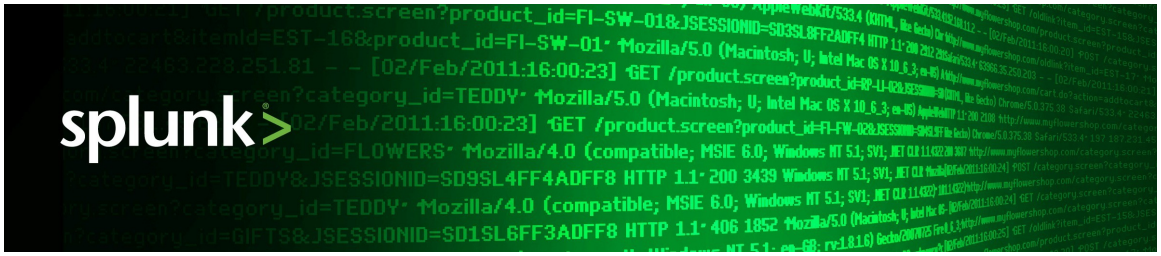

Splunk® DB Connect Deploy and Use Splunk DB Connect 1.2.2

Generated: 9/28/2016 11:54 am

Table of Contents

Introduction

About Splunk DB Connect

Splunk DB Connect lets you enrich and combine your machine data with database data. You can use the app to configure database queries and lookups in minutes via the Splunk Web interface.

Quickly deploy Splunk Enterprise for real-time machine data collection, indexing, analysis, and visualization. Then use Splunk DB Connect to import and index the data already stored in your database to gain more insight.

Furthermore, database lookups let you reference fields in an external database that match fields in your event data. Using these matches, you can add more meaningful information and searchable fields to enrich your event data.

How this app fits into the Splunk picture

Splunk DB Connect is one of a variety of **apps** and **add-ons** available in the Splunk ecosystem. All Splunk apps and add-ons run on top of a Splunk Enterprise installation. You must first install Splunk Enterprise, then install the Splunk DB Connect app.

- For installation instructions, see "Install Splunk DB Connect."
- For details about Splunk apps and add-ons, refer to "What are apps and add-ons?" in the Splunk Admin Manual.
- To download Splunk Enterprise, visit the download page on splunk.com.
- To get more apps and add-ons, visit Splunk Apps.

How to get help and learn more about Splunk

Splunk DB Connect version 1.0.8 and later is officially supported by Splunk.

How to get help

To get help with the Splunk DB Connect App, send an email to support@splunk.com, or log a support case via the Splunk Support Portal.

If your deployment is large or complex, you can engage a member of the Splunk Professional Services team. They will assist you in deploying the Splunk DB Connect App.

Learn more about Splunk

There are a variety of resources available to help you learn more about Splunk and the Splunk DB Connect App, including:

- Splunk Enterprise documentation
- Splunk Answers
- The #splunk IRC channel on EFNET

Before you deploy

Deployment requirements

Supported databases

Splunk DB Connect tests and supports connection to these databases:

- DB2
- Microsoft SQL Server
    - See Add a database to enable the Microsoft driver
- MySQL
- Oracle Database
- Sybase, Adaptive Server Enterprise version 15.7 Developer's Edition

You can also connect to these unsupported databases:

- Generic ODBC support
- H2
- HyperSQL
- PostgreSQL
- SQLite

Provide the necessary JDBC drivers to add your own database types.

Supported Splunk versions

The Splunk DB Connect app runs on Splunk 4.3 and later.

Notes:

- Splunk DB Connect has not been tested and is not supported with Splunk Free.
- Splunk DB Connect is not currently certified or supported in clustered environments.
- Splunk DB Connect is not compatible with Splunk servers configured for FIPS compliance.

Supported operating systems

Splunk DB Connect runs on supported Splunk versions on the following operating systems:

- Linux
- Mac OS X
- Windows Server 2003/2008/2008R2
- Windows XP/7 (for development/testing)

Java Runtime Environment (JRE)

Before deploying Splunk DB Connect, install the Java Runtime Environment (JRE) from Oracle Java SE Downloads.

- Supported versions: DB Connect 1.1.x requires Java 1.6 or 1.7 only. DB Connect 1.2 requires Java 7 or 8 only. DB Connect 2.0 requires Java 8 only.
- Do not use a hotspot (client mode only) JVM.

  Note: Only the Oracle JRE is certified and supported for use with Splunk DB Connect. Customers have reported problems when starting the Java Bridge Server under alternate JREs or JDKs such as OpenJDK or IBM Java. You can download the Oracle JRE and JDK packages from http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Splunk Licenses and DB Connect

If you configure DB Connect to output data from a connected database into a Splunk index, the amount of data indexed does count towards your Splunk license. However, performing database lookups against a connected database in DB Connect does not count towards the license.

