credentials grant

For the client credentials grant, you must create an application user on your Dynamics 365 web instance.

Take the following steps to create an application user.

1. Logon to your Dynamics 365 instance.
2. From the application dropdown, select **Settings** and then **Security**.
3. From **Security**, click **Users**.
4. From **Users**, click **New**.
5. Switch from the standard **User** form to the **User: Application User** form by clicking the **User** dropdown and selecting **Application User**.
6. Enter the following details in the **User: Application User** form.

Note: The **Application ID URI** and **Azure AD Object ID** fields are non-editable fields and may be left blank.

- Enter a user name.

- Enter the client ID of your application in the **Application ID** field.
- Enter a full name and primary email in the fields provided.

7. Save the profile of the application user by clicking the save icon in the lower-right corner of the browser.

Results:

You have successfully created an application user to implement the client credentials grant.

See also

OAuth 2.0 authentication on page 47
Client credentials grant on page 48

NTLM authentication

NTLM authentication can be used to authenticate with Dynamics 365.

To configure the driver to use NTLM:

- Set the ServiceURL property to the base URL of the Dynamics 365 instance to which you want to issue requests. For example, `http://myonpreminstance.sso3.dynamics365.net/api/data/v9.1/`.
- Set the AuthenticationMethod property to `NTLM`.
- Set the NTLMDomain property to specify the NTLM domain.
- Set the User property to specify the name of the user connecting to the service.
- Set the Password property to specify the password for the user connecting to the service.

Note: The User and Password properties are not required to be stored in the connection string. They can also be sent separately by the application.

The following examples show the connection information required to establish a session using NTLM.

Connection URL

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:dynamics365
    ServiceURL=http://myonpreminstance.sso3.dynamics365.net/api/data/v9.1/;
    AuthenticationMethod=NTLM;NTLMDomain=sso3;User=jsmith;Password=secret;");
```

Data Source

```
Dynamics365DataSource mds = new Dynamics365DataSource();
mds.setDescription("My Dynamics 365 Data Source");
mds.setServiceURL("http://myonpreminstance.sso3.dynamics365.net/api/data/v9.1/");
mds.AuthenticationMethod("NTLM");
mds.NTLMDomain("sso3");
mds.User("jsmith");
mds.Password("secret");
```

See also

Connection property descriptions on page 59

Data encryption

All communication between the driver and Dynamics 365 is encrypted using TLS/SSL.

Important: The driver complies with FIPS when FIPS mode is enabled with the client JVM. See "FIPS (Federal Information Processing Standard)" for more information.

FIPS (Federal Information Processing Standard)

The Federal Information Processing Standard (or FIPS) is a cryptography standard created by the U.S. government. FIPS specifications require certain secure algorithms, cryptographic modules, and random number generation. The driver is FIPS compliant for data encryption when FIPS is enabled for the JVM on the client machine.

The following applies when the driver is running in a FIPS environment:

- The driver complies with 140-3 and 140-2 standards.
- The driver uses PKCS #11 providers to access keystores.

The driver was tested with FIPS 140-3 enabled using Red Hat OpenJDK 21 on a Red Hat Universal Base Image 9 instance.

Limitations

Because certain algorithms used by Dynamics 365 are not FIPS compliant, NTLM authentication is not supported in a FIPS enabled environment. See "NTLM authentication" for more information.

See also

NTLM authentication on page 56

Performance considerations

BatchUpdateChunkSize: BatchUpdateChunkSize can impact performance by reducing the number of round trips required to complete an insert, update, or delete operation. For example, at the default setting of 200 rows, a 10,000 row insert operation would require 50 round trips. Setting BatchUpdateChunkSize to 500 rows would reduce the number of round trips to 20.

CreateMap: CreateMap determines whether the driver creates the internal files required for a relational map of the native data when establishing a connection. When the driver creates these internal files, an initial connection can take a few minutes, depending on network speeds and the amount of metadata the driver must retrieve from the service. For example, when CreateMap is set to OnChange (the default) and changes have been made to the Dynamics 365 back-end schema, the connection may be delayed because the driver runs a discovery on the Dynamics 365 schema and overwrites the internal files used to create a relational map of the data.

FetchSize: FetchSize can be used to adjust the trade-off between throughput and response time. In general, setting larger values for FetchSize will improve throughput, but can reduce response time. You should set FetchSize to suit your environment. Smaller fetch sizes can improve the initial response time of the query. Larger fetch sizes improve overall fetch times at the cost of additional memory.

InsensitiveResultSetBufferSize: To improve performance when using scroll-insensitive result sets, the driver can cache the result set data in memory instead of writing it to disk. By default, the driver caches 2 MB of insensitive result set data in memory and writes any remaining result set data to disk. Performance can be improved by increasing the amount of memory used by the driver before writing data to disk or by forcing the driver to never write insensitive result set data to disk. The maximum cache size setting is 2 GB.

ShowNameColumns: ShowNameColumns exposes an additional name column for Choice, Yes/No, and Lookup field types in Dynamics CRM apps. Enabling this property may impact runtime performance for two reasons. First, additional server processing is required to resolve field labels. Second, response payload sizes increase because both original and name column data are returned.

See also

BatchUpdateChunkSize on page 67
CreateMap on page 70
FetchSize on page 73
InsensitiveResultSetBufferSize on page 76
ShowNameColumns on page 89

4

Connection property descriptions

You can use connection properties to customize the driver for your environment. This section organizes connection properties according to functionality. You can use connection properties with either the JDBC `DriverManager` or a JDBC data source. For a `DriverManager` connection, a property is expressed as a key value pair and takes the form *`property=value`*. For a data source connection, a property is expressed as a JDBC method and takes the form `set`*`property`*`(`*`value`*`)`.

Note:

- In a JDBC data source, string values must be enclosed in double quotation marks, for example, `setUser("abc@defcorp.com")`.
- The data type listed for each connection property is the Java data type used for the property value in a JDBC data source.
- Connection property names are case-insensitive. For example, `Password` is the same as `password`.
- For connection properties that support string values, use the following escape sequence to specify values containing leading or trailing spaces and curly brackets: `{`*`value`*`}`. For example: `User={hello }` or `Password={{hello}}`.

The following tables describe the connection properties by functionality.

- Statement pooling properties
- Additional properties

OAuth 2.0 properties

The following table summarizes properties used for OAuth 2.0 authentication. The OAuth 2.0 properties you must specify depend on the grant type used in your environment. If you are unsure of the grant type or its requirements, contact your system administrator. For details on supported grant types, see OAuth 2.0 authentication on page 47.

Property	Data Source Method	Default
AccessToken on page 65	`getAccessToken()` `setAccessToken(String)`	No default value
AuthenticationMethod on page 66	`getAuthenticationMethod()` `setAuthenticationMethod(String)`	`OAuth2`
AuthURI on page 67	`getOauthAuthUri()` `setOauthAuthUri(String)`	No default value
ClientID on page 68	`getClientId()` `setClientId(String)`	No default value
ClientSecret on page 69	`getClientSecret()` `setClientSecret(String)`	No default value
RefreshToken on page 85	`getRefreshToken()` `setRefreshToken(String)`	No default value
Scope on page 88	`getOauthScope()` `setOauthScope(String)`	No default value
ServiceURL on page 88	`getServiceUrl()` `setServiceUrl(String)`	No default value
TokenURI on page 94	`getOauthTokenUri()` `setOauthTokenUri(String)`	No default value

NTLM properties

The following table summarizes properties used for NTLM.

Property	Data Source Method	Default
AuthenticationMethod on page 66	`getAuthenticationMethod()` `setAuthenticationMethod(String)`	No default value
NTLMDomain on page 79	`getNtlmDomain()` `setNtlmDomain(String)`	No default value
Password on page 80	`getPassword()` `setPassword(String)`	No default value
ServiceURL on page 88	`getServiceUrl()` `setServiceUrl(String)`	No default value
User on page 95	`getUser()` `setUser(String)`	No default value

Mapping properties

The following table summarizes connection properties involved in mapping the Dynamics 365 data model to a SQL model.

Property	Data Source Method	Default
CreateMap on page 70	`getCreateMap()` `setCreateMap(String)`	`OnChange`
CustomPrefix on page 72	`getCustomPrefix()` `setCustomPrefix(String)`	`Include`
CustomPrefixName on page 73	`getCustomPrefixName()` `setCustomPrefixName(String)`	`new_`
KeywordConflictSuffix on page 77	`getKeywordConflictSuffix()` `setKeywordConflictSuffix(String)`	No default value
RefreshSchema on page 84	`getRefreshSchema()` `setRefreshSchema(Boolean)`	`false`
SchemaMap on page 86	`getSchemaMap()` `setSchemaMap(String)`	Default value depends on environment
ShowNameColumns on page 89	`getShowNameColumns()` `setShowNameColumns(Boolean)`	`false`

Proxy server properties

The following table summarizes proxy server connection properties.

Property	Data Source Method	Default
ProxyHost on page 80	`getProxyHost()` `setProxyHost(String)`	No default value
ProxyPassword on page 81	`getProxyPassword()` `setProxyPassword(String)`	No default value
ProxyPort on page 82	`getProxyPort()` `setProxyPort(Integer)`	`0` which means the default is determined by the ProxyHost property. For HTTP URLs: `80` For HTTPS URLs: `443`
ProxyUser on page 82	`getProxyUser()` `setProxyUser(String)`	No default value

Web service properties

The following table summarizes Web service connection properties.

Property	Data Source Method	Default
StmtCallLimit on page 92	`getStmtCallLimit()` `setStmtCallLimit(Integer)`	`0` (no limit)
StmtCallLimitBehavior on page 93	`getStmtCallLimitBehavior()` `setStmtCallLimitBehavior(String)`	`ErrorAlways`
WSCompressData on page 96	`getWsCompressData()` `setWsCompressData(String)`	`Compress`
WSTimeout on page 96	`getWSTimeout()` `setWSTimeout(Integer)`	`300`

Statement pooling properties

The following table summarizes statement pooling connection properties.

Property	Data Source Method	Default
ImportStatementPool on page 75	`getImportStatementPool()` `setImportStatementPool(String)`	No default value

Property	Data Source Method	Default
MaxPooledStatements on page 78	`getMaxPooledStatements()` `setMaxPooledStatements(Integer)`	`0`
RegisterStatementPoolMonitorMBean on page 85	`getRegisterStatementPoolMonitorMbean()` `setRegisterStatementPoolMonitorMbean(Boolean)`	`false`

Additional properties

The following table summarizes additional connection properties.

Property	Data Source Method	Default
BatchUpdateChunkSize on page 67	`getBatchUpdateChunkSize()` `setBatchUpdateChunkSize(Integer)`	`200`
ConvertNull on page 70	`getConvertNull()` `setConvertNull(Integer)`	`true` (data type check is performed if column value is null)
CrossCompany on page 71	`getCrossCompany()` `setCrossCompany(Boolean)`	`false`
FetchSize on page 73	`getFetchSize()` `setFetchSize(Integer)`	`100` (rows)
InitializationString on page 75	`getInitializationString()` `setInitializationString(String)`	No default value
InsensitiveResultSetBufferSize on page 76	`getInsensitiveResultsetBufferSize()` `setInsensitiveResultsetBufferSize(Integer)`	`2048` (KB of memory)
LogConfigFile on page 77	`getLogConfigFile()` `setLogConfigFile(String)`	`ddlogging.properties`
ReadOnly on page 83	`getReadOnly()` `setReadOnly(Boolean)`	`false`

Property	Data Source Method	Default
SpyAttributes on page 90	`getSpyAttributes()` `setSpyAttributes(String)`	No default value
TransactionMode on page 94	`getTransactionMode()` `setTransactionMode(String)`	`NoTransactions`

For details, see the following topics:

- AccessToken
- AuthenticationMethod
- AuthURI
- BatchUpdateChunkSize
- ClientID
- ClientSecret
- ConvertNull
- CreateMap
- CrossCompany
- CustomPrefix
- CustomPrefixName
- FetchSize
- ImportStatementPool
- InitializationString
- InsensitiveResultSetBufferSize
- KeywordConflictSuffix
- LogConfigFile
- MaxPooledStatements
- NTLMDomain
- Password
- ProxyHost
- ProxyPassword
- ProxyPort
- ProxyUser
- ReadOnly

AccessToken

Purpose

Specifies the access token used to authenticate to Dynamics 365 with OAuth 2.0 enabled. Typically, this property is configured by the application; however, in some scenarios, you may need to secure a token using external processes. In those instances, you can also use this property to set the access token manually.

Valid Values

String

where:

String

is an access token you have obtained from the authentication service.

Notes

- Access tokens expire ten minutes after generation. Once connected, the access token remains valid till the session is disconnected.
- See "OAuth 2.0 authentication" for examples and more information.

Data Source Methods

