PUBLIC

SAP BusinessObjects Business Intelligence platform

Document Version: 4.3 Support Package 4 – 2023-12-07

Data Access Driver Developer Guide

© 2024 SAP SE or an SAP affiliate company. All rights reserved.

THE BEST RUN SAP

Content

1 Document History. 3

2 About this Guide. 4

3 Data Access Driver Development System. 5

3.1 System Architecture. 5

3.2 Working with Data Access Driver. 6

Open Drivers. 7

Setting up the Driver Configuration Files. 7

Configuring Drivers. 9

Using Connection Pool. 12

Configuring New Connection Wizard. 14

4 Setting up the DDK Environment. 15

5 Creating Connections. 16

6 Preparing for Data Request 18

7 Requesting Data from the Data Source. 21

8 Closing the Connection. 22

1 Document History

The following table provides an overview of important document changes.

Version	Date	Description
Data Access Driver Developer Guide	August, 2016	First release of this document.

2 About this Guide

This guide gives you information on how to develop a custom driver using Data Access Driver Development Kit (DDK), how to customize the drivers for any Open Connectivity and use the drivers as part of the SAP BusinessObjects BI Platform.

In addition, DDK provides you Java APIs for accessing Connection Objects from BI Platform and using the Connection Objects in the custom driver.

Who Should use this Guide

Data Access Driver Developer Guide is for developers who want to create custom drivers and consume the customized driver as a part of the SAP BusinessObjects BI Platform.

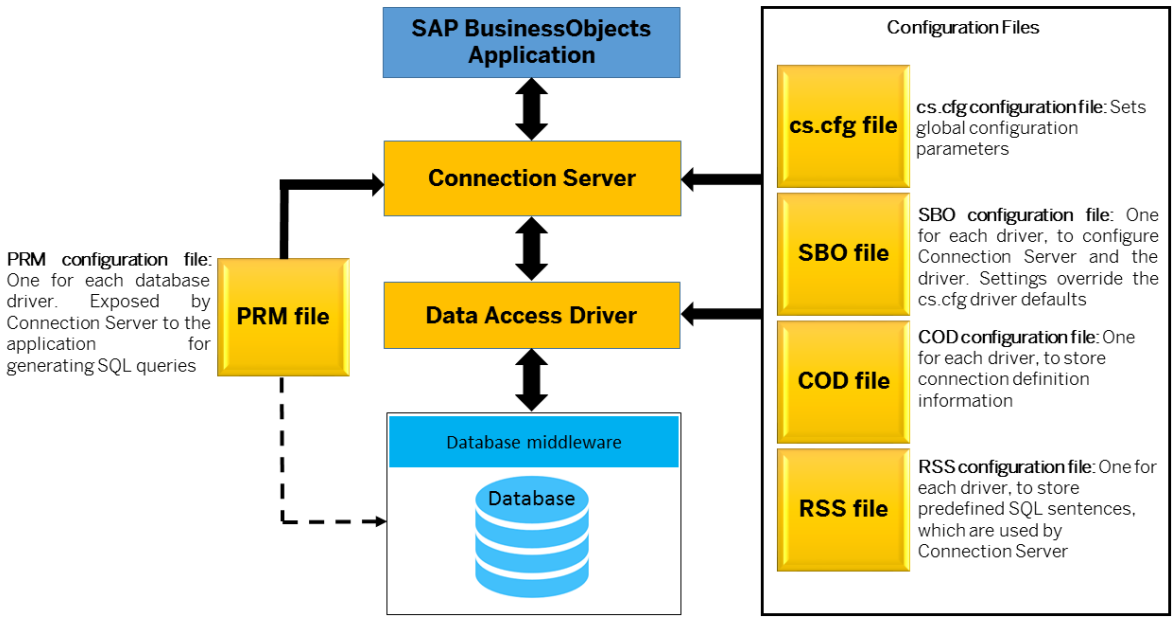
To understand the guide better, you need:

- Knowledge on Java technology
- Experience with developing and testing Java applications
- An understanding of SAP BusinessObjects applications that use Connection Server
- Experience with Connection Server configurations
- Knowledge on database functionality intended for custom driver

3 Data Access Driver Development System

3.1 System Architecture

The diagram below details where Data Access Driver fits into SAP BusinessObjects BI Platform configuration.



