InterSystems Implementation Reference for Java Third Party APIs

Version 2019.1
2019-05-13

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

InterSystems Implementation Reference for Java Third Party APIs
InterSystems IRIS Data Platform Version 2019.1 2019-05-13

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

About This Book

This book provides technical information and installation procedures (if necessary) for the InterSystems JDBC driver, Spark connector, and Hibernate dialect.

The following topics are discussed in this book:

- Introduction — provides an overview of the supported frameworks.
- JDBC Driver Support — provides full details about JDBC supported features and extensions.
- Apache Spark Support — lists basic features and links to information on major extensions.
- Hibernate Support — provides installation instructions and discusses supported features.

There is also a detailed Table of Contents.

1

Introduction

InterSystems IRIS™ provides the following Java third party connectivity options:

The InterSystems JDBC Driver

The InterSystems JDBC Driver is a fully compliant type 4 implementation of the JDBC standard.

See JDBC Driver Support for detailed information on JDBC driver compliance and enhancements, including the level of support for all optional features and a list of all InterSystems IRIS-specific additional features.

Also see the following related documentation:

- *Using Java with the InterSystems JDBC Driver* provides a full description of features and usage.

The InterSystems Spark Connector

The InterSystems Spark Connector is an implementation of the Data Source API for Apache Spark that allows the Spark data processing engine to make optimal use of the InterSystems IRIS Data Platform and its distributed data capabilities.

See Apache Spark Support for details.

Also see the following related documentation:

- See *Using the InterSystems Spark Connector* for full details on installation, configuration, and usage.

The InterSystems Hibernate Dialect

The InterSystems Hibernate Dialect is an implementation of the Hibernate dialect interface. Since every vendor's implementation of SQL is slightly different, the dialect interface allows vendors to create custom Hibernate mappings for a specific database. Vendor-provided dialect implementations are distributed as part of Hibernate.

See Hibernate Support for details.

2

JDBC Driver Support

The InterSystems IRIS™ JDBC Driver is a fully compliant type 4 implementation of the JDBC 4.2 standard. This chapter lists all classes and interfaces of the JDBC 4.2 API, indicates the level of support for each one, and describes all InterSystems-specific features. The following topics are discussed:

- JDBC and the InterSystems JDBC Driver — provides an overview and resource links for the JDBC driver.
- JDBC Driver Compliance — lists all classes and interfaces specified by the JDBC standard, and indicates the current level of support.
- Variants and Unsupported Optional Methods — provides details on classes that include permitted variances from the standard.
- InterSystems Enhancements and Extensions — lists and discusses InterSystems extensions to the standard JDBC API.

2.1 JDBC and the InterSystems JDBC Driver

The Java JDBC API is the industry standard for vendor-neutral database connectivity. It provides a reliable way for Java applications to connect to data sources on any supported platform and to query or perform operations on them with SQL.

InterSystems JDBC is implemented in a type 4 driver to deliver the highest possible performance. *Type 4* means that it is a direct-to-database pure Java driver, installed inside the client JVM and requiring no external software support. It is fully compliant with the JDBC 4.2 API specification, supporting all required interfaces and adhering to all JDBC 4.2 guidelines and requirements. InterSystems IRIS supports all features except SQL Exception handling enhancements, National Character Set conversions, and the XML data type.

See *Using Java JDBC with InterSystems IRIS* for a full description of API features and usage. That book also provides an overview of all InterSystems IRIS Java technologies enabled by the JDBC driver (see "InterSystems Java Connectivity Options").

2.1.1 Installation and Configuration

The InterSystems JDBC driver is included in the standard InterSystems IRIS installation package. No extra installation or setup procedures are required. See "Client-Server Configuration" in *Using Java JDBC with InterSystems IRIS* for information on client requirements and usage.

2.2 JDBC Driver Compliance

This section provides information on the level of support for each JDBC interface.