```
public String getAccessToken()
```

```
public void setAccessToken(String)
```

Default Value

No default value

Data Type

String

See also

OAuth 2.0 authentication on page 47
Access token flow on page 47

AuthenticationMethod

Purpose

Determines which authentication method the driver uses during the course of a session.

Valid Values

`OAuth2` | `NTLM`

Behavior

If set to `OAuth2`, the driver uses OAuth 2.0 to authenticate with the service.

If set to `NTLM`, the driver uses NLTM to authenticate with the service.

Data Source Methods

```
public String getAuthenticationMethod()
public void setAuthenticationMethod(String)
```

Default Value

`OAuth2`

Data Type

String

See also

Connection URL examples on page 14
OAuth 2.0 authentication on page 47
NTLM authentication on page 56

AuthURI

Purpose

Specifies the endpoint for obtaining an authorization code from a third-party authorization service for OAuth 2.0 implementations.

Valid Values

String

where:

String

is the endpoint for retrieving the OAuth 2.0 authorization code from the third party authorization service.

Notes

- When this endpoint is queried, the authorization service presents an interface prompting the user to approve or deny access to backend data.
- See "OAuth 2.0 authentication" for examples and more information.

Data Source Methods

```
public String getAuthUri()
```

```
public void setAuthUri(String)
```

Default Value

No default value

Data Type

String

See also

OAuth 2.0 authentication on page 47
Obtain application client information and endpoints on page 52

BatchUpdateChunkSize

Purpose

Specifies the maximum number of rows sent to the service when an insert, update, or delete operation impacts multiple rows.

Valid Values

x

where:

x

is a number of rows greater than or equal to 1.

Notes

This property can impact performance by reducing the number of round trips required to complete an insert, update, or delete operation. For example, at the default setting of 200 rows, a 10,000 row insert operation would require 50 round trips. Setting BatchUpdateChunkSize to 500 rows would reduce the number of round trips to 20.

Data Source Methods

```
public Integer getBatchUpdateChunkSize()
```

```
public void setBatchUpdateChunkSize(Integer)
```

Default Value

```
200
```

Data Type

Integer

See also

Performance considerations on page 57

ClientID

Purpose

Specifies the client ID key for your application when authenticating to Dynamics 365 with OAuth 2.0 enabled.

Valid Values

String

where:

String

is the client ID key for your application.

Notes

See "OAuth 2.0 authentication" for more information.

Data Source Methods

```
public String getClientId()
```

```
public void setClientId(String)
```

Default Value

No default value

Data Type

String

See also

OAuth 2.0 authentication on page 47
Obtain application client information and endpoints on page 52

ClientSecret

Purpose

Specifies the client secret for your application when authenticating to Dynamics 365 with OAuth 2.0 enabled.

Important: The client secret is a confidential value used to authenticate the application to the service. To prevent unauthorized access, this value must be securely maintained.

Valid Values

String

where:

String

is the client secret for your application.

Notes

See "OAuth 2.0 authentication" for more information.

Data Source Methods

```
public String getClientSecret()
```

```
public void setClientSecret(String)
```

Default Value

No default value

Data Type

String

See also

OAuth 2.0 authentication on page 47
Obtain application client information and endpoints on page 52

ConvertNull

Purpose

Controls how data conversions are handled for null values.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver checks the data type being requested against the data type of the table column that stores the data. If a conversion between the requested type and column type is not defined, the driver generates an "unsupported data conversion" exception regardless of whether the column value is `NULL`.

If set to `false`, the driver does not perform the data type check if the value of the column is `NULL`. This allows null values to be returned even though a conversion between the requested type and the column type is undefined.

Data Source Methods

```
public Boolean getConvertNull()
```

```
public void setConvertNull(Boolean)
```

Default Value

`true`

Data Type

Boolean

CreateMap

Purpose

Determines whether the driver creates the internal files required for a relational map of the native data model when establishing a connection.

Valid Values

`NotExist` | `ForceNew` | `No` | `Session` | `OnChange`

Behavior

If set to `NotExist`, the driver uses the current group of internal files specified by SchemaMap. If the files do not exist, the driver creates them.

If set to `ForceNew`, the driver deletes the group of internal files specified by SchemaMap and creates a new group of these files at the same location.

If set to `No`, the driver uses the current group of internal files specified by SchemaMap. If the files do not exist, the connection fails.

If set to `Session`, the driver uses memory to store the internal configuration information and relational map of the native data model. After the session, the view is discarded.

If set to `OnChange`, the driver checks for changes to the native data model for CRM apps. If changes to the schema are detected, the driver deletes the group of internal files specified by SchemaMap and creates a new group of these files at the same location. For ERP apps, `OnChange` is ignored, and the driver uses the current group of internal files specified by SchemaMap.

Notes

- The internal files share the same directory as the schema map's configuration file. This directory is specified with the SchemaMap connection property.
- You can refresh the internal files related to an existing relational view of your data with the SQL extension Refresh Map. Refresh Map runs a discovery against your native data and updates your internal files accordingly.
- The default behavior for ERP apps is different from the default behavior for CRM apps. For ERP apps, the default setting `OnChange` is ignored, and the driver uses the current group of internal files specified by SchemaMap. If you are using the driver with an ERP app, the `ForceNew` or `Session` settings can be used to pick up changes made to the native data model. Alternatively, you may use the Refresh Map SQL extension to discover changes to the native data model.

Data Source Methods

```
public String getCreateMap()
```

```
public void setCreateMap(String)
```

Default Value

`OnChange`

Data Type

String

See also

SchemaMap on page 86
Performance considerations on page 57

CrossCompany

Purpose

Determines whether the driver can fetch data only from the default company of the user or from all the companies to which the user has access.

Valid Values

`true` | `false`

Behavior

If set to `false`, the driver can fetch data only from the default company of the user.

If set to `true`, the driver can fetch data from all the companies to which the user has access.

Notes

- The CrossCompany connection property is supported only for ERP apps.

Data Source Methods

```
public Boolean getCrossCompany()
```

```
public void setCrossCompany(Boolean)
```

Default Value

`false`

Data Type

Boolean

CustomPrefix

Purpose

Determines whether the driver includes or strips a prefix denoting user-defined objects from table and column names when mapping the Dynamics 365 data model.

Valid Values

`Strip` | `Include`

Behavior

If set to `Strip`, the driver strips the prefix.

If set to `Include`, the driver includes the prefix.

Notes

When this property is set to `Strip`, the driver removes the prefix `new_` from any field or object that begins with this string. This property may be used to specify a prefix other than `new_`.

Data Source Methods

