| | [**Overview**](http://docs.google.com/overview-summary.html) | **Package** | Class | [**Use**](http://docs.google.com/package-use.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV PACKAGE**](http://docs.google.com/javax/sound/sampled/spi/package-summary.html)   [**NEXT PACKAGE**](http://docs.google.com/javax/sql/rowset/package-summary.html) | [**FRAMES**](http://docs.google.com/index.html?javax/sql/package-summary.html)    [**NO FRAMES**](http://docs.google.com/package-summary.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |

## Package javax.sql

Provides the API for server side data source access and processing from the JavaTM programming language.

**See:**

[**Description**](#3znysh7)

| **Interface Summary** | |
| --- | --- |
| [**CommonDataSource**](http://docs.google.com/javax/sql/CommonDataSource.html) | Interface that defines the methods which are common between DataSource, XADataSource and ConnectionPoolDataSource. |
| [**ConnectionEventListener**](http://docs.google.com/javax/sql/ConnectionEventListener.html) | An object that registers to be notified of events generated by a PooledConnection object. |
| [**ConnectionPoolDataSource**](http://docs.google.com/javax/sql/ConnectionPoolDataSource.html) | A factory for PooledConnection objects. |
| [**DataSource**](http://docs.google.com/javax/sql/DataSource.html) | A factory for connections to the physical data source that this DataSource object represents. |
| [**PooledConnection**](http://docs.google.com/javax/sql/PooledConnection.html) | An object that provides hooks for connection pool management. |
| [**RowSet**](http://docs.google.com/javax/sql/RowSet.html) | The interface that adds support to the JDBC API for the JavaBeansTM component model. |
| [**RowSetInternal**](http://docs.google.com/javax/sql/RowSetInternal.html) | The interface that a RowSet object implements in order to present itself to a RowSetReader or RowSetWriter object. |
| [**RowSetListener**](http://docs.google.com/javax/sql/RowSetListener.html) | An interface that must be implemented by a component that wants to be notified when a significant event happens in the life of a RowSet object. |
| [**RowSetMetaData**](http://docs.google.com/javax/sql/RowSetMetaData.html) | An object that contains information about the columns in a RowSet object. |
| [**RowSetReader**](http://docs.google.com/javax/sql/RowSetReader.html) | The facility that a disconnected RowSet object calls on to populate itself with rows of data. |
| [**RowSetWriter**](http://docs.google.com/javax/sql/RowSetWriter.html) | An object that implements the RowSetWriter interface, called a *writer*. |
| [**StatementEventListener**](http://docs.google.com/javax/sql/StatementEventListener.html) | An object that registers to be notified of events that occur on PreparedStatements that are in the Statement pool. |
| [**XAConnection**](http://docs.google.com/javax/sql/XAConnection.html) | An object that provides support for distributed transactions. |
| [**XADataSource**](http://docs.google.com/javax/sql/XADataSource.html) | A factory for XAConnection objects that is used internally. |

| **Class Summary** | |
| --- | --- |
| [**ConnectionEvent**](http://docs.google.com/javax/sql/ConnectionEvent.html) | An Event object that provides information about the source of a connection-related event. |
| [**RowSetEvent**](http://docs.google.com/javax/sql/RowSetEvent.html) | An Event object generated when an event occurs to a RowSet object. |
| [**StatementEvent**](http://docs.google.com/javax/sql/StatementEvent.html) | A StatementEvent is sent to all StatementEventListeners which were registered with a PooledConnection. |

## Package javax.sql Description

Provides the API for server side data source access and processing from the JavaTM programming language. This package supplements the java.sql package and, as of the version 1.4 release, is included in the Java Platform, Standard Edition (Java SETM). It remains an essential part of the Java Platform, Enterprise Edition (Java EETM).

The javax.sql package provides for the following:

1. The DataSource interface as an alternative to the DriverManager for establishing a connection with a data source
2. Connection pooling and Statement pooling
3. Distributed transactions
4. Rowsets

Applications use the DataSource and RowSet APIs directly, but the connection pooling and distributed transaction APIs are used internally by the middle-tier infrastructure.