2.2.1 Required java.sql Interfaces

The following interfaces must be implemented. Some classes contain methods that are optional if the implementation depends on a feature that the database does not support. The *standard implementation* annotation indicates that the generic implementation of the class has been used without alteration:

- java.sql.CallableStatement — implemented with some permitted variances (see "CallableStatement: Unsupported Methods" and "CallableStatement getBinaryStream() Extension Method").
- java.sql.ClientInfoStatus — *standard implementation*.
- java.sql.Connection — implemented with some permitted variances (see Connection: Unsupported or Restricted Methods).
- java.sql.DatabaseMetaData — implemented with some permitted variances (see "DatabaseMetaData: Variant Methods").
- java.sql.Date — *standard implementation*.
- java.sql.Driver — implemented with some permitted variances (see "Driver: Unsupported Methods").
- java.sql.DriverManager — *standard implementation*.
- java.sql.DriverPropertyInfo — *standard implementation*.
- java.sql.ParameterMetaData — all methods fully supported.
- java.sql.PreparedStatement — implemented with some permitted variances (see "PreparedStatement: Unsupported Methods").
- java.sql.ResultSet — implemented with some permitted variances (see "ResultSet: Unsupported or Restricted Methods") .
- java.sql.ResultSetMetaData — all methods fully supported.
- java.sql.RowIdLifeTime — *standard implementation*.
- java.sql.SQLPermission — *standard implementation*.
- java.sql.Statement — implemented with some permitted variances (see "Statement: Unsupported or Restricted Methods").
- java.sql.Time — *standard implementation*.
- java.sql.Timestamp — *standard implementation*.
- java.sql.Types — *standard implementation*.
- java.sql.Wrapper — all methods fully supported.

2.2.2 Optional java.sql Interfaces

All optional java.sql interfaces are listed below. *Italicized* items are not implemented:

- *java.sql.Array*
- java.sql.Blob — all methods fully supported.

- java.sql.Clob — all methods fully supported.
- java.sql.NClob — all methods fully supported.
- *java.sql.Ref*
- java.sql.RowId — all methods fully supported.
- java.sql.Savepoint — all methods fully supported.
- *java.sql.SQLData*
- *java.sql.SQLInput*
- *java.sql.SQLOutput*
- *java.sql.SQLXML*
- *java.sql.Struct*

2.2.3 java.sql Exceptions

The InterSystems JDBC driver throws only the following exceptions:

- java.sql.BatchUpdateException
- java.sql.SQLException
- java.sql.SQLWarning

The following exceptions are listed here for completeness, but are not required and are never used:

- *DataTruncation*
- *SQLClientInfoException*
- *SQLDataException*
- *SQLFeatureNotSupportedException*
- *SQLIntegrityConstraintViolationException*
- *SQLInvalidAuthorizationSpecException*
- *SQLNonTransientConnectionException*
- *SQLNonTransientException*
- *SQLRecoverableException*
- *SQLSyntaxErrorException*
- *SQLTimeoutException*
- *SQLTransactionRollbackException*
- *SQLTransientConnectionException*
- *SQLTransientException*

2.2.4 Required javax.sql Interfaces

The following required interfaces are supported. The *standard implementation* annotation indicates that the generic implementation of the class has been used without alteration:

- javax.sql.ConnectionEvent — *standard implementation*.
- javax.sql.DataSource — implemented with enhancements and additional methods (see "DataSource Extensions and Enhancements" for details).
- javax.sql.RowSetEvent — *standard implementation*.
- javax.sql.StatementEvent — *standard implementation*.

2.2.5 Optional javax.sql Interfaces

All optional javax.sql interfaces are listed below. *Italicized* items are not implemented:

- *javax.sql.CommonDataSource* — not implemented. Use javax.sql.DataSource instead (see "DataSource Extensions and Enhancements" for related information).
- javax.sql.ConnectionEventListener — all methods fully supported.
- javax.sql.ConnectionPoolDataSource — implemented with variants and additional methods (see "ConnectionPoolDataSource Extensions and Enhancements" for details).
- javax.sql.PooledConnection — all methods fully supported.
- *javax.sql.Rowset*
- *javax.sql.RowSetInternal*
- *javax.sql.RowSetListener*
- *javax.sql.RowSetMetaData*
- *javax.sql.RowSetReader*
- *javax.sql.RowSetWriter*
- *javax.sql.StatementEventListener*
- *javax.sql.XAConnection*
- *javax.sql.XADataSource*

2.3 Variants and Unsupported Optional Methods

The following interfaces have optional methods that the InterSystems JDBC driver does not support, or methods implemented in a non-standard manner:

- CallableStatement: Unsupported Methods
- Connection: Unsupported or Restricted Methods
- DatabaseMetaData: Variant Methods
- Driver: Unsupported Methods
- PreparedStatement: Unsupported Methods
- ResultSet: Unsupported or Restricted Methods
- Statement: Unsupported or Restricted Methods

2.3.1 CallableStatement: Unsupported Methods

Unsupported Optional Methods

java.sql.CallableStatement does not support the following optional methods:

- **getArray()**

```
Array getArray(int i)
Array getArray(String parameterName)
```

- **getObject()**

```
Object getObject(int i, java.util.Map map)
Object getObject(String parameterName, java.util.Map map)
```

- **getRef()**

```
Ref getRef(int i)
Ref getRef(String parameterName)
```

- **getRowId()** and **setRowId()**

```
java.sql.RowId getRowId(int i)
java.sql.RowId getRowId(String parameterName)

void setRowId(String parameterName, java.sql.RowId x)
```

- **getURL()** and **setURL()**

```
java.net.URL getURL(int i)
java.net.URL getURL(String parameterName)

void setURL(String parameterName, java.net.URL val)
```

- **getSQLXML()** and **setSQLXML()**

```
java.sql.SQLXML getSQLXML(int parameterIndex)
java.sql.SQLXML getSQLXML(String parameterName)

void setSQLXML(String parameterName, java.sql.SQLXML xmlObject)
```

Note: The java.sql.CallableStatement class also has one InterSystems extension method, which is discussed elsewhere (see “CallableStatement getBinaryStream() Extension Method”).

2.3.2 Connection: Unsupported or Restricted Methods

Unsupported Optional Methods

java.sql.Connection does not support the following optional methods:

- **abort()**

```
void abort(Executor executor)
```

- **createArrayOf()**

```
java.sql.Array createArrayOf(String typeName, Object[] elements)
```

- **createBlob()**

```
Blob createBlob()
```

- **createClob()**

  ```
  Clob createClob()
  ```

- **createNClob()**

  ```
  java.sql.NClob createNClob()
  ```

- **createSQLXML**()

  ```
  java.sql.SQLXML createSQLXML()
  ```

- **createStruct()**

  ```
  java.sql.Struct createStruct(String typeName, Object[] attributes)
  ```

- **getTypeMap()**

  ```
  java.util.Map getTypeMap()
  ```

- **setTypeMap()**

  ```
  void setTypeMap(java.util.Map map)
  ```

Optional Methods with Restrictions

The following optional java.sql.Connection methods are implemented with restrictions or limitations:

- **prepareCall**()

  Only `TYPE_FORWARD_ONLY` is supported for *resultSetType*. Only `CONCUR_READ_ONLY` is supported for *resultSetConcurrency*.

  ```
  java.sql.CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)
  ```

- **setReadOnly**()

  A no-op (the InterSystems IRIS driver does not support `READ_ONLY` mode)

  ```
  void setReadOnly(Boolean readOnly)
  ```

- **setCatalog**()

  A no-op (the InterSystems IRIS driver does not support catalogs)

  ```
  void setCatalog(String catalog)
  ```

- **setTransactionIsolation**()

  Only `TRANSACTION_READ_COMMITED` and `TRANSACTION_READ_UNCOMMITED` are supported for *level*.

  ```
  void setTransactionIsolation(int level)
  ```

The following java.sql.Connection methods do not support `CLOSE_CURSORS_AT_COMMIT` for *resultSetHoldability*:

- **createStatement()**

  ```
  java.sql.Statement createStatement(int resultSetType, int result, int resultSetHoldability)
  ```

- **prepareCall**()

  ```
  java.sql.CallableStatement prepareCall(String sql,
                                         int resultSetType,
                                         int resultSetConcurrency,
                                         int resultSetHoldability)
  ```

- **prepareStatement**()

```
java.sql.PreparedStatement prepareStatement(String sql,
                                            int resultSetType,
                                            int resultSetConcurrency,
                                            int resultSetHoldability)
```

InterSystems IRIS currently supports only zero or one Auto Generated Keys. An exception is thrown if the java.sql.Connection methods below provide *columnIndexes* or *columnNames* arrays whose lengths are not equal to one.