Data Access Drivers

Data Access Driver is a software component that acts as an interface between Connection Server and database middleware.

A data access driver must be able to perform the following tasks:

- Establish a connection with the database and manage the connection.
- Retrieve database metadata.
- Receive data requests from Connection Server, for example SQL queries
- Forward queries to the database for processing.
- Receive the requested data from the database and return the data in the correct format to Connection Server.

Connection Server

Connection Server is the data access software that manages the connection between an SAP BusinessObjects application and a data source. Connection Server allows applications such as Universe Design Tool, Information Designer Tool and SAP BusinessObjects Web Intelligence to connect to and run queries against a data source.

Configuration Files

Configuration files define parameters to configure a connection between the following systems:

- The application and Connection Server
- The application and the Data Access Driver
- Connection Server and the Data Access Driver

3.2 Working with Data Access Driver

The Driver Development Kit (DDK) API enables you to develop Data Access Drivers and use them with Connection Server.

You can provide access to the data sources of SAP BusinessObjects applications even if there is no driver available. For example, you can create connections to non-relational data sources such as XML data files and web services.

In addition, you can use the DDK API to provide database independent connectivity between SAP BusinessObjects applications and a wide range of databases, such as:

- SQL-oriented databases
- XML-SQL databases
- Tabular data sources such as spreadsheets or flat Comma Separated Value (CSV) files

SAP BusinessObjects applications generates SQL queries that are applied to a data source. Depending on the database middleware, the driver needs to parse the generated queries and convert them into a form that the database can process. The DDK API package comes with a sample that provides classes and methods to develop an SQL parser for your Data Access Driver, such as:

- Open drivers
- Configuration files
- Connection pool
- New connection wizard

3.2.1 Open Drivers

An Open driver operates in the same way as the standard Data Access Drivers delivered with SAP BusinessObjects Business Intelligence Platform.

An Open driver does the following:

- Ensures that the SQL query has the correct syntax and format for the data source.
- Applies the SQL query to the data source.
- Returns the query-generated data to Connection Server so that it is available for your application.

3.2.2 Setting up the Driver Configuration Files

To customize a driver, you must create the following configuration files for the driver:

- PRM configuration files
- COD configuration files
- SBO configuration files
- RSS configuration files

3.2.2.1 PRM Configuration Files

Each Data Access Driver is associated with a PRM configuration file, and the files control how an application generates SQL queries depending on the database software capabilities. Applications such as Information Design Tool use PRM files.

SAP BusinessObjects applications use databases as a source of data, and PRM files provides parameters that describe the capabilities of these databases.

Based on the connection and database, PRM files allow database-dependent factors to control SQL queries that you use in the universe. In addition, you can configure some of the database parameters within a universe. These configurations override PRM file settings consequently.

PRM files are located in the following folders:

- `<connectionserver-install-dir>\connectionServer\<RDBMS>` directories, where <RDBMS> is the network layer or middleware name.
- `<connectionserver-install-dir>\connectionServer\<RDBMS>\extensions\qt` directories. These PRM files are called extended files.

For more information on parameters of extended files, see the SQL and MDX reference chapter in the *Information Design Tool User Guide*.

3.2.2.2 COD Configuration Files

The COD file defines the screens and fields that the *New Connection* wizard displays when you select the driver to configure a connection. You have to set the sections and entries of the COD file to configure the New Connection wizard.

For each Open driver, the COD file must be available in the same location as the corresponding SBO file.

3.2.2.3 SBO Configuration Files

The SBO file defines the available drivers to use with Connection Server, and specifies default parameters for each driver. The file must contain a section for each available driver that uses DDK to develop. You also have to edit the SBO file to register the driver sample so that it is available for connections.

For more information on SBO files, refer to the *Data Access Guide*.

3.2.2.4 RSS Configuration Files

Each Data Access Driver has an RSS file which stores predefined SQL sentences used by the Connection Server.