```
public String getCustomPrefix()
```

```
public void setCustomPrefix(String)
```

Default Value

`Include`

Data Type

String

See also

CustomPrefixName on page 73

CustomPrefixName

Purpose

Specifies the prefix that the driver removes from table and column names. When CustomPrefix is set to `Strip`, the driver removes this prefix from objects and fields when mapping the Dynamics 365 data model.

Valid Values

`String`

where:

`String`

is a string of alphanumeric characters.

Data Source Methods

```
public String getCustomPrefixName()
```

```
public void setCustomPrefixName(String)
```

Default Value

No default value

Data Type

String

See also

CustomPrefix on page 72

FetchSize

Purpose

Specifies the maximum number of rows that the driver processes before returning data to the application when executing a Select. This value provides a suggestion to the driver as to the number of rows it should internally process before returning control to the application. The driver may fetch fewer rows to conserve memory when processing exceptionally wide rows.

Valid Values

`0` | *x*

where:

x

is a positive integer indicating the number of rows that should be processed.

Behavior

If set to `0`, the driver processes all the rows of the result before returning control to the application. When large data sets are being processed, setting FetchSize to `0` can diminish performance and increase the likelihood of out-of-memory errors.

If set to *x*, the driver limits the number of rows that may be processed for each fetch request before returning control to the application.

Notes

- To optimize throughput and conserve memory, the driver uses an internal algorithm to determine how many rows should be processed based on the width of rows in the result set. Therefore, the driver may process fewer rows than specified by FetchSize when the result set contains exceptionally wide rows. Alternatively, the driver processes the number of rows specified by FetchSize when the result set contains rows of unexceptional width.
- FetchSize can be used to adjust the trade-off between throughput and response time. Smaller fetch sizes can improve the initial response time of the query. Larger fetch sizes can improve overall response times at the cost of additional memory.
- You can use FetchSize to reduce demands on memory and decrease the likelihood of out-of-memory errors. Simply, decrease FetchSize to reduce the number of rows the driver is required to process before returning data to the application.

Data Source Methods

```
public Integer getFetchSize()
```

```
public void setFetchSize(Integer)
```

Default Value

`100`

Data Type

Integer

See also

Performance considerations on page 57

ImportStatementPool

Purpose

Specifies the path and file name of the file to be used to load the contents of the statement pool. When this property is specified, statements are imported into the statement pool from the specified file.

Valid Values

String

where:

String

is the path and file name of the file to be used to load the contents of the statement pool.

Notes

- If the driver cannot locate the specified file when establishing the connection, the connection fails and the driver throws an exception.
- For more information, refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public String getImportStatementPool()
```

```
public void setImportStatementPool(String)
```

Default Value

No default value

Data Type

String

InitializationString

Purpose

Specifies one or multiple SQL commands to be executed by the driver after it has established a connection and has performed all initialization for the connection. If the execution of a SQL command fails, the connection attempt also fails and the driver throws an exception indicating which SQL command or commands failed.

Valid Values

command[[;*command*]...]

where:

command

is a SQL command.

Notes

Multiple commands must be separated by semicolons. In addition, if this property is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified.

Data Source Methods

```
public String getInitializationString()
```

```
public void setInitializationString(String)
```

Default Value

No default value

Data Type

String

InsensitiveResultSetBufferSize

Purpose

Determines the amount of memory that is used by the driver to cache insensitive result set data.

Valid Values

`-1` | `0` | *x*

where:

x

is a positive integer that represents the amount of memory.

Behavior

If set to `-1`, the driver caches insensitive result set data in memory. If the size of the result set exceeds available memory, an OutOfMemoryException is generated. With no need to write result set data to disk, the driver processes the data efficiently.

If set to `0`, the driver caches insensitive result set data in memory, up to a maximum of 2 MB. If the size of the result set data exceeds available memory, then the driver pages the result set data to disk, which can have a negative performance effect. Because result set data may be written to disk, the driver may have to reformat the data to write it correctly to disk.

If set to *x*, the driver caches insensitive result set data in memory and uses this value to set the size (in KB) of the memory buffer for caching insensitive result set data. If the size of the result set data exceeds available memory, then the driver pages the result set data to disk, which can have a negative performance effect. Because the result set data may be written to disk, the driver may have to reformat the data to write it correctly to disk. Specifying a buffer size that is a power of 2 results in efficient memory use.

Data Source Methods

```
public Integer getInsensitiveResultsetBufferSize()
```

```
public void setInsensitiveResultsetBufferSize(Integer)
```

Default Value

2048

Data Type

Integer

KeywordConflictSuffix

Purpose

Specifies a string of up to 5 alphanumeric characters that the driver appends to any object or field name that conflicts with a SQL engine keyword.

Valid Values

String

where:

String

is a string of up to 5 alphanumeric characters.

Example

A field called CASE exists in the data schema. To avoid a naming conflict with the SQL engine keyword CASE, you could set KeywordConflictSuffix=TAB. In this scenario, the driver maps the CASE field to the CASETAB column.

Data Source Methods

```
public String getKeywordConflictSuffix()
```

```
public void setKeywordConflictSuffix(String)
```

Default Value

No default value

Data Type

String

LogConfigFile

Purpose

Specifies the file name, and optionally, the path of the properties file used to initialize driver logging.

Valid Values

String

where:

String

is the relative or fully qualified path of the properties file to load to initialize driver logging. If you do not specify a path, the driver looks for this file in the current working directory. If the specified file does not exist, the driver continues searching for an appropriate properties file as described in "Using Java Logging" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public String getLogConfigFile()
```

```
public void setLogConfigFile(String)
```

Default Value

ddlogging.properties

Data Type

String

MaxPooledStatements

Purpose

Specifies the maximum number of prepared statements to be pooled for each connection and enables the driver's internal prepared statement pooling when set to an integer greater than zero (0). The driver's internal prepared statement pooling provides performance benefits when the driver is not running from within an application server or another application that provides its own statement pooling.

Valid Values

0 | *x*

where:

x

is a positive integer that represents a number of prepared statements to be cached.

Behavior

If set to 0, the driver's internal prepared statement pooling is not enabled.

If set to *x*, the driver's internal prepared statement pooling is enabled and the driver uses the specified value to cache up to that many prepared statements created by an application. If the value set for this property is greater than the number of prepared statements that are used by the application, all prepared statements that are created by the application are cached. Because CallableStatement is a sub-class of PreparedStatement, CallableStatements also are cached.

Example

If the value of this property is set to `20`, the driver caches the last 20 prepared statements that are created by the application.

Notes

When you enable statement pooling, your applications can access the Statement Pool Monitor directly with DataDirect-specific methods. However, you can also enable the Statement Pool Monitor as a JMX MBean. To enable the Statement Pool Monitor as an MBean, statement pooling must be enabled with MaxPooledStatements and the Statement Pool Monitor MBean must be registered using the RegisterStatementPoolMonitorMBean connection property.

Data Source Methods

```
public Integer getMaxPooledStatements()
```

```
public void setMaxPooledStatements(Integer)
```

Default Value

`0`

Data Type

Integer

See also

ImportStatementPool on page 75

NTLMDomain

Purpose

Specifies the NTLM domain when NTLM authentication is enabled (`AuthenticationMethod=NTLM`).

Valid Values

`String`

where:

`String`

is the NTLM domain used for NTLM authentication.

Data Source Methods

```
public String getNtlmDomain()
```

```
public void setNtlmDomain(String)
```

Default Value

No default value

Data Type

String

See also

NTLM on page 18
NTLM authentication on page 56

Password

Purpose

A password that is used to connect to the service.

Important: Setting the password using a data source is not recommended. The data source persists all properties, including password, in clear text.

Behavior

password

where:

password

is a valid password. The password is case-sensitive.

Data Source Methods

```
public String getPassword()
```

```
public void setPassword(String)
```

Default Value

No default value

Data Type

String

See also

NTLM on page 18
NTLM authentication on page 56

ProxyHost

Purpose

Identifies a proxy server to use for the first connection.

Valid Values

server_name | *IP_address*

where:

server_name

is the name of the proxy server, which may be qualified with the domain name.

IP_address

is an IP address, specified in either IPv4 or IPv6 format, or a combination of the two.

Data Source Methods

```
public String getProxyHost()
public void setProxyHost(String)
```

Default Value

No default value

Data Type

String

See also

Proxy server on page 19

ProxyPassword

Purpose

Specifies the password needed to connect to a proxy server for the first connection.

Valid Values

password

where:

password

is a valid password for that server. Contact your system administrator to obtain a valid password.

Data Source Methods

```
public String getProxyPassword()
public void setProxyPassword(String)
```

Default Value

No default value

Data Type

String

See also

Proxy server on page 19

ProxyPort

Purpose

Specifies the port number where the proxy server is listening for HTTP or HTTPS requests for the first connection.

Valid Values

port

where:

port

is the port number on which the proxy server is listening. Contact your system administrator to obtain the correct port.

Data Source Methods

```
public Integer getProxyPort()
```

```
public void setProxyPort(Integer)
```

Default Value

`0` which means that the default value is determined by whether the value specified for the ProxyHost property is an HTTP or HTTPS URL.

For HTTP: `80`

For HTTPS: `443`

Data Type

Integer

See also

Proxy server on page 19

ProxyUser

Purpose

Specifies the user name needed to connect to a proxy server for the first connection.

Valid Values

user_name

where:

user_name

is a valid user ID for the proxy server.

Data Source Methods

```
public String getProxyUser()
```

```
public void setProxyUser(String)
```

Default Value

No default value

Data Type

String

See also

Proxy server on page 19

ReadOnly

Purpose

Specifies whether the connection has read-only access to the data source.

Valid Values

`true` | `false`

Behavior

If set to `true`, the connection has read-only access.

If set to `false`, the connection is opened for read/write access, and you can use all commands supported by the product.

Notes

You can use use the JDBC connection method setReadOnly to set a read-only state for a connection.

Data Source Methods