- **prepareStatement**()

```
java.sql.PreparedStatement prepareStatement(String sql, int[] columnIndexes)
java.sql.PreparedStatement prepareStatement(String sql, String[] columnNames)
```

2.3.3 DatabaseMetaData: Variant Methods

Variant Methods

java.sql.DatabaseMetaData is fully supported, but has methods that vary from the JDBC standard due to InterSystems-specific handling of their return values. The following methods are affected:

- **supportsMixedCaseQuotedIdentifiers()**

  InterSystems IRIS returns `false`, which is not JDBC compliant.

  ```
  boolean supportsMixedCaseQuotedIdentifiers()
  ```

- **getIdentifierQuoteString()**

  If delimited id support is turned on, InterSystems IRIS returns " (double quote character), which is what a JDBC compliant driver should return; otherwise InterSystems IRIS returns a space.

  ```
  String getIdentifierQuoteString()
  ```

2.3.4 Driver: Unsupported Methods

Unsupported Optional Method

java.sql.Driver does not support the following optional method:

- **getParentLogger()**

  ```
  void getParentLogger()
  ```

2.3.5 PreparedStatement: Unsupported Methods

Unsupported Optional Methods

java.sql.PreparedStatement does not support the following optional methods:

- **setArray()**

  ```
  void setArray(int i, Array x)
  ```

- **setRef()**

  ```
  void setRef(int i, Ref x)
  ```

- **setRowId()**

```
void setRowId(int parameterIndex, RowId x)
```

- **setSQLXML()**

```
void setSQLXML(int parameterIndex, SQLXML xmlObject)
```

- **setUnicodeStream()**

  Deprecated in Java JDK specification.

```
void setUnicodeStream(int i, InputStream x, int length)
```

- **setURL()**

```
void setURL(int i, java.net.URL x)
```

2.3.6 ResultSet: Unsupported or Restricted Methods

Optional Method with Restrictions

InterSystems IRIS does not support `TYPE_SCROLL_SENSITIVE` result set types. The following method is implemented with restrictions:

- **setFetchDirection()**

  Does not support `ResultSet.FETCH_REVERSE` (instead, use **afterLast** to move the result set's cursor to after the last row, and use **previous** to scroll backwards).

```
void setFetchDirection(int direction)
```

Unsupported Optional Methods

java.sql.ResultSet does not support the following optional methods:

- **getArray()**

```
Array getArray(int i)
Array getArray(String colName)
```

- **getCursorName()**

```
String getCursorName()
```

- **getObject()**

```
Object getObject(int i, java.util.Map map)
Object getObject(String colName, java.util.Map map)
```

- **getRef()**

```
Ref getRef(int i)
Ref getRef(String colName)
```

- **getHoldability()**

```
int getHoldability()
```

- **getUnicodeStream()**

  Deprecated in Java JDK specification.

```
java.io.InputStream getUnicodeStream(int i)
java.io.InputStream getUnicodeStream(String colName)
```

- **getURL()**

```
java.net.URL getURL(int i)
java.net.URL getURL(String colName)
```

- **updateArray()**

```
void updateArray(int i, Array x)
void updateArray(String colName, Array x)
```

- **updateRef()**

```
void updateRef(int i, Ref x)
void updateRef(String colName, Ref x)
```

2.3.7 Statement: Unsupported or Restricted Methods

Unsupported Optional Methods

java.sql.Statement does not support the following optional methods:

- **cancel()**

```
void cancel()
```

- **closeOnCompletion()**

```
void closeOnCompletion()
```

- **isCloseOnCompletion()**

```
boolean isCloseOnCompletion()
```

Optional Methods with Restrictions

The following optional java.sql.Statement methods are implemented with restrictions or limitations:

- **getResultSetHoldability()**

  Only `HOLD_CURSORS_OVER_COMMIT`

```
int getResultSetHoldability()
```

- **setCursorName()**

  A no-op.

```
void setCursorName(String name)
```

- **setEscapeProcessing()**

  A no-op (does not apply)

```
void setEscapeProcessing(Boolean enable)
```

- **setFetchDirection()**

  Does not support `ResultSet.FETCH_REVERSE` (instead, use **afterLast** to move the result set's cursor to after the last row, and use **previous** to scroll backwards).

```
void setFetchDirection(int direction)
```

InterSystems IRIS currently supports only zero or one auto-generated key. An exception is thrown if the java.sql.Statement methods below provide *columnIndexes* or *columnNames* arrays whose lengths are not equal to one:

- **execute()**

```
boolean execute(String sql, int[] columnIndexes)
boolean execute(String sql, String[] columnNames)
```

- **executeUpdate()**

```
int executeUpdate(String sql, int[] columnIndexes)
int executeUpdate(String sql, String[] columnNames)
```

2.4 InterSystems Enhancements and Extensions

The following classes provide additional InterSystems-specific extension methods:

- CallableStatement getBinaryStream() Extension Method
- ConnectionPoolDataSource Extensions and Enhancements discusses com.intersystems.jdbc.IRISConnectionPoolDataSource, which is the InterSystems implementation of the javax.sql.ConnectionPoolDataSource interface.
- DataSource Extensions and Enhancements discusses com.intersystems.jdbc.IRISDataSource, which is the InterSystems implementation of javax.sql.DataSource.