For each Open driver, the COD file must be available in the same location as the corresponding SBO file.

3.2.2.5 Creating Configuration Files

You have to create the configuration files that your Open driver needs to establish a connection.

To create the configuration files, perform the following steps:

1. Go to the `<connectionserver-install-dir>\connectionServer\DDK\examples\open` directory.
2. Copy the `open_sample.prm` and `open_sampleen.cod` files and paste the into the `<connectionserver-install-dir>\connectionServer\DDK\examples\open` directory.
3. Rename the files to `<mydriver>.prm` and `<mydriver>en.cod` files.

   > **Note**
   >
   > You need to ensure there are localized PRM and COD files corresponding to each language on the machine where the driver is to be used. For example in a Japanese environment, the connection wizard uses the `<mydriver>ja.cod` file.

4. Edit the `open_sample.sbo` file as follows:
   1. Copy the following lines in the `DataBases` section:

      ```
      <DataBase Active="Yes" Name="DDK Sample Text Files">
      ```

```
 <Class
JARFile="dbd_open_sample">com.businessobjects.connectionserver.datasources.
opensample.CSVDriver</Class>
 <Parameter Name="Extensions">csv,open</Parameter>
 <Parameter Name="Description File">open_sample</Parameter>
 <Parameter Name="SQL Parameter File">open_sample</Parameter>
 <Parameter Name="Max Rows Available">Yes</Parameter>
</DataBase>
```

The `DataBases` section contains a `DataBase` entry for each available driver. The `DataBase` element defines that the Text Files Data Access Driver is active and defines the data source name.

2. Edit the `DataBase` element as follows:

```
<DataBase Active="Yes" Name="My Database">
 <Class
JARFile="dbd_mydriver">com.businessobjects.connectionserver.datasources.myc
lass.CSVDriver</Class>
 <Parameter Name="Extensions">mydriver,open</Parameter>
 <Parameter Name="Description File">mydriver</Parameter>
 <Parameter Name="SQL Parameter File">mydriver</Parameter>
 <Parameter Name="Max Rows Available">Yes</Parameter>
</DataBase>
```

The `Class` element sets the class that the Connection Server loads to initialize the driver as available.

3. Set the `Max Rows Available` parameter value according to what your driver can support.

5. Save the `open_sample.sbo` file.

You have created the configuration files that your Open driver needs to establish a connection.

3.2.3 Configuring Drivers

3.2.3.1 Driver Development Kit Configuration Interfaces

The DDK API provides interfaces to handle the configuration settings of Data Access Driver described in the sections below.

Context Interface

The `Context` interface helps you to manage configuration information of Data Access Drivers. This provides a series of methods to retrieve information from configuration files (PRM, SBO) and other information such as user credentials and dynamic configuration properties.

DriverConfiguration Interface

The `DriverConfiguration` interface provides default settings as defined in the SBO file of the driver. These settings cannot be changed for the duration of a Data Access Driver session.

If you develop a driver that requires additional settings, you need to update the SBO file to add `DriverConfiguration` settings. This enables you to extend the driver functionality.

PropertySet Interface

The `PropertySet` interface provides dynamic driver configuration properties.

Properties can be either read-only or modified by the driver, that is, in your code, you can change a `PropertySet` configuration value depending on what happens during processing.

DbParameterSet Interface

`DbParameterSet` interface provides settings as defined in the PRM file of the driver. These settings cannot be changed for the duration of a Data Access Driver session.

If you develop a driver that requires additional settings, you can update the PRM file to add DbParameterSet settings or by creating a new one. This enables you to extend the driver functionality.

> **Note**
>
> Some settings in the PropertySet interface are the same as those in the DriverConfiguration interface, and some are additional. For more information, refer to the *Driver Development Kit Javadocs*.