```
public Boolean getReadOnly()
```

```
public void setReadOnly(Boolean)
```

Default Value

`false`

Data Type

Boolean

RefreshSchema

Purpose

Specifies whether the driver automatically refreshes the relational map of the schema when a user connects to the service.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver automatically refreshes the map when a user first connects to the service. The driver rebuilds the relational map of the schema, and any changes that have been made to the schema since the last refresh will be shown in the metadata.

If set to `false`, the driver does not refresh the relational map when a user first connects.

Notes

- This property should not be enabled (`RefreshSchema=true`) when `CreateMap=Session`.
- This property is equivalent to executing the Refresh Schema statement.

Data Source Methods

```
public Boolean getRefreshSchema()
```

```
public void setRefreshSchema(Boolean)
```

Default Value

`false`

Data Type

Boolean

See also

SchemaMap on page 86
CreateMap on page 70
Refresh Map (EXT) on page 105

RefreshToken

Purpose

Specifies the refresh token used to either request a new access token or renew an expired access token for OAuth 2.0 implementations.

Important: The refresh token is a confidential value used to authenticate to the service. To prevent unauthorized access, this value must be securely maintained.

Valid Values

String

where:

String

is the refresh token you have obtained from the Dynamics 365 service.

Notes

- See "OAuth 2.0 authentication" for more information.

Data Source Methods

```
public String getRefreshToken()
```

```
public void setRefreshToken(String)
```

Default Value

No default value

Data Type

String

See also

OAuth 2.0 authentication on page 47
Refresh token grant on page 49

RegisterStatementPoolMonitorMBean

Purpose

Registers the Statement Pool Monitor as a JMX MBean when statement pooling has been enabled with MaxPooledStatements. This allows you to manage statement pooling with standard JMX API calls and to use JMX-compliant tools, such as JConsole.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver registers an MBean for the statement pool monitor for each statement pool. This gives applications access to the Statement Pool Monitor through JMX when statement pooling is enabled.

If set to `false`, the driver does not register an MBean for the Statement Pool Monitor for any statement pool.

Notes

- Registering the MBean exports a reference to the Statement Pool Monitor. The exported reference can prevent garbage collection on connections if the connections are not properly closed. When garbage collection does not take place on these connections, out of memory errors can occur.
- For more information, refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public Boolean getRegisterStatementPoolMonitorMbean()
```

```
public void setRegisterStatementPoolMonitorMbean(Boolean)
```

Default Value

`false`

Data Type

Boolean

SchemaMap

Purpose

Specifies either the name or the absolute path and name of the configuration file where the map of the Dynamics 365 schema is written. The driver looks for this file when connecting to Dynamics 365. If the file does not exist, the driver creates one.

Valid Values

`String`

where:

`String`

is either the name or the absolute path and name (including the `.config` extension) of the configuration file. For example, if SchemaMap is set to a value of:

- `ABC`, the driver either creates or looks for the configuration file `ABC` in the working directory of your application.
- `C:\\Users\\Default\\AppData\\Local\\Progress\\DataDirect\\Dynamics365_Schema\\abc@defcorp.com.config`, the driver either creates or looks for the configuration file `abc@defcorp.com.config` in the directory `C:\Users\Default\AppData\Local\Progress\DataDirect\Dynamics365_Schema`.

Example

Escapes are needed when specifying SchemaMap for a data source, but they are not used when specifying SchemaMap in a DriverManager connection URL.

DriverManager example

```
SchemaMap=C:\Users\Default\AppData\Local\Progress\DataDirect\Dynamics365_Schema\abc.config
```

DataSource example

```
setSchemaMap("C:\\Users\\Default\\AppData\\Local\\Progress\\DataDirect\\Dynamics365_Schema\\abc.config")
```

Notes

- When connecting to the service, the driver looks for the schema map configuration file. If the configuration file does not exist, the driver creates the schema map configuration file using the name and location you have provided. If you do not provide a name and location for the configuration file, the driver creates it using default values.
- When using NTLM authentication, the default prefix name for the configuration file is the user name. When using OAuth 2.0 authentication, the default prefix is the client ID.
- The driver uses the path specified in this connection property to store additional internal files.
- You can refresh the internal files related to an existing view of your data by using the SQL extension Refresh Map. Refresh Map runs a discovery against your native data and updates your internal files accordingly.

Data Source Methods

```
public String getSchemaMap()
```

```
public void setSchemaMap(String)
```

Default Value

The default is determined by the environment. The driver attempts to create the files in a subdirectory of the first available directory in the following order:

- Windows
  - DD_HOME environment variable
  - dd.home system property
  - LOCALAPPDATA environment variable
  - APPDATA environment variable
  - user.home system property
- UNIX/Linux
  - DD_HOME environment variable
  - dd.home system property
  - user.home system property

For both Windows and UNIX/Linux, the file path takes the following format.

```
<available_location>/progress/datadirect/Dynamics365_Schema/<user_name>.config
```

Data Type

String

Scope

Purpose

Specifies a space-separated list of OAuth scopes that limit the permissions granted by an access token.

Valid Values

String

where:

String

is a space-separated list of security scopes.

Data Source Methods

`public String getScope()`

`public void setScope(String)`

Default Value

No default value

Data Type

String

See also

OAuth 2.0 authentication on page 47
Determine the scope on page 53

ServiceURL

Purpose

Specifies the endpoint for the Dynamics 365 service to which you are connecting.

Valid Values

String

where:

String

is the Dynamics 365 endpoint.

Data Source Methods

```
public String getServiceUrl()
```

```
public void setServiceUrl(String)
```

Default Value

No default value

Data Type

String

See also

OAuth 2.0 authentication on page 47
NTLM authentication on page 56
Obtain the service URL on page 51

ShowNameColumns

Purpose

Exposes an additional name column for Choice, Yes/No, and Lookup field types in Dynamics CRM apps.

Valid Values

`true` | `false`

Behavior

If set to `false`, the driver does not expose name columns.

If set to `true`, the driver exposes name columns.

Example

If a user creates a column named "PRIORITY" and implements it as a Choice field with "High" and "Low" options, enabling this property adds a "PRIORITYNAME" column with type String that contains the text labels "High" and "Low".

Notes

- This property works only with Dynamics CRM apps.
- Enabling this property may impact runtime performance due to increased server processing and larger response payloads.

Data Source Methods

```
public Boolean getShowNameColumns()
```

```
public void setShowNameColumns(Boolean)
```

Default Value

`false`

Data Type

Boolean

See also

Performance considerations on page 57

SpyAttributes

Purpose

Enables DataDirect Spy to log detailed information about calls that are issued by the driver on behalf of the application. DataDirect Spy is not enabled by default.

Valid Values

(*spy_attribute*[;*spy_attribute*]...)

where:

spy_attribute

is any valid DataDirect spy attribute.

Behavior

Attribute	Description
`linelimit=`*`numberofchars`*	Sets the maximum number of characters that DataDirect Spy logs on a single line. The default is 0 (no maximum limit).
`log=(file)`*`filename`*	Directs logging to the file specified by *`filename`*. For Windows, if coding a path to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: `log=(file)C:\\temp\\spy.log;logIS=yes;logTName=yes`

Attribute	Description
log=(filePrefix)*file_prefix*	Directs logging to a file prefixed by *file_prefix*. The log file is named *file_prefixX*.log where: *X* is an integer that increments by 1 for each connection on which the prefix is specified. For example, if the attribute log=(filePrefix)C:\\temp\\spy_ is specified on multiple connections, the following logs are created: C:\temp\spy_1.log C:\temp\spy_2.log C:\temp\spy_3.log ... If coding a path to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: log=(filePrefix)C:\\temp\\spy_;logIS=yes;logTName=yes.
log=System.out	Directs logging to the Java output standard, System.out.
logIS={ yes | no | nosingleread }	Specifies whether DataDirect Spy logs activity on InputStream and Reader objects. When logIS=nosingleread, logging on InputStream and Reader objects is active; however, logging of the single-byte read InputStream.read or single-character Reader.read is suppressed to prevent generating large log files that contain single-byte or single character read messages. The default is no.
logLobs={ yes | no }	Specifies whether DataDirect Spy logs activity on BLOB and CLOB objects.
logTName={ yes | no }	Specifies whether DataDirect Spy logs the name of the current thread. The default is no.
timestamp={ yes | no }	Specifies whether a timestamp is included on each line of the DataDirect Spy log. The default is no.