2.4.1 CallableStatement getBinaryStream() Extension Method

java.sql.CallableStatement implements the following additional InterSystems-specific extension method:

- **getBinaryStream()**

  Retrieves the value of the designated parameter (where *i* is the index of the parameter) as a java.io.InputStream object.

  ```
  java.io.InputStream getBinaryStream(int i)
  ```

  This method is a complement to the standard **setBinaryStream()** method, and an alternative to **getCharacterStream()** (which returns java.io.Reader).

2.4.2 ConnectionPoolDataSource Extensions and Enhancements

The com.intersystems.jdbc.IRISConnectionPoolDataSource class fully implements the javax.sql.ConnectionPoolDataSource interface. This class does not inherit the methods of javax.sql.CommonDataSource, which is not supported by the InterSystems JDBC driver.

Restricted Method

getPooledConnection() is implemented because it is required by the JDBC standard, but the InterSystems IRIS implementation should never be called directly. InterSystems IRIS driver connections must always be obtained by calling the **getConnection()** method. (See "Using a Connection Pool" in *Using Java JDBC with InterSystems IRIS* for more information).

- **getPooledConnection()**

```
javax.sql.PooledConnection getPooledConnection()
javax.sql.PooledConnection getPooledConnection(String usr,String pwd)
```

CAUTION: Calling applications should never use the **getPooledConnection()** methods or the PooledConnection class. InterSystems IRIS driver connections must always be obtained by calling the **getConnection()** method (which is inherited from IRISDataSource). The InterSystems IRIS driver provides pooling transparently through the java.sql.Connection object that it returns.

IRISConnectionPoolDataSource inherits from IRISDataSource (see "DataSource Extensions and Enhancements"), which provides additional InterSystems extension methods.

2.4.2.1 ConnectionPoolDataSource Extension Methods

IRISConnectionPoolDataSource also supports the following additional InterSystems IRIS-only management methods (see "Using a Connection Pool" in *Using Java JDBC with InterSystems IRIS* for more information):

- **restartConnectionPool()**

  Restarts a connection pool. Closes all physical connections, and empties the connection pool.

  ```
  void restartConnectionPool()
  ```

- **getPoolCount()**

  Returns the current number of entries in the connection pool.

  ```
  int getPoolCount()
  ```

- **setMaxPoolSize()**

  Sets a maximum connection pool size. If the maximum size is not set, it defaults to 40.

  ```
  void setMaxPoolSize(int max)
  ```

- **getMaxPoolSize()**

  Returns the current maximum connection pool size

  ```
  int getMaxPoolSize()
  ```

2.4.3 DataSource Extensions and Enhancements

The com.intersystems.jdbc.IRISDataSource class fully implements the javax.sql.DataSource interface. This class does not inherit the methods of javax.sql.CommonDataSource, which is not supported by the InterSystems JDBC driver.

Enhanced Required Method

The InterSystems IRIS implementation of this method is enhanced to provide automatic, transparent connection pooling. (See "Using a Connection Pool" in *Using Java JDBC with InterSystems IRIS* for more information).

- **getConnection()**

  ```
  java.sql.Connection getConnection()
  java.sql.Connection getConnection(String usr,String pwd)
  ```

2.4.3.1 DataSource Extension Methods

In addition to the methods defined by the interface, IRISDataSource also implements the following methods that can be used to get or set DataSource properties supported by InterSystems IRIS. (See "Setting Connection Properties" in *Using Java JDBC with InterSystems IRIS* for more information).

- **getConnectionSecurityLevel()**

  Returns an *int* representing the current Connection Security Level setting.

  ```
  int getConnectionSecurityLevel()
  ```

- **getDatabaseName()**

  Returns a String representing the current database (InterSystems IRIS namespace) name.

  ```
  String getDatabaseName()
  ```

- **getDataSourceName()**

  Returns a String representing the current data source name.

  ```
  String getDataSourceName()
  ```

- **getDefaultTransactionIsolation()**

  Gets the current default transaction isolation.

  ```
  int getDefaultTransactionIsolation()
  ```

- **getDescription()**

  Returns a String representing the current description.

  ```
  String getDescription()
  ```