3.2.3.2 Retrieving Driver Configuration Settings

Use the `getconfiguration` defined in the `Context` interface to retrieve `DriverConfiguration` interface settings. Once you have retrieved the settings, use the following methods to retrieve data:

- `boolean getBoolean(String key)` to return a boolean value
- `int getInteger(String key)` to return an integer value
- `Object getObject(String key)` to return an object
- `String getString(String key)` to return a string

> **Note**
>
> You can set new values for `DriverConfiguration` interface settings by editing the SBO file and re-starting a driver session.
>
> You can use the `update(String connectionDefinition)` method that is also defined in the `Context` interface to help the environment generate valid contexts and create connections. The method updates the context credentials and properties with the given connection definition.

3.2.3.3 Setting and Retrieving Connection Properties

You can use the `getProperties` method defined in the `Context` interface and `PropertySet` interface to set and retrieve driver property values. The method used depends on the property data type. Use the `setIntProperty` and `getIntProperty` methods for example to set and retrieve integer properties.

> Example
>
> When creating the connection, the driver sample uses `PropertySet` interface methods to set the Max Rows value to the retrieved value.
>
> ```
> setMaxRows (getContext ().getProperties ()
> .getIntProperty (PropertySet.MAX_ROWS));
> ```
>
> The following code sets the identifier case property to case sensitive:
>
> ```
> getContext ().getProperties ().setStringProperty
> (PropertySet.IDENTIFIER_CASE, "SensitiveCase");
> ```

3.2.3.4 Setting Default PropertySet Values

When an SAP BusinessObjects application establishes a connection to a data source, ConnectionServer sets each property in the `PropertySet` interface to a default value. A property default value can be set in one of the following ways:

- ConnectionServer retrieves the value from the SBO configuration file, or from the `cs.cfg` file if the values are not defined in the SBO file.
- ConnectionServer retrieves the value from the configuration files, and retrieves the value from the database middleware if the values are not defined in the configuration files.
- ConnectionServer retrieves the value from the database middleware without checking the configuration files first.
- ConnectionServer sets the property to a specific value

For more information about the `PropertySet` interface, refer the DDK Javadocs.s

3.2.3.5 Retrieving Database Parameter Settings

You can use the `getDbParameters` method defined in the `Context` interface to retrieve the `DbParameterSet` interface settings from the PRM file of your Data Access Driver. After retrieving the `DbParameterSet` interface settings, use the methods defined in the `DbParameterSet` interface to retrieve specific database capability information such as domains, function names and SQL syntax.

For more information about PRM files, refer to the Data Access Guide.

3.2.4 Using Connection Pool

3.2.4.1 About the Connection Pool

A driver opens a connection to the database in order to access data. The following methods are used to connect to the database:

- Every time the ConnectionServer requires information, the Data Access Driver opens a connection to the database, retrieves the data, and closes the connection.
- The Connection Server keeps the available connections open and maintains their details in a connection pool.
  Each time the ConnectionServer requires information from the data source, the Data Access Driver checks the connection pool to see if it contains an unused and suitable connection. If a suitable connection is available, the ConnectionServer consumes it.

3.2.4.2 Implementing Connection pool

To manage connections of a pool, the driver implements the following DDK interfaces:

- The `<Driver_Name> connection` class implements the `Connection` interface and provides methods to manage connections in a connection pool.
- The `ConnectionPool` interface provides methods that the driver uses to manage a connection pool.

Connection interface

The Connection interface includes methods with the following functionalities:

- The connection pool uses computed unique key to identify a connection profile.
- Set and get the length of time that connections are to remain active.
- Close connections.

ConnectionPool interface

The `ConnectionPool` interface includes methods with the following functionalities:

- Acquire details of connections in the pool.
- Add and release connections in the pool.
- Lock connections that are in use in order to enable exclusive actions on the connections.
  The `ConnectionPool` interface methods releases the lock when the connection returns to the pool. The Lock functionality ensures that only one commit or rollback is performed on the connection.

3.2.4.3 Retrieving a Connection from the Pool

The driver sample calls the following sequence of methods when an SAP BusinessObjects application requests a connection from a connection pool. This happens when the `createConnection` method is called.