Example

The following value instructs the driver to log all JDBC activity to a file using a maximum of 80 characters for each line.

```
(log=(file)/tmp/spy.log;linelimit=80)
```

Notes

- If coding a path on Windows to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: `log=(file)C:\\temp\\spy.log`.
- If a log file name does not include the `.log` extension, the driver automatically appends it. For example, a file named `spy.jsp` is renamed to `spy.jsp.log` by the driver.
- For more information, refer to "Tracking JDBC calls with DataDirect Spy" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public String getSpyAttributes()
```

```
public void setSpyAttributes(String)
```

Default Value

No default value

Data Type

String

StmtCallLimit

Purpose

Specifies the maximum number of web service calls the driver can make when executing any single SQL statement or metadata query.

Valid Values

`0` | *x*

where:

x is a positive integer that defines the maximum number of web service calls up to 2147483647 the driver can make when executing any single SQL statement or metadata query.

Behavior

If set to `0`, there is no limit.

If set to *x*, the driver uses this value to set the maximum number of web service calls on a single connection that can be made when executing a SQL statement. This limit can be overridden by changing the `STMT_CALL_LIMIT` session attribute using the `ALTER SESSION` statement. For example, the following statement sets the statement call limit to 10 web service calls:

```
ALTER SESSION SET STMT_CALL_LIMIT=10
```

If the web service call limit is exceeded, the behavior of the driver depends on the value specified for the `StmtCallLimitBehavior` property.

Data Source Methods

```
public Integer getStmtCallLimit()
```

```
public void setStmtCallLimit(Integer)
```

Default Value

`0`

Data Type

Integer

See also

StmtCallLimitBehavior on page 93

StmtCallLimitBehavior

Purpose

Specifies the behavior of the driver when the maximum web service call limit specified by the StmtCallLimit property is exceeded.

Valid Values

`ReturnResults` | `ErrorAlways`

Behavior

If set to `ReturnResults`, the driver returns any partial results it received prior to the call limit being exceeded. The driver generates a warning that not all of the results were fetched.

If set to `ErrorAlways`, the driver generates an exception if the maximum web service call limit is exceeded.

Data Source Methods

```
public String getStmtCallLimitBehavior()
```

```
public void setStmtCallLimitBehavior(String)
```

Default Value

`ErrorAlways`

Data Type

String

See also

StmtCallLimit on page 92

TokenURI

Purpose

Specifies the endpoint for retrieving access tokens when OAuth 2.0 authentication is enabled.

Valid Values

String

where:

String

is the endpoint used to retrieve access tokens.

Notes

See "OAuth 2.0 authentication" for more information.

Data Source Methods

```
public String getTokenUri()
public void setTokenUri(String)
```

Default Value

No default value

Data Type

String

See also

OAuth 2.0 authentication on page 47
Obtain application client information and endpoints on page 52

TransactionMode

Purpose

Specifies how the driver handles manual transactions.

Valid Values

`NoTransactions` | `Ignore`

Behavior

If set to `NoTransactions`, the data source and the driver do not support transactions. Metadata indicates that the driver does not support transactions.

If set to `Ignore`, the data source does not support transactions and the driver always operates in auto-commit mode. Calls to set the driver to manual commit mode and to commit transactions are ignored. Calls to rollback a transaction cause the driver to throw an exception indicating that no transaction is started. Metadata indicates that the driver supports transactions and the ReadUncommitted transaction isolation level.

Data Source Methods

```
public String getTransactionMode()
```

```
public void setTransactionMode(String)
```

Default Value

`NoTransactions`

Data Type

String

User

Purpose

Specifies the user name that is used to connect to the service.

Valid Values

`String`

where:

`String`

is a valid user name. The user name is case-insensitive.

Data Source Methods

```
public String getUser()
```

```
public void setUser(String)
```

Default Value

No default value

Data Type

String

See also

NTLM on page 18
NTLM authentication on page 56

WSCompressData

Purpose

Specifies whether the driver compresses data it sends to or receives from the service.

Valid Values

`Compress` | `None`

Behavior

If set to `Compress`, the driver sends and receives compressed data to and from the service.

If set to `None`, the driver sends and receives uncompressed data to and from the service.

Notes

Setting WSCompressData to `None` can significantly degrade performance.

Data Source Methods

```
public String getWSCompressData()
public void setWSCompressData(String)
```

Default Value

`Compress`

Data Type

String

WSTimeout

Purpose

Specifies the time, in seconds, that the driver waits for a response to a web service request.

Valid Values

`0` | *x*

where:

x

is a positive integer that defines the number of seconds the driver waits for a response to a web service request.

Behavior

If set to `0`, the driver waits indefinitely for a response; there is no timeout.

If set to *x*, the driver uses the value as the default timeout, measured in seconds, for any statement created by the connection.

If a Select request times out and WSRetryCount is set to retry timed-out requests, the driver retries the request the specified number of times.

Data Source Methods

```
public Integer getWSTimeout()
```

```
public void setWSTimeout(Integer)
```

Default Value

`300`

Data Type

Integer

5

Supported SQL statements and extensions

The driver provides support for the SQL statements and the SQL extensions described in this section. SQL extensions are denoted by an (EXT) in the topic title.

For details, see the following topics:

Alter Session (EXT)

Purpose

Changes various attributes of a local or remote session. A local session maintains the state of the overall connection. A remote session maintains the state that pertains to a particular remote data source connection.

Syntax

```
ALTER SESSION SET attribute_name=value
```

where:

attribute_name

specifies the name of the attribute to be changed. Attributes apply to either local or remote sessions.

value

specifies the value for that attribute.

The following table lists the local and remote session attributes, and provides descriptions of each.

Table 4: Alter Session Attributes

Attribute Name	Session Type	Description
Current_Schema	Local	Sets the current schema for the local session. The current schema is the schema used when an identifier in a SQL statement is unqualified. The string value must be the name of a schema visible in the local session. For example: `ALTER SESSION SET CURRENT_SCHEMA=DYNAMICS 365`

Attribute Name	Session Type	Description
Stmt_Call_Limit	Local	Sets the maximum number of Web service calls the driver can make in executing a statement. Setting the Stmt_Call_Limit attribute has the same effect as setting the Statement Call Limit connection option. It sets the default Web service call limit used by any statement on the connection. Executing this command on a statement overrides the previously set Statement Call Limit for the connection. The value specified must be a positive integer or `0`. The value `0` means that no call limit exists. For example: `ALTER SESSION SET STMT_CALL_LIMIT=150`

Ws_Call_Count	Remote	Resets the Web service call count of a remote session to the value specified. The value must be `0` or a positive integer. WS_Call_Count represents the total number of Web service calls made to the remote data source instance for the current session. For example:
		`ALTER SESSION SET dynamics 365.WS_CALL_COUNT=0`
		The current value of WS_Call_Count can be obtained by referring to the System_Remote_Sessions system table (see SYSTEM_REMOTE_SESSIONS Catalog Table for details). For example:
		`SELECT * from information_schema.system_remote_sessions WHERE session_id = cursessionid()`

Delete

Purpose

The Delete statement is used to delete rows from a table.

Syntax

`DELETE FROM` *`table_name`* `[WHERE` *`search_condition`*`]`

where:

`table_name`

specifies the name of the table from which you want to delete rows.

`search_condition`

is an expression that identifies which rows to delete from the table.

Notes

- The Where clause determines which rows are to be deleted. Without a Where clause, all rows of the table are deleted, but the table is left intact. See "Where Clause" for information about the syntax of Where clauses. Where clauses can contain subqueries.

Example A

This example shows a Delete statement on the emp table.

```
DELETE FROM emp WHERE emp_id = 'E10001'
```

Each Delete statement removes every record that meets the conditions in the Where clause. In this case, every record having the employee ID `E10001` is deleted. Because employee IDs are unique in the employee table, at most, one record is deleted.

Example B

This example shows using a subquery in a Delete clause.

```
DELETE FROM emp WHERE dept_id = (SELECT dept_id FROM dept WHERE dept_name =
'Marketing')
```

The records of all employees who belong to the department named Marketing are deleted.

Notes

- To enable Insert, Update, and Delete, set the ReadOnly connection option to `false`.

See also

Where clause on page 110

Explain Plan

Purpose

Retrieves a detailed list of the elements in the execution plan. It generates a result set with a single column named `OPERATION`. The individual elements that comprise the plan are returned as rows in the result set.

Syntax

```
EXPLAIN PLAN FOR {SELECT ... | DELETE ... | INSERT ... | UPDATE ...}
```

The returned list of elements includes the indexes used for performing the query and can be used to optimize the query.

Insert

Purpose

The Insert statement is used to add new rows to a local table. You can specify either of the following options:

- List of values to be inserted as a new row
- Select statement that copies data from another table to be inserted as a set of new rows

Syntax

```
INSERT INTO table_name [(column_name[,column_name]...)] {VALUES (expression
[,expression]...) | select_statement}
```

table_name

is the name of the table in which you want to insert rows.

column_name

is optional and specifies an existing column. Multiple column names (a column list) must be separated by commas. A column list provides the name and order of the columns, the values of which are specified in the Values clause. If you omit a *column_name* or a column list, the value expressions must provide values for all columns defined in the table and must be in the same order that the columns are defined for the table. Table columns that do not appear in the column list are populated with the default value, or with NULL if no default value is specified.

expression

is the list of expressions that provides the values for the columns of the new record. Typically, the expressions are constant values for the columns. Character string values must be enclosed in single quotation marks (’). See "Literals" for more information.

select_statement

is a query that returns values for each *column_name* value specified in the column list. Using a Select statement instead of a list of value expressions lets you select a set of rows from one table and insert it into another table using a single Insert statement. The Select statement is evaluated before any values are inserted. This query cannot be made on the table into which values are inserted. See "Select" for information about Select statements.

Notes

- To enable Insert, Update, and Delete, set the ReadOnly connection option to `false`.

See also

Literals on page 119
Select on page 105

Specifying an external ID column

Use the following syntax to specify an external ID column to look up the value of a foreign key column.

Syntax

column_name `EXT_ID [`*schema_name*`.[`*table_name*`.] ]`*ext_id_column*

where:

`EXT_ID`

is used to specify that the column specified by *ext_id_column* is used to look up the rowid to be inserted into the column specified by *column_name*.

schema_name

is the name of the schema of the table that contains the foreign key column being specified as the external ID column.

table_name

is the name of the table that contains the foreign key column being specified as the external ID column.

ext_id_column

is the external ID column.

Example A

This example uses a list of expressions to insert records. Each Insert statement adds one record to the table. In this case, one record is added to the table emp. Values are specified for five columns. The remaining columns in the table are assigned the default value or NULL if no default value is specified.

```
INSERT INTO emp (last_name,
                    first_name,
                    emp_id,
                    salary,
                    hire_date)
VALUES ('Smith', 'John', 'E22345', 27500, {1999-04-06})
```

Example B

This example uses a Select statement to insert records. The number of columns in the result of the Select statement must match exactly the number of columns in the table if no column list is specified, or it must match the number of column names specified in the column list. A new entry is created in the table for every row of the Select result.