- **getEventClass()**

  Returns a String representing an Event Class object.

  ```
  String getEventClass()
  ```

- **getKeyRecoveryPassword()**

  Returns a String representing the current Key Recovery Password setting.

  ```
  getKeyRecoveryPassword()
  ```

- **getNodelay()**

  Returns a boolean representing a current TCP_NODELAY option setting.

  ```
  boolean getNodelay()
  ```

- **getPassword()**

  Returns a String representing the current password.

  ```
  String getPassword()
  ```

- **getPortNumber()**

  Returns an *int* representing the current port number.

  ```
  int getPortNumber()
  ```

- **getServerName()**

  Returns a String representing the current server name.

  ```
  String getServerName()
  ```

- **getServicePrincipalName()**

  Returns a String representing the current Service Principal Name setting.

  ```
  String getServicePrincipalName()
  ```

- **getSSLConfigurationName()**

  Returns a String representing the current SSL Configuration Name setting.

  ```
  getSSLConfigurationName()
  ```

- **getURL()**

  Returns a String representing a current URL for this connection.

  ```
  String getURL()
  ```

- **getUser()**

  Returns a String representing the current username.

  ```
  String getUser()
  ```

- **setConnectionSecurityLevel()**

  Sets the connection security level

  ```
  Sets the Connection Security Level for this DataSource object.
  ```

- **setDatabaseName()**

  Sets the database name (InterSystems IRIS namespace) for this connection.

  ```
  void setDatabaseName(String dn)
  ```

- **setDataSourceName()**

  Sets the data source name for this connection. `DataSourceName` is an optional setting and is not used by IRISDataSource to connect.

  ```
  void setDataSourceName(String dsn)
  ```

- **setDefaultTransactionIsolation()**

  Sets the default transaction isolation level.

  ```
  void setDefaultTransactionIsolation(int level)
  ```

- **setDescription()**

  Sets the description for this connection. `Description` is an optional setting and is not used by IRISDataSource to connect.

  ```
  void setDescription(String d)
  ```

- **setEventClass()**

  Sets the Event Class for this connection. The Event Class is a mechanism specific to InterSystems IRIS JDBC. It is completely optional, and the vast majority of applications will not need this feature.

The InterSystems JDBC server will dispatch to methods implemented in a class when a transaction is about to be committed and when a transaction is about to be rolled back. The class in which these methods are implemented is referred to as the "event class." If an event class is specified during login, then the JDBC server will dispatch to **%OnTranCommit** just prior to committing the current transaction and will dispatch to **%OnTranRollback** just prior to rolling back (aborting) the current transaction. User event classes should extend **%ServerEvent**. The methods do not return any values and cannot abort the current transaction.

```
void setEventClass(String e)
```

- **setKeyRecoveryPassword()**

  Sets the Key Recovery Password for this connection.

  ```
  setKeyRecoveryPassword(java.lang.String password)
  ```

- **setLogFile()**

  Unconditionally sets the log file name for this connection.

  ```
  setLogFile(java.lang.String logFile)
  ```

- **setNodelay()**

  Sets the `TCP_NODELAY` option for this connection. Toggling this flag can affect the performance of the application. If not set, it defaults to `true`.

  ```
  void setNodelay(boolean nd)
  ```

- **setPassword()**

  Sets the password for this connection.

  ```
  void setPassword(String p)
  ```

- **setPortNumber()**

  Sets the port number for this connection

  ```
  void setPortNumber(int pn)
  ```

- **setServerName()**

  Sets the server name for this connection.

  ```
  void setServerName(String sn)
  ```

- **setServicePrincipalName()**

  Sets the Service Principal Name for this connection.

  ```
  void setServicePrincipalName(String name)
  ```

- **setSSLConfigurationName()**

  Sets the SSL Configuration Name for this connection.

  ```
  setSSLConfigurationName(java.lang.String name)
  ```

- **setURL()**

  Sets the URL for this connection.

  ```
  void setURL(String u)
  ```

- **setUser()**

Sets the username for this connection.

```
void setUser(String u)
```

3

Apache Spark Support

The InterSystems IRIS™ *Spark Connector* is an implementation of the Data Source API for Apache Spark that allows the Spark data processing engine to make optimal use of the InterSystems IRIS Data Platform and its distributed data capabilities.