1. The `computeKey` method computes a key to identify the connection profile with user credentials (user login and password).
   `Object key = CSVConnection.computeKey(filename);`
2. The `getIntProperty` method retrieves the `POOL_TIME` parameter value.
   `int poolTime = Context().getProperties().getIntProperty (PropertySet.POOL_TIME);`

   > **Note**
   >
   > The `POOL_TIME` setting determines how long an unused connection is kept in the connection pool. It is defined by the universe creator when creating the connection from the New Connection wizard.

3. The `getBooleanProperty` method helps you to determine whether the data source can share connections.

   ```
   boolean sharedConnection = Context().getProperties()
   .getBooleanProperty(PropertySet.SHARED_CONNECTION);
   ```

4. The `getConnectionPool` method retrieves a connection from the pool if one is operating.

   ```
   if (poolTime != 0) {
   connection = (CSVConnection)
   getConnectionPool().acquire(key, sharedConnection);
   if (connection != null) {
   connection.setDuration(poolTime);
   }
   }
   ```

   The `PoolTime0` value indicates that the connection pool cannot be used for this kind of connection.
5. The `CSVConnection` method creates a new connection if no connection is available in the pool.

   ```
   if (connection == null) {
   connection = new CSVConnection(key, poolTime, new CSVFile (filename,
   separator, properties));
   }
   ```

3.2.4.4 Releasing a Connection from the Pool

The driver sample closes a connection by releasing it and making it available for further operation. This happens when the `disconnect` method is called.

The `releaseConnection` method releases a connection to the pool if the `poolTime` value is 0. If not, the `close` method closes the connection.

```
private void releaseConnection () throws DDKException
{
try
{
if (connection != null)
```

```
{
if (getContext ().getProperties ().getIntProperty (PropertySet.POOL_TIME) != 0)
{
getConnectionPool ().release (connection);
}
else
connection.close ();
Using a connection pool
connection = null;
}
}
...
}
```

> **Note**
>
> - The driver must release the Shareable connections back to the pool if they are no longer required.
> - The connection pool needs to close the shareable connections.
> - The driver should not close a shareable connection explicitly, as this could cause a live connection to a data source to be lost.

3.2.5 Configuring New Connection Wizard

3.2.5.1 About the New Connection Wizard

The New Connection wizard is a part of the Universe Design Tool and the Information Design Tool. You can use the New Connection wizard to create a connection to a data source and provide data to your universe. In addition, the New Connection wizard collects the user-supplied information used to configure a connection.

You set up the COD configuration file of your Data Access Driver to define the input fields that the New Connection wizard displays. The file must be located in the same directory along with the corresponding SBO file.

The New Connection wizard collects information. You can implement DDK API methods in your driver code to retrieve collected information. The Connection wizard uses this information to configure and create a connection.

3.2.5.2 What the New Connection wizard does

The New Connection wizard first reads the `DataBases` section of the SBO file, and displays available drivers in the *Database Middleware Selection* window.

The `Family` parameter sets the section in the New Connection wizard under which the driver appears. The `Description` file parameter defines the name of the COD file that specifies the configuration information that the driver needs.

For more information on SBO parameters, refer to the Data Access Guide .

4 Setting up the DDK Environment

The `<Driver_Name>` class implements the `DBEnvironment` interface to handle specific driver and configuration information. The driver calls the following sequence of methods:

1. `inquiry` method retrieves an XML string built from the `Inquiry` parameter value of the <Driver_Name> COD configuration file.

   ```
   public String inquiry (String id, Map params) throws DDKException
   ```

   The XML string has the following structure:

   ```
   <Inquiry ID = "InquiryID">
   <Entry>...</Entry>
   <Entry>...</Entry>
   <Entry>...</Entry>
   </Inquiry>
   ```

   > **Note**
   >
   > Execution of the above step depends on the conditions set in the COD file.

2. The `validate`
   method validates the definition of the connection.

   ```
   public void validate (ConnectionDefinition definition) throws DDKException
   ```

3. The `newDBEnvironmentInstance` method creates an instance of the `<Driver_Name>` Environment class.

   ```
   public static DBEnvironment newDBEnvironmentInstance (String networkLayer,
   String dbms, Context context)
   throws DDKException
   {
   return new CSVEnvironment (networkLayer, dbms, context);
   }
   ```

5 Creating Connections

The`<Driver_Name>` class implements the `OpenDriver` interface. This interface extends the `DBDriver` interface from the `com.businessobjects.connectionserver.datasources.ddk` package of the DDK API.