```
INSERT INTO emp1 (first_name,
                  last_name,
                  emp_id,
                  dept,
                  salary)
SELECT first_name, last_name, emp_id, dept, salary FROM emp
WHERE dept = 'D050'
```

Example C

This example uses a list of expressions to insert records and specifies an external ID column (a foreign key column) named accountId that references a table that has an external ID column named AccountNum.

```
INSERT INTO emp (last_name,
                    first_name,
                    emp_id,
                    salary,
                    hire_date,
                    accountId EXT_ID AccountNum)
VALUES ('Smith', 'John', 'E22345', 27500, {1999-04-06}, 0001)
```

Refresh Map (EXT)

Purpose

The REFRESH MAP statement adds newly discovered objects to your relational view of native data. It also incorporates any configuration changes made to your relational view by reloading the schema definition and associated files.

Syntax

```
REFRESH MAP
```

Notes

- REFRESH MAP is an expensive query since it involves the discovery of native data.

See also

CreateMap on page 70
RefreshSchema on page 84
SchemaMap on page 86

Select

Purpose

Use the Select statement to fetch results from one or more tables.

Syntax

```
SELECT select_clause from_clause
[where_clause]
[groupby_clause]
[having_clause]
[{UNION [ALL | DISTINCT] |
 {MINUS [DISTINCT] | EXCEPT [DISTINCT]} |
 INTERSECT [DISTINCT]} select_statement]
[limit_clause]
```

where:

select_clause

specifies the columns from which results are to be returned by the query. See "Select clause" for a complete explanation.

from_clause

specifies one or more tables on which the other clauses in the query operate. See "From clause" for a complete explanation.

where_clause

is optional and restricts the results that are returned by the query. See "Where clause" for a complete explanation.

groupby_clause

is optional and allows query results to be aggregated in terms of groups. See "Group By clause" for a complete explanation.

having_clause

is optional and specifies conditions for groups of rows (for example, display only the departments that have salaries totaling more than $200,000). See "Having clause" for a complete explanation.

UNION

is an optional operator that combines the results of the left and right Select statements into a single result. See "Union operator" for a complete explanation.

INTERSECT

is an optional operator that returns a single result by keeping any distinct values from the results of the left and right Select statements. See "Intersect operator" for a complete explanation.

EXCEPT | MINUS

are synonymous optional operators that returns a single result by taking the results of the left Select statement and removing the results of the right Select statement. See "Except and Minus operators" for a complete explanation.

orderby_clause

is optional and sorts the results that are returned by the query. See "Order By clause" for a complete explanation.

limit_clause

is optional and places an upper bound on the number of rows returned in the result. See "Limit clause" for a complete explanation.

Select clause

Purpose

Use the Select clause to specify with a list of column expressions that identify columns of values that you want to retrieve or an asterisk (*) to retrieve the value of all columns.

Syntax

```
SELECT [{LIMIT offsetnumber | TOP number}] [ALL | DISTINCT] {* | column_expression
[[AS] column_alias] [,column_expression [[AS] column_alias], ...]}
```

where:

`LIMIT offset number`

creates the result set for the Select statement first and then discards the first number of rows specified by *offset* and returns the number of remaining rows specified by *number*. To not discard any of the rows, specify `0` for *offset*, for example, `LIMIT 0` *number*. To discard the first *offset* number of rows and return all the remaining rows, specify `0` for *number*, for example, `LIMIT` *offset*`0`.

`TOP` *number*

is equivalent to `LIMIT 0`*number*.

column_expression

can be simply a column name (for example, `last_name`). More complex expressions may include mathematical operations or string manipulation (for example, `salary * 1.05`). See "SQL expressions" for details. *column_expression* can also include aggregate functions. See "Aggregate functions" for details.

column_alias

can be used to give the column a descriptive name. For example, to assign the alias department to the column dep:

```
SELECT dep AS department FROM emp
```

`DISTINCT`

eliminates duplicate rows from the result of a query. This operator can precede the first column expression. For example:

```
SELECT DISTINCT dep FROM emp
```

Notes

- Separate multiple column expressions with commas (for example, `SELECT last_name, first_name, hire_date`).
- Column names can be prefixed with the table name or table alias. For example, `SELECT emp.last_name` or `e.last_name`, where `e` is the alias for the table `emp`.
- NULL values are not treated as distinct from each other. The default behavior is that all result rows be returned, which can be made explicit with the keyword `ALL`.

See also

SQL expressions on page 119

Aggregate functions

Aggregate functions can also be a part of a Select clause. Aggregate functions return a single value from a set of rows. An aggregate can be used with a column name (for example, `AVG(salary)`) or in combination with a more complex column expression (for example, `AVG(salary * 1.07)`).

The following table lists supported aggregate functions.

Note: Doubly nested aggregates, such as `SUM(COUNT(col1))`, are currently not permitted by the driver.

Table 5: Aggregate Functions

Aggregate	Returns
AVG	The average of the values in a numeric column expression. For example, `AVG(salary)` returns the average of all salary column values.
COUNT	The number of values in any field expression. For example, `COUNT(name)` returns the number of name values. When using `COUNT` with a field name, `COUNT` returns the number of non-NULL column values. A special example is `COUNT(*)`, which returns the number of rows in the set, including rows with NULL values. **Note:** The driver does not support `COUNT(DISTINCT *)`. For example, `SELECT COUNT(DISTINCT *) FROM mytable` results in a syntax error.
MAX	The maximum value in any column expression. For example, `MAX(salary)` returns the maximum salary column value.
MIN	The minimum value in any column expression. For example, `MIN(salary)` returns the minimum salary column value.
SUM	The total of the values in a numeric column expression. For example, `SUM(salary)` returns the sum of all salary column values.

Example

The following example uses the `COUNT`, `MAX`, and `AVG` aggregate functions:

```
SELECT
     COUNT(amount) AS numOpportunities,
     MAX(amount) AS maxAmount,
     AVG(amount) AS avgAmount
FROM opportunity o INNER JOIN user u
     ON o.ownerId = u.id
WHERE o.isClosed = 'false' AND
     u.name = 'MyName'
```

From clause

Purpose

The From clause indicates the tables to be used in the Select statement.

Syntax

`FROM` *`table_name`* `[`*`table_alias`*`] [,...]`

where:

`table_name`

is the name of a table or a subquery. Multiple tables define an implicit inner join among those tables. Multiple table names must be separated by a comma. For example:

```
SELECT * FROM emp, dep
```

Subqueries can be used instead of table names. Subqueries must be enclosed in parentheses. See "Subquery in a From clause" for an example.

table_alias

is a name used to refer to a table in the rest of the Select statement. When you specify an alias for a table, you can prefix all column names of that table with the table alias.

Example

The following example specifies two table aliases, e for emp and d for dep:

```
SELECT e.name, d.deptName
FROM emp e, dep d
WHERE e.deptId = d.id
```

table_alias is a name used to refer to a table in the rest of the Select statement. When you specify an alias for a table, you can prefix all column names of that table with the table alias. For example, given the table specification:

```
FROM emp E
```

you may refer to the last_name field as E.last_name. Table aliases must be used if the Select statement joins a table to itself. For example:

```
SELECT * FROM emp E, emp F WHERE E.mgr_id = F.emp_id
```

The equal sign (=) includes only matching rows in the results.

Join in a From clause

Purpose

You can use a Join as a way to associate multiple tables within a Select statement. Joins may be either explicit or implicit. For example, the following is the example from the previous section restated as an explicit inner join:

```
SELECT * FROM emp INNER JOIN dep ON id=empId
SELECT e.name, d.deptName
FROM emp e INNER JOIN dep d ON e.deptId = d.id;
```

whereas the following is the same statement as an implicit inner join:

```
SELECT * FROM emp, dep WHERE emp.deptID=dep.id
```

Note: The ON clause in a join expression must evaluate to a true or false value.

Syntax

```
FROM table_name {RIGHT OUTER | INNER | LEFT OUTER | CROSS | FULL OUTER} JOIN table.key
 ON search-condition
```

Example

In this example, two tables are joined using LEFT OUTER JOIN. T1, the first table named includes nonmatching rows.

```
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.key = T2.key
```

If you use a CROSS JOIN, no ON expression is allowed for the join.

Subquery in a From clause

Subqueries can be used in the From clause in place of table references (*table_name*).

Example

```
SELECT * FROM (SELECT * FROM emp WHERE sal > 10000) new_emp, dept WHERE
new_emp.deptno = dept.deptno
```

See also

Subqueries on page 116

Where clause

Purpose

Specifies the conditions that rows must meet to be retrieved.

Syntax

WHERE *expr1 rel_operator expr2*

where:

expr1

is either a column name, literal, or expression.

expr2

is either a column name, literal, expression, or subquery. Subqueries must be enclosed in parentheses.

rel_operator

is the relational operator that links the two expressions.

Example

The following Select statement retrieves the first and last names of employees that make at least $20,000.