This chapter provides technical details about the InterSystems Spark Connector. The following topics are discussed:

- Apache Spark and the InterSystems Spark Connector — provides an overview and resource links for the Spark Connector.
- Spark Connector Compliance and Compatibility — describes the InterSystems iris class that implements the Data Source API, and compares it to the standard Spark jdbc implementation.
- Spark Connector Extensions — lists and discusses InterSystems extensions to the Data Source API.

3.1 Apache Spark and the InterSystems Spark Connector

Apache Spark is a high performance Java analytics engine for use in clustered computing environments. It's heart is the Resilient Distributed Dataset (RDD) which represents a distributed, fault tolerant, collection of data that can be operated on in parallel. Spark includes libraries for SQL, machine learning, graph processing, stream processing, and many other functions.

Spark provides a jdbc data source that allows the results of a complex SQL query executed within the database to be retrieved by Spark as a Dataset, and for a Dataset to be written back into the database as a SQL table.

The InterSystems IRIS data platform can connect to Spark using only the jdbc data source, but the InterSystems Spark Connector implements a custom iris data source that provides important enhancements for optimal performance.

Important: The terms jdbc and iris (lower case, in the same typography as other class names) are used frequently in this book, and always refer specifically to the data source provider class names, never to Java JDBC or InterSystems IRIS.

3.1.1 Installation and Configuration

See the following sections in *Using the InterSystems Spark Connector* for information on installation and configuration:

- Requirements and Configuration provides InterSystems-specific configuration information.
- Spark Connector Best Practices describes ways to optimize Spark Connector hardware and software.

Also see the following related documents:

- Apache Spark documentation on Spark Configuration.
- The org.apache.spark.SparkConf class documentation.

3.2 Spark Connector Compliance and Compatibility

The InterSystems IRIS Spark Connector is a plug-compatible replacement for the Spark jdbc data source.

3.3 Spark Connector Extensions

The InterSystems IRIS Spark Connector provides InterSystems-specific extension methods to improve usability.

Spark is implemented using a combination of Java and Scala, and can run on any JVM. The Data Source API and all extensions are implemented in Scala.

4

Hibernate Support

The InterSystems Hibernate Dialect is an implementation of the Hibernate dialect interface. Since every vendor's implementation of SQL is slightly different, the dialect interface allows vendors to create custom Hibernate mappings for a specific database. Vendor-provided dialect implementations are distributed as part of Hibernate.

This chapter provides technical details about the InterSystems Hibernate Dialect. The following topics are discussed:

- Hibernate and the InterSystems Hibernate Dialect — provides an overview and resource links for the Hibernate Dialect.
- Installation and Configuration — provides InterSystems-specific instructions.
- Dialect File Locations — lists the InterSystems dialect files and their required locations.

4.1 Hibernate and the InterSystems Hibernate Dialect

Java Persistence Architecture (JPA) is the recommended persistence technology for complex object hierarchies in Java projects. InterSystems currently supports JPA via the Hibernate implementations of the JPA specifications. Hibernate is an open source framework from JBoss that acts as a wrapper around JDBC to provide object/relational mapping (ORM) services for relational databases. Hibernate provides a vendor-neutral persistence service, which may be a requirement for some projects.

The InterSystems Hibernate Dialect is an implementation of the Hibernate dialect interface. Since every vendor's implementation of SQL is slightly different, Hibernate includes vendor-provided "dialects" that customize its mappings to specific databases. Current Hibernate distributions include a high performance, customized InterSystems dialect class.

4.1.1 When to Use Hibernate

Hibernate provides the infrastructure to persist objects to relational tables. Essentially, it is a wrapper around JDBC that allows you to focus on working with objects while transparently handling conversion between objects and tables in SQL queries. Hibernate can be used in most environments, but it is not always the best option. Here are some considerations to bear in mind:

- Hibernate is helpful when you have a complex but static object model. You must know what your data looks like and how the classes interact before you map them to your InterSystems IRIS table model.
- Since Hibernate objects are cached, other applications should never interact with the data while Hibernate is accessing it. If you are working in an environment with real-time data that must remain accessible to other applications, you should consider XEP (see *Persisting Java Objects with InterSystems XEP*) as a possible alternative.

- Hibernate is good for common CRUD operations with simple querying, but more complex queries may be easier to write, or more efficient, using JDBC directly.

4.2 Installation and Configuration

This section provides instructions for setting up your system to use Hibernate with InterSystems IRIS. The instructions assume that the correct versions of both InterSystems IRIS and Hibernate are installed and operational.