Connection Server creates the connection by calling the following sequence of methods:

1. The `initDriver` method initializes the driver.

```
public static void initDriver (DriverConfiguration config,
String networkLayer, String dataBase)
```

2. The `newDBDriverInstance` method creates and returns an instance of the driver.

```
public static DBDriver newDBDriverInstance (Context context, ConnectionPool
pool)
throws DDKException
{
return new <name -of-the-driver> (context, pool);
}
```

   The `<Driver_Name>` constructor method does the following:

   1. The SAP BusinessObjects application generates the query, and the constructor allocates a parser to parse the query.

```
parser = ParserFactory.newParser ();
```

   2. Initializes the driver from its configuration:

```
DriverConfiguration config = context.getConfiguration ();
{
...
try
{
setArrayFetchSupported (config.getBoolean ("Array Fetch Available"));
}
...
}
```

3. The `connect` method connects the driver to the data source.

```
public void connect () throws DDKException
{
createConnection ();
setConnected (true);
...
}
```

   The `createConnection` method retrieves information such as credentials from the New Connection wizard and sets up the connection.

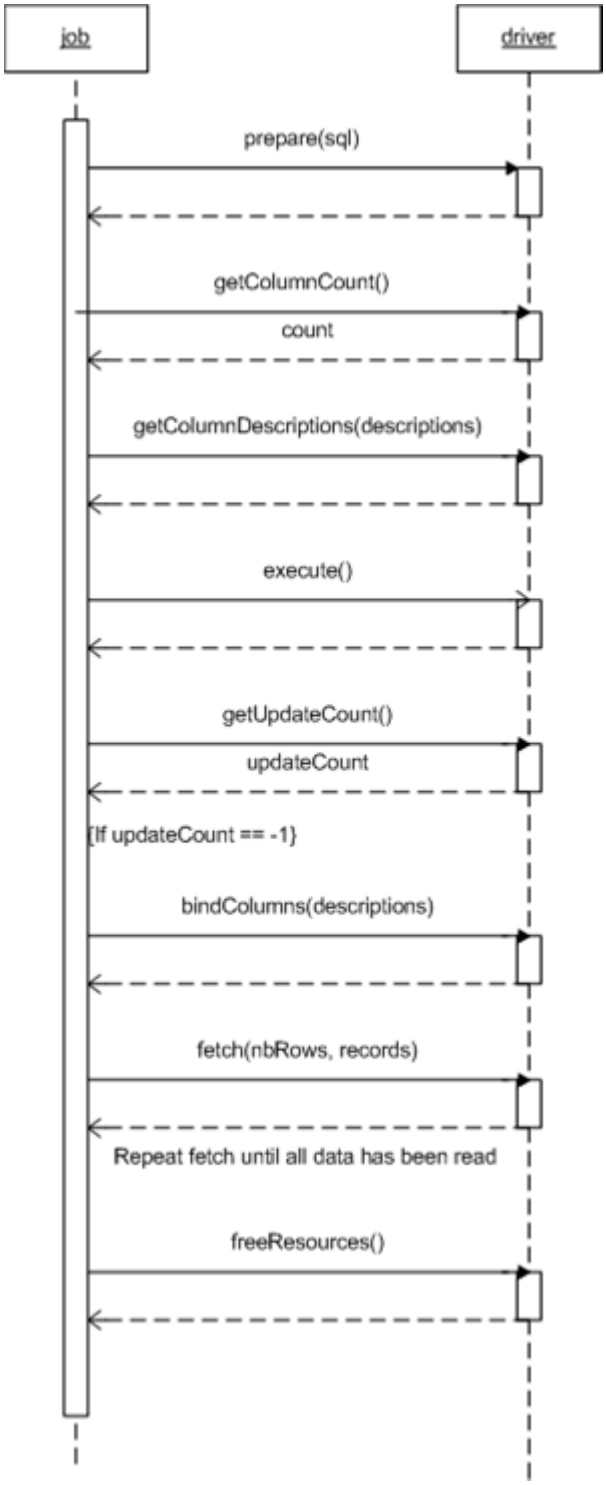
```
private void createConnection () throws DDKException
{
try
{
String filename = (String)getContext ().getCredentials ()
```

```
.get ("DATASOURCE");
String separator = (String)getContext ().getCredentials ()
.get ("SEPARATOR");
...
}
```

> **Note**
>
> The `createConnection` method allows you to implement a connection pool.