## Using a DataSource Object to Make a Connection

The javax.sql package provides the preferred way to make a connection with a data source. The DriverManager class, the original mechanism, is still valid, and code using it will continue to run. However, the newer DataSource mechanism is preferred because it offers many advantages over the DriverManager mechanism.

These are the main advantages of using a DataSource object to make a connection:

* Changes can be made to a data source's properties, which means that it is not necessary to make changes in application code when something about the data source or driver changes.
* Connection and Statement pooling and distributed transactions are available through a DataSource object that is implemented to work with the middle-tier infrastructure. Connections made through the DriverManager do not have connection and statement pooling or distributed transaction capabilities.

Driver vendors provide DataSource implementations. A particular DataSource object represents a particular physical data source, and each connection the DataSource object creates is a connection to that physical data source.

A logical name for the data source is registered with a naming service that uses the Java Naming and Directory InterfaceTM (JNDI) API, usually by a system administrator or someone performing the duties of a system administrator. An application can retrieve the DataSource object it wants by doing a lookup on the logical name that has been registered for it. The application can then use the DataSource object to create a connection to the physical data source it represents.

A DataSource object can be implemented to work with the middle tier infrastructure so that the connections it produces will be pooled for reuse. An application that uses such a DataSource implementation will automatically get a connection that participates in connection pooling. A DataSource object can also be implemented to work with the middle tier infrastructure so that the connections it produces can be used for distributed transactions without any special coding.

## Connection Pooling and Statement Pooling

Connections made via a DataSource object that is implemented to work with a middle tier connection pool manager will participate in connection pooling. This can improve performance dramatically because creating new connections is very expensive. Connection pooling allows a connection to be used and reused, thus cutting down substantially on the number of new connections that need to be created.

Connection pooling is totally transparent. It is done automatically in the middle tier of a Java EE configuration, so from an application's viewpoint, no change in code is required. An application simply uses the DataSource.getConnection method to get the pooled connection and uses it the same way it uses any Connection object.

The classes and interfaces used for connection pooling are:

* ConnectionPoolDataSource
* PooledConnection
* ConnectionEvent
* ConnectionEventListener
* StatementEvent
* StatementEventListener

The connection pool manager, a facility in the middle tier of a three-tier architecture, uses these classes and interfaces behind the scenes. When a ConnectionPoolDataSource object is called on to create a PooledConnection object, the connection pool manager will register as a ConnectionEventListener object with the new PooledConnection object. When the connection is closed or there is an error, the connection pool manager (being a listener) gets a notification that includes a ConnectionEvent object.

If the connection pool manager supports Statement pooling, for PreparedStatements, which can be determined by invoking the method DatabaseMetaData.supportsStatementPooling, the connection pool manager will register as a StatementEventListener object with the new PooledConnection object. When the PreparedStatement is closed or there is an error, the connection pool manager (being a listener) gets a notification that includes a StatementEvent object.

## Distributed Transactions

As with pooled connections, connections made via a DataSource object that is implemented to work with the middle tier infrastructure may participate in distributed transactions. This gives an application the ability to involve data sources on multiple servers in a single transaction.

The classes and interfaces used for distributed transactions are:

* XADataSource
* XAConnection

These interfaces are used by the transaction manager; an application does not use them directly.

The XAConnection interface is derived from the PooledConnection interface, so what applies to a pooled connection also applies to a connection that is part of a distributed transaction. A transaction manager in the middle tier handles everything transparently. The only change in application code is that an application cannot do anything that would interfere with the transaction manager's handling of the transaction. Specifically, an application cannot call the methods Connection.commit or Connection.rollback, and it cannot set the connection to be in auto-commit mode (that is, it cannot call Connection.setAutoCommit(true)).

An application does not need to do anything special to participate in a distributed transaction. It simply creates connections to the data sources it wants to use via the DataSource.getConnection method, just as it normally does. The transaction manager manages the transaction behind the scenes. The XADataSource interface creates XAConnection objects, and each XAConnection object creates an XAResource object that the transaction manager uses to manage the connection.

## Rowsets

The RowSet interface works with various other classes and interfaces behind the scenes. These can be grouped into three categories.