4.2.1 Requirements

The following software must be installed on your system:

- InterSystems IRIS™
- Hibernate 5.2 or 5.3. Hibernate can be downloaded from www.hibernate.org.
- A supported version of the Java JDK 1.8 or higher (see "Supported Java Technologies" in the online *InterSystems Supported Platforms* document for this release).

4.2.2 Directories

The instructions in this chapter refer to the following directories:

- <install-dir> — the InterSystems IRIS installation directory. To locate <install-dir> in your instance of InterSystems IRIS, open the InterSystems terminal and issue the following command:

  ```
  write $system.Util.InstallDirectory()
  ```

  See "InterSystems IRIS Installation Directory" in the *Installation Guide* for system-specific information on the location of <install-dir>.
- <hibernate_root> — your Hibernate installation directory.

4.2.3 System Settings

Make the following changes to your system:

- *intersystems-jdbc-3.0.0.jar File*

  The intersystems-jdbc-3.0.0.jar file contains the InterSystems JDBC driver. If you haven't already done so, copy the appropriate version of intersystems-jdbc-3.0.0.jar from the default location to <hibernate_root>\lib.
- *Java Classpath*

  Make sure the following items are on your Java classpath:

  – The jar files from <hibernate_root>\lib
  – The directory or directories where the Hibernate configuration files (hibernate.properties and hibernate.cfg.xml) are kept. By default, both files are in <hibernate_root>\etc.

4.2.4 Hibernate Configuration

In the Hibernate configuration files (either hibernate.properties or hibernate.cfg.xml), specify the connection information for your database, and the name of the InterSystems dialect class.

The following five configuration properties are required:

- *dialect* — The fully qualified name of the InterSystems dialect class. The base dialect class is:

  ```
  org.hibernate.dialect.InterSystemsIRISDialect
  ```

  You can use a custom dialect class derived from this base class if you need to enable support for the Hibernate primary key generator classes.

- *driver_class* — The fully qualified name of the InterSystems JDBC driver:

  ```
  com.intersystems.jdbc.IRISDriver
  ```

  The JDBC driver is contained in the intersystems-jdbc-3.0.0.jar file (see "System Settings" for details).

- *username* — Username for the InterSystems IRIS namespace you want to access (default is _SYSTEM).

- *password* — Password for the InterSystems IRIS namespace (default is SYS).

- *url* — The URL for the InterSystems JDBC driver. The format for the URL is:

  ```
  jdbc:IRIS://<host>:<port>/<namespace>
  ```

  where <host> is the IP address of the machine hosting InterSystems IRIS, <port> is the SuperServer TCP port of your InterSystems IRIS instance, and <namespace> is the namespace that contains your InterSystems IRIS database data (see Defining a JDBC Connection URL for more details).

A typical entry in hibernate.properties would contain the following lines (change url, username, and password as appropriate for your system)):

```
hibernate.dialect org.hibernate.dialect.InterSystemsIRISDialect
hibernate.connection.driver_class com.intersystems.jdbc.IRISDriver
hibernate.connection.url jdbc:IRIS://127.0.0.1:51773/USER/
hibernate.connection.username _SYSTEM
hibernate.connection.password SYS
```

The following example shows the same information as it would appear in hibernate.cfg.xml:

```
<hibernate-configuration>
  <session-factory>
    <property name="dialect">
      org.hibernate.dialect.InterSystemsIRISDialect
    </property>
    <property name="connection.driver_class">
      com.intersystems.jdbc.IRISDriver</property>
    <property name="connection.username">_SYSTEM</property>
    <property name="connection.password">SYS</property>
    <property name="connection.url">
      jdbc:IRIS://127.0.0.1:51773/USER
    </property>
  </session-factory>
</hibernate-configuration>
```

CAUTION: If the same property is set in both hibernate.properties and hibernate.cfg.xml, Hibernate will use the value from hibernate.cfg.xml.

4.3 Dialect File Locations

The InterSystems Hibernate dialect consists of four files that should be located as follows

(where <*hibernate*> is hibernate-orm\hibernate-core\src\main\java\org\hibernate):

- InterSystemsIRISDialect.java in <*hibernate*>\dialect\
- IntersystemsIRISIdentityColumnSupport.java in <*hibernate*>\dialect\identity\
- InterSystemsIRISSQLExceptionConversionDelegate.java in <*hibernate*>\exception\internal\
- InterSystemsIRISJoinFragment.java in <*hibernate*>\sql\

If you do not already have these files, contact the InterSystems Worldwide Response Center (WRC) for download information.