6 Preparing for Data Request

Given below is the sequence diagram of execution of a query handled by the driver.



After initializing the driver and creating the connection, Connection Server is ready to process requests to metadata and data. It calls the following sequence of driver methods:

1. The data source executes the request from the `prepare` method.

```
public void prepare (String query) throws DDKException
{
try
```

```
{
statement = parser.parseQuery (query);
validateStatement ();
}
...
}
```

- If the data source does not support the SQL language, the driver parses the query.
- If the data source supports SQL, the driver can directly submit the query to the data source for validation.

2. The `getColumnCount` method returns the number of columns of the `ResultSet` object.

```
public int getColumnCount () throws DDKException
{
return statement.getColumnNames ().size ();
}
```

3. The `getColumnDescriptions` method returns the column's description of the `ResultSet` object. `getColumnDescriptions` fills the descriptionSet parameter with the details of the columns returned by the query.

```
public void getColumnDescriptions (DescriptionSet descriptionSet)
```

4. The `BindColumns` method defines a binding between the datasource columns and driver buffers used to retrieve data. `BindColumns` uses the `DataManager` class.

```
public void bindColumns (DescriptionSet descriptionSet) throws DDKException
```

7 Requesting Data from the Data Source

After retrieving the result metadata, the Connection Server is ready to execute the query and calls the following sequence of methods:

1. The `execute` method applies the request to the data source.

   ```
   public void execute () throws DDKException
   ```

2. The `fetch` method fills a set of records with data retrieved from the data source.

   ```
   void fetch (int requestedSize, RecordSet recordSet) throws DDKException
   ```

   `requestedSize` is the maximum number of records to fetch from the data source. `recordSet` is the output set of record to be filled by the driver.

3. The `freeResources` method releases the resources used when fetching the data.

   ```
   public void freeResources () throws DDKException
   {
   columnIndexes = null;
   rowsFetched = 0;
   Creating an Open driver
   dataIterator = null;
   fieldBinding = null;
   metadataRequested
   ```

   Example

   The code snippet fills `sizeToFetch` number of records, or the maximum number of rows if it is less than `sizeToFetch`.

   ```
   for (int i = 0; i < sizeToFetch && (hasNext
   = dataIterator.hasNext ()); i++)
   {
   String[] rowData = (String[])dataIterator.next ();
   Record record = recordSet.nextRecord ();
   for (int j = 0; j < columnIndexes.length; j++)
   {
   String cell = rowData[columnIndexes[j]];
   try
   {
   fieldBinding[j].fetch (record, cell);
   }
   catch (NumberFormatException exception)
   {
   throw getContext ().getErrors ().getError
   (ERROR_BAD_BINDING + exception.getMessage ());
   }
   }
   }
   ```

8 Closing the Connection

The connection is closed when the driver no longer consumes the connection. Connection Server calls the following sequence of methods to close the connection:

1. The `disconnect` method releases the connection.

   ```
   public void disconnect () throws DDKException
   {
   releaseConnection ();
   setConnected (false);
   }
   ```

   > **Note**
   >
   > The connection pool implements the `releaseConnection` method. Without the connection pool, it calls the `close()` method of the `<Driver_Name> Connection` object.

2. `close` method of `<Driver_Name> Connection` class releases the `ResultSet` object. The driver remains available for further operations.

   ```
   public void close ()
   {
   }
   ```

3. `The finiDriver` method ends the driver process.`public static void finiDriver ()`

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

© 2024 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see https://www.sap.com/about/legal/trademark.html for additional trademark information and notices.

THE BEST RUN SAP