```
SELECT last_name, first_name FROM emp WHERE salary >= 20000
```

See also

SQL expressions on page 119
Subqueries on page 116

Group By clause

Purpose

Specifies the names of one or more columns by which the returned values are grouped. This clause is used to return a set of aggregate values.

Syntax

```
GROUP BY column_expression [,...]
```

where:

column_expression

is either a column name or a SQL expression. Multiple values must be separated by a comma. If *column_expression* is a column name, it must match one of the column names specified in the Select clause. Also, the Group By clause must include all non-aggregate columns specified in the Select list.

Example

The following example totals the salaries in each department:

```
SELECT dept_id, sum(salary) FROM emp GROUP BY dept_id
```

This statement returns one row for each distinct department ID. Each row contains the department ID and the sum of the salaries of the employees in the department.

See also

SQL expressions on page 119
Subqueries on page 116

Having clause

Purpose

Specifies conditions for groups of rows (for example, display only the departments that have salaries totaling more than $200,000). This clause is valid only if you have already defined a Group By clause.

Syntax

```
HAVING expr1 rel_operator expr2
```

where:

expr1 | expr2

can be column names, constant values, or expressions. These expressions do not have to match a column expression in the Select clause. See "SQL expressions" for details regarding SQL expressions.

rel_operator

is the relational operator that links the two expressions.

Example

The following example returns only the departments that have salaries totaling more than $200,000:

```
SELECT dept_id, sum(salary) FROM emp GROUP BY dept_id HAVING sum(salary) > 200000
```

See also

SQL expressions on page 119
Subqueries on page 116

Union operator

Purpose

Combines the results of two Select statements into a single result. The single result is all the returned rows from both Select statements. By default, duplicate rows are not returned. To return duplicate rows, use the All keyword (`UNION ALL`).

Syntax

```
select_statement
UNION [ALL | DISTINCT] | {MINUS [DISTINCT] | EXCEPT [DISTINCT]} | INTERSECT
[DISTINCT]select_statement
```

Notes

- When using the Union operator, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT last_name, salary, hire_date FROM emp
UNION
SELECT name, pay, birth_date FROM person
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`salary FROM emp` has a different data type than `last_name FROM raises`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT last_name, salary FROM emp
UNION
SELECT salary, last_name FROM raises
```

Intersect operator

Purpose

Intersect operator returns a single result set. The result set contains rows that are returned by both Select statements. Duplicates are returned unless the Distinct operator is added.

Syntax

```
select_statement
INTERSECT [DISTINCT]
select_statement
```

where:

`DISTINCT`

eliminates duplicate rows from the results.

Notes

- When using the Intersect operator, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT last_name, salary, hire_date FROM emp
INTERSECT [DISTINCT]
SELECT name, pay, birth_date FROM person
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`salary FROM emp` has a different data type than `last_name FROM raises`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT last_name, salary FROM emp
INTERSECT
SELECT salary, last_name FROM raises
```

Except and Minus operators

Purpose

Return the rows from the left Select statement that are not included in the result of the right Select statement.

Syntax

```
select_statement
{EXCEPT [DISTINCT] | MINUS [DISTINCT]}
select_statement
```

where:

`DISTINCT`

eliminates duplicate rows from the results.

Notes

- When using one of these operators, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT last_name, salary, hire_date FROM emp
EXCEPT
SELECT name, pay, birth_date FROM person
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`salary FROM emp` has a different data type than `last_name FROM raises`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT last_name, salary FROM emp
EXCEPT
SELECT salary, last_name FROM raises
```

Order By clause

Purpose

The Order By clause specifies how the rows are to be sorted.

Syntax

`ORDER BY` *`sort_expression`* `[DESC | ASC] [,...]`

where:

`sort_expression`

is either the name of a column, a column alias, a SQL expression, or the positioned number of the column or expression in the select list to use.

The default is to perform an ascending (ASC) sort.

Example

To sort by `last_name` and then by `first_name`, you could use either of the following Select statements:

```
SELECT emp_id, last_name, first_name FROM emp
ORDER BY last_name, first_name
```

or

```
SELECT emp_id, last_name, first_name FROM emp
ORDER BY 2,3
```

In the second example, `last_name` is the second item in the Select list, so `ORDER BY 2,3` sorts by `last_name` and then by `first_name`.

See also

SQL expressions on page 119

Limit clause

Purpose

Places an upper bound on the number of rows returned in the result.

Syntax

`LIMIT number_of_rows [OFFSET offset_number]`

where:

number_of_rows

specifies a maximum number of rows in the result. A negative number indicates no upper bound.

OFFSET

specifies how many rows to skip at the beginning of the result set. *offset_number* is the number of rows to skip.

Notes

- In a compound query, the Limit clause can appear only on the final Select statement. The limit is applied to the entire query, not to the individual Select statement to which it is attached.

Example

The following example returns a maximum of 20 rows.

```
SELECT last_name, first_name FROM emp WHERE salary > 20000 ORDER BY dept_idc LIMIT
20
```

Update

Purpose

An Update statement changes the value of columns in the selected rows of a table.

Syntax

`UPDATE table_name SET column_name = expression`

`[, column_name = expression] [WHERE conditions]`

table_name

is the name of the table for which you want to update values.

column_name

is the name of a column, the value of which is to be changed. Multiple column values can be changed in a single statement.

expression

is the new value for the column. The expression can be a constant value or a subquery that returns a single value. Subqueries must be enclosed in parentheses.

Example A

The following example changes every record that meets the conditions in the Where clause. In this case, the salary and exempt status are changed for all employees having the employee ID `E10001`. Because employee IDs are unique in the emp table, only one record is updated.

```
UPDATE emp SET salary=32000, exempt=1
WHERE emp_id = 'E10001'
```

Example B

The following example uses a subquery. In this example, the salary is changed to the average salary in the company for the employee having employee ID `E10001`.

```
UPDATE emp SET salary = (SELECT avg(salary) FROM emp)
WHERE emp_id = 'E10001'
```

Notes

- To enable Insert, Update, and Delete, set the ReadOnly connection option to `false`.
- A Where clause can be used to restrict which rows are updated.

See also

Subqueries on page 116
Where clause on page 110

Subqueries

A query is an operation that retrieves data from one or more tables or views. In this reference, a top-level query is called a Select statement, and a query nested within a Select statement is called a subquery.

A subquery is a query expression that appears in the body of another expression such as a Select, an Update, or a Delete statement. In the following example, the second Select statement is a subquery:

```
SELECT * FROM emp WHERE deptno IN (SELECT deptno FROM dept)
```

IN predicate

Purpose

The In predicate specifies a set of values against which to compare a result set. If the values are being compared against a subquery, only a single column result set is returned.

Syntax

value [NOT] IN (*value1*, *value2*,...)

OR

value [NOT] IN (*subquery*)

Example

```
SELECT * FROM emp WHERE deptno IN
(SELECT deptno FROM dept WHERE dname <> 'Sales')
```

EXISTS predicate

Purpose

The Exists predicate is true only if the cardinality of the subquery is greater than 0; otherwise, it is false.

Syntax

EXISTS (*subquery*)

Example

```
SELECT empno, ename, deptno FROM emp e WHERE EXISTS
(SELECT deptno FROM dept WHERE e.deptno = dept.deptno)
```

UNIQUE predicate

Purpose

The Unique predicate is used to determine whether duplicate rows exist in a virtual table (one returned from a subquery).

Syntax

UNIQUE (*subquery*)

Example

```
SELECT * FROM dept d WHERE UNIQUE
(SELECT deptno FROM emp e WHERE e.deptno = d.deptno)
```

Correlated subqueries

Purpose

A correlated subquery is a subquery that references a column from a table referred to in the parent statement. A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a Select, Update, or Delete statement.

A correlated subquery answers a multiple-part question in which the answer depends on the value in each row processed by the parent statement. For example, you can use a correlated subquery to determine which employees earn more than the average salaries for their departments. In this case, the correlated subquery specifically computes the average salary for each department.

Syntax

```
SELECT select_list
   FROM table1 t_alias1
   WHERE expr rel_operator
   (SELECT column_list
       FROM table2 t_alias2
       WHERE t_alias1.columnrel_operatort_alias2.column)
UPDATE table1 t_alias1
   SET column =
   (SELECT expr
       FROM table2 t_alias2
        WHERE t_alias1.column = t_alias2.column)
DELETE FROM table1 t_alias1
   WHERE column rel_operator
   (SELECT expr
       FROM table2 t_alias2
       WHERE t_alias1.column = t_alias2.column)
```

Notes

- Correlated column names in correlated subqueries must be explicitly qualified with the table name of the parent.

Example A

The following statement returns data about employees whose salaries exceed their department average. This statement assigns an alias to emp, the table containing the salary information, and then uses the alias in a correlated subquery:

```
SELECT deptno, ename, sal FROM emp x WHERE sal >
  (SELECT AVG(sal) FROM emp WHERE x.deptno = deptno)
  ORDER BY deptno
```

Example B

This is an example of a correlated subquery that returns row values:

```
SELECT * FROM dept "outer" WHERE 'manager' IN
  (SELECT managername FROM emp
  WHERE "outer".deptno = emp.deptno)
```

Example C

This is an example of finding the department number (deptno) with multiple employees:

```
SELECT * FROM dept main WHERE 1 <
  (SELECT COUNT(*) FROM emp WHERE deptno = main.deptno)
```

Example D

This is an example of correlating a table with itself:

```
SELECT deptno, ename, sal FROM emp x WHERE sal >
  (SELECT AVG(sal) FROM emp WHERE x.deptno = deptno)
```

SQL expressions

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value. You can use expressions in the Where, and Having of Select statements; and in the Set clauses of Update statements.

Expressions enable you to use mathematical operations as well as character string manipulation operators to form complex queries.

The driver supports both unquoted and quoted identifiers. An unquoted identifier must start with an ASCII alpha character and can be followed by zero

Quoted identifiers must be enclosed in double quotation marks (""). A quoted identifier can contain any Unicode character including the space character. The driver recognizes the Unicode escape sequence \uxxxx as a Unicode character. You can specify a double quotation mark in a quoted identifier by escaping it with a double quotation mark.

The maximum length of both quoted and unquoted identifiers is 128 characters.

Valid expression elements are:

- Column names
- Literals
- Operators
- Functions

Column names

The most common expression is a simple column name. You can combine a column name with other expression elements.

Literals

Literals are fixed data values. For example, in the expression `PRICE * 1.05`, the value 1.05 is a constant. Literals are classified into types, including the following:

- Binary
- Character string
- Date
- Floating point
- Integer
- Numeric
- Time
- Timestamp

The following table describes the literal format for supported SQL data types.