1. Event Notification
   * RowSetListener  
     A RowSet object is a JavaBeansTM component because it has properties and participates in the JavaBeans event notification mechanism. The RowSetListener interface is implemented by a component that wants to be notified about events that occur to a particular RowSet object. Such a component registers itself as a listener with a rowset via the RowSet.addRowSetListener method.  
     When the RowSet object changes one of its rows, changes all of it rows, or moves its cursor, it also notifies each listener that is registered with it. The listener reacts by carrying out its implementation of the notification method called on it.
   * RowSetEvent  
     As part of its internal notification process, a RowSet object creates an instance of RowSetEvent and passes it to the listener. The listener can use this RowSetEvent object to find out which rowset had the event.
2. Metadata
   * RowSetMetaData  
     This interface, derived from the ResultSetMetaData interface, provides information about the columns in a RowSet object. An application can use RowSetMetaData methods to find out how many columns the rowset contains and what kind of data each column can contain.  
     The RowSetMetaData interface provides methods for setting the information about columns, but an application would not normally use these methods. When an application calls the RowSet method execute, the RowSet object will contain a new set of rows, and its RowSetMetaData object will have been internally updated to contain information about the new columns.
3. The Reader/Writer Facility  
   A RowSet object that implements the RowSetInternal interface can call on the RowSetReader object associated with it to populate itself with data. It can also call on the RowSetWriter object associated with it to write any changes to its rows back to the data source from which it originally got the rows. A rowset that remains connected to its data source does not need to use a reader and writer because it can simply operate on the data source directly.
   * RowSetInternal  
     By implementing the RowSetInternal interface, a RowSet object gets access to its internal state and is able to call on its reader and writer. A rowset keeps track of the values in its current rows and of the values that immediately preceded the current ones, referred to as the *original* values. A rowset also keeps track of (1) the parameters that have been set for its command and (2) the connection that was passed to it, if any. A rowset uses the RowSetInternal methods behind the scenes to get access to this information. An application does not normally invoke these methods directly.
   * RowSetReader  
     A disconnected RowSet object that has implemented the RowSetInternal interface can call on its reader (the RowSetReader object associated with it) to populate it with data. When an application calls the RowSet.execute method, that method calls on the rowset's reader to do much of the work. Implementations can vary widely, but generally a reader makes a connection to the data source, reads data from the data source and populates the rowset with it, and closes the connection. A reader may also update the RowSetMetaData object for its rowset. The rowset's internal state is also updated, either by the reader or directly by the method RowSet.execute.
   * RowSetWriter  
     A disconnected RowSet object that has implemented the RowSetInternal interface can call on its writer (the RowSetWriter object associated with it) to write changes back to the underlying data source. Implementations may vary widely, but generally, a writer will do the following:  
     + Make a connection to the data source
     + Check to see whether there is a conflict, that is, whether a value that has been changed in the rowset has also been changed in the data source
     + Write the new values to the data source if there is no conflict
     + Close the connection

The RowSet interface may be implemented in any number of ways, and anyone may write an implementation. Developers are encouraged to use their imaginations in coming up with new ways to use rowsets.

**IMPORTANT NOTE:** Code that uses API marked "Since 1.6" must be run using a JDBC technology driver that implements the JDBC 4.0 API. You must check your driver documentation to be sure that it implements the particular features you want to use.

## Package Specification

* [Specification of the JDBC 4.0 API](http://java.sun.com/products/jdbc/download.html)

## Related Documentation

The Java Series book published by Addison-Wesley Longman provides detailed information about the classes and interfaces in the javax.sql package:

* [*JDBCTM API Tutorial and Reference, Third Edition:*](http://java.sun.com/docs/books/jdbc)

**Since:** 1.4

| | [**Overview**](http://docs.google.com/overview-summary.html) | **Package** | Class | [**Use**](http://docs.google.com/package-use.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV PACKAGE**](http://docs.google.com/javax/sound/sampled/spi/package-summary.html)   [**NEXT PACKAGE**](http://docs.google.com/javax/sql/rowset/package-summary.html) | [**FRAMES**](http://docs.google.com/index.html?javax/sql/package-summary.html)    [**NO FRAMES**](http://docs.google.com/package-summary.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).