Table 6: Literal Syntax Examples

SQL Type	Literal Syntax	Example
BIGINT	*n* where *n* is any valid integer value in the range of the INTEGER data type	`12 or -34 or 0`
BOOLEAN	Min Value: 0 Max Value: 1	`0` `1`
DATE	DATE'*date*'	`'2010-05-21'`
DATETIME	TIMESTAMP'*ts*'	`'2010-05-21 18:33:05.025'`

DECIMAL	*n.f*
	where:
	n
	is the integral part
	f
	is the fractional part
		`0.25`
		`3.1415`
		`-7.48`

DOUBLE	*n.f*E*x* where: *n* is the integral part *f* is the fractional part *x* is the exponent	`1.2E0` or `2.5E40` or `-3.45E2` or `5.67E-4`
INTEGER	*n* where *n* is a valid integer value in the range of the INTEGER data type	`12` or `-34` or `0`
LONGVARBINARY	'*hex_value*'	`'000482ff'`
LONGVARCHAR	'*value*'	`'This is a string literal'`
TIME	TIME'*time*'	`'2010-05-21 18:33:05.025'`
VARCHAR	'*value*'	`'This is a string literal'`

Character string literals

Text specifies a character string literal. A character string literal must be enclosed in single quotation marks. To represent one single quotation mark within a literal, you must enter two single quotation marks. When the data in the fields is returned to the client, trailing blanks are stripped.

A character string literal can have a maximum length of 32 KB, that is, (32*1024) bytes.

Example

```
'Hello'
'Jim''s friend is Joe'
```

Numeric literals

Unquoted numeric values are treated as numeric literals. If the unquoted numeric value contains a decimal point or exponent, it is treated as a real literal; otherwise, it is treated as an integer literal.

Example

```
+1894.1204
```

Binary literals

Binary literals are represented with single quotation marks. The valid characters in a binary literal are 0-9, a-f, and A-F.

Example

```
'00af123d'
```

Date/Time literals

Date and time literal values are enclosed in single quotion marks ('*value*').

- The format for a Date literal is DATE'*date*'.
- The format for a Time literal is TIME'*time*'.
- The format for a Timestamp literal is TIMESTAMP'*ts*'.

Integer literals

Integer literals are represented by a string of numbers that are not enclosed in quotation marks and do not contain decimal points.

Notes

- Integer constants must be whole numbers; they cannot contain decimals.
- Integer literals can start with sign characters (+/-).

Example

```
1994 or -2
```

Operators

This section describes the operators that can be used in SQL expressions.

Note: Numeric operators are restricted to numeric types. Numeric operators do not support non-numeric types.

Unary operator

A unary operator operates on only one operand.

operator operand

Binary operator

A binary operator operates on two operands.

operand1 operator operand2

If an operator is given a null operand, the result is always null. The only operator that does not follow this rule is concatenation (||).

Arithmetic operators

You can use an arithmetic operator in an expression to negate, add, subtract, multiply, and divide numeric values. The result of this operation is also a numeric value. The + and - operators are also supported in date/time fields to allow date arithmetic. The following table lists the supported arithmetic operators.

Table 7: Arithmetic Operators

Operator	Purpose	Example
+ -	Denotes a positive or negative expression. These are unary operators.	`SELECT * FROM emp WHERE comm = -1`
* /	Multiplies, divides. These are binary operators.	`UPDATE emp SET sal = sal + sal * 0.10`
+ -	Adds, subtracts. These are binary operators.	`SELECT sal + comm FROM emp WHERE empno > 100`

Concatenation operator

The concatenation operator manipulates character strings. The following table lists the only supported concatenation operator.

Table 8: Concatenation Operator

Operator	Purpose	Example
||	Concatenates character strings.	`SELECT 'Name is' || ename FROM emp`

The result of concatenating two character strings is the data type VARCHAR.

Comparison operators

Comparison operators compare one expression to another. The result of such a comparison can be TRUE, FALSE, or UNKNOWN (if one of the operands is NULL). The driver considers the UNKNOWN result as FALSE.

The following table lists the supported comparison operators.

Table 9: Comparison Operators

Operator	Purpose	Example
=	Equality test.	`SELECT * FROM emp WHERE sal = 1500`
!=<>	Inequality test.	`SELECT * FROM emp WHERE sal != 1500`
><	"Greater than" and "less than" tests.	`SELECT * FROM emp WHERE sal > 1500 SELECT * FROM emp WHERE sal < 1500`
>=<=	"Greater than or equal to" and "less than or equal to" tests.	`SELECT * FROM emp WHERE sal >= 1500 SELECT * FROM emp WHERE sal <= 1500`
LIKE	% and _ wildcards can be used to search for a pattern in a column. The percent sign denotes zero, one, or multiple characters, while the underscore denotes a single character. The right-hand side of a LIKE expression must evaluate to a string or binary.	`SELECT * FROM emp WHERE ENAME LIKE 'J%'`

ESCAPE clause in LIKE operator
LIKE 'pattern string' ESCAPE 'c'
	The Escape clause is supported in the LIKE predicate to indicate the escape character. Escape characters are used in the pattern string to indicate that any wildcard character that is after the escape character in the pattern string should be treated as a regular character.
	The default escape character is backslash (\).
		`SELECT * FROM emp WHERE ENAME LIKE 'J%\_%' ESCAPE '\'`
		This matches all records with names that start with letter 'J' and have the '_' character in them.
		`SELECT * FROM emp WHERE ENAME LIKE 'JOE\_JOHN' ESCAPE '\'`
		This matches only records with name 'JOE_JOHN'.

[NOT] IN	"Equal to any member of" test.	`SELECT * FROM emp WHERE job IN ('CLERK','ANALYST') SELECT * FROM emp WHERE sal IN (SELECT sal FROM emp WHERE deptno = 30)`
[NOT] BETWEEN x AND y	"Greater than or equal to x" and "less than or equal to y."	`SELECT * FROM emp WHERE sal BETWEEN 2000 AND 3000`

Operator	Purpose	Example
EXISTS	Tests for existence of rows in a subquery.	`SELECT empno, ename, deptno FROM emp e WHERE EXISTS (SELECT deptno FROM dept WHERE e.deptno = dept.deptno)`
IS [NOT] NULL	Tests whether the value of the column or expression is NULL.	`SELECT * FROM emp WHERE ename IS NOT NULL SELECT * FROM emp WHERE ename IS NULL`

Logical operators

A logical operator combines the results of two component conditions to produce a single result or to invert the result of a single condition. The following table lists the supported logical operators.

Table 10: Logical Operators

Operator	Purpose	Example
NOT	Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE. If it is UNKNOWN, it remains UNKNOWN.	`SELECT * FROM emp WHERE NOT (job IS NULL)` `SELECT * FROM emp WHERE NOT (sal BETWEEN 1000 AND 2000)`
AND	Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE; otherwise, returns UNKNOWN.	`SELECT * FROM emp WHERE job = 'CLERK' AND deptno = 10`
OR	Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE; otherwise, returns UNKNOWN.	`SELECT * FROM emp WHERE job = 'CLERK' OR deptno = 10`

Example

In the Where clause of the following Select statement, the AND logical operator is used to ensure that managers earning more than $1000 a month are returned in the result:

```
SELECT * FROM emp WHERE jobtitle = manager AND sal > 1000
```

Operator precedence

As expressions become more complex, the order in which the expressions are evaluated becomes important. The following table shows the order in which the operators are evaluated. The operators in the first line are evaluated first, then those in the second line, and so on. Operators in the same line are evaluated left to right in the expression.You can change the order of precedence by using parentheses. Enclosing expressions in parentheses forces them to be evaluated together.

Table 11: Operator Precedence

Precedence	Operator
1	+ (Positive), - (Negative)
2	*(Multiply), / (Division)
3	+ (Add), - (Subtract)
4	|| (Concatenate)
5	=, >, <, >=, <=, <>, != (Comparison operators)
6	NOT, IN, LIKE
7	AND
8	OR

Example A

The query in the following example returns employee records for which the department number is 1 or 2 and the salary is greater than $1000:

```
SELECT * FROM emp WHERE (deptno = 1 OR deptno = 2) AND sal > 1000
```

Because parenthetical expressions are forced to be evaluated first, the OR operation takes precedence over AND.

Example B

In the following example, the query returns records for all the employees in department 1, but only employees whose salary is greater than $1000 in department 2.

```
SELECT * FROM emp WHERE deptno = 1 OR deptno = 2 AND sal > 1000
```

The AND operator takes precedence over OR, so that the search condition in the example is equivalent to the expression `deptno = 1 OR (deptno = 2 AND sal > 1000)`.

Functions

The driver supports a number of functions that you can use in expressions, including String, Numeric, Timedate, and System functions.

Refer to "Scalar functions" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

Conditions

A condition specifies a combination of one or more expressions and logical operators that evaluates to either TRUE, FALSE, or UNKNOWN. You can use a condition in the Where clause of the Delete, Select, and Update statements; and in the Having clauses of Select statements. The following describes supported conditions.

Table 12: Conditions

Condition	Description
Simple comparison	Specifies a comparison with expressions or subquery results. `= , !=, <>, < , >, <=, <=`
Group comparison	Specifies a comparison with any or all members in a list or subquery. `[= , !=, <>, < , >, <=, <=] [ANY, ALL, SOME]`
Membership	Tests for membership in a list or subquery. `[NOT] IN`
Range	Tests for inclusion in a range. `[NOT] BETWEEN`
NULL	Tests for nulls. `IS NULL, IS NOT NULL`
EXISTS	Tests for existence of rows in a subquery. `[NOT] EXISTS`
LIKE	Specifies a test involving pattern matching. `[NOT] LIKE`
Compound	Specifies a combination of other conditions. `CONDITION [AND/OR] CONDITION`