For more information about Splunk licenses, see "How Splunk licensing works."

Splunk DB Connect Release Notes

For the latest known issues and fixed problems in Splunk DB Connect, see Release Notes.

Architecture and performance

Splunk topology

If you have a trial or personal Splunk deployment running on a single host (indexer and Splunk Web both running on the same system), you can install Splunk DB Connect on this system.

To use Splunk DB Connect for reporting or database lookups in a search head pooling environment, you must install the app on a search head. For instructions on installing apps in a search head pooling environment, see Create a search head pool. For instructions on configuring search head pooling for Splunk DB Connect, see Set up search head pooling.

> **Note:** Splunk DB Connect is not currently certified or supported for use with search head clusters or indexer clusters. For more information, see About Splunk DB Connect and search head clustering and indexer clustering, later in this topic.

In a distributed environment, you must perform lookups on the search head where Splunk DB Connect is installed. To perform a lookup locally, add `local=1` after the `lookup` command.

Example:

```
index=test | lookup local=1 mysql_table ip_address as clientip OUTPUT
host | table clientip, host
```

This is not currently possible when using automatic lookups. For more information on automatic lookups, see Edit existing automatic lookups or configure a new lookup to run automatically.

For database inputs, depending on the anticipated volume of your deployment, there are 3 options:

- Small scale: install Splunk DB Connect on a **search head** for monitoring and configure it to forward events to the indexer(s)
- Medium scale: use a dedicated Splunk **heavy forwarder** to perform monitoring and forward events to indexer(s).
- Large scale: Use multiple dedicated Splunk **forwarders** and partition the monitors among them.

About search head pooling and dbmon-tail

We do not recommend using dbmon-tail inputs in a search head pooling environment. In a search head pooling environment, each search head has its own persistent storage that keeps track of the last rising column. This can cause Splunk to index different values for each search head.

We recommend instead that you use a dedicated heavy forwarder with DB Connect installed, to forward data to Splunk indexers.

About Splunk DB Connect and search head clustering and indexer clustering

A search head cluster, introduced in Splunk Enterprise 6.2, is a group of search heads that serves as a central resource for searching. An indexer cluster is a group of Splunk Enterprise indexers that replicates external data. Splunk DB Connect is not currently certified or supported for use with search head clusters or indexer clusters. However, you have the following options:

- Use search head pooling with Splunk DB Connect. For more information, see Set up search head pooling. Be aware that search head pooling was deprecated in Splunk 6.2, and may not be available in future releases.
- Use data inputs and outputs on a dedicated search head or heavy forwarder.

Performance considerations

Because Splunk DB Connect queries your database, there is a possibility that your queries may impact database performance. In particular, if the initial run of your query to the database retrieves a lot of data, this may affect the performance of your database. Subsequent runs of the query should have less impact, as they are only retrieving new data since the previous run of the query. To mitigate this, you can set the `tail.follow.only` option in the dbmon-tail stanza in `inputs.conf`.

Lookups generate multiple selects that should be within the expected workload for a database and should not affect performance. Splunk DB Connect executes a separate SELECT statement for each unique combination of input fields. This may happen more than once per search, because the search preview function in Splunk may invoke the lookup multiple times during execution of a search for parts of the results. Splunk does not cache the results between invocations of the lookup.

Install Splunk DB Connect

Install Splunk DB Connect

This page shows you how to install and configure Splunk DB Connect. It assumes that you have an existing Splunk instance to use as the underlying platform. For information on installing Splunk, refer to "Before you install" in the Splunk Enterprise platform documentation.

Note: Modifying `inputs.conf` file stanzas outside of the DB Connect app, such as in the search app, or manager context is not supported.

Install the Splunk DB Connect App

The easiest way to install the Splunk DB Connect App is to use Splunk Web, as follows:

1. Download Splunk DB Connect and save it to a temporary location accessible from your Splunk instance.

2. Log into Splunk Web, go to **Apps > Manage Apps** and click **Install app from file**.

3. Select the app package `splunk-db-connect_<version>.tgz` and upload it.

4. When the upload is complete, follow the instructions to restart Splunk.

Upgrade from a previous version

Note: Currently, there is no built-in mechanism to rollback an upgrade, so we strongly recommend making a backup of the `$SPLUNK_HOME/etc/apps/dbx` directory prior to upgrading.

Upgrading from an earlier version of Splunk DB Connect is similar to installing the app from scratch:

1. Download the latest Splunk DB Connect installation package from Splunk Apps.

2. Log into Splunk Web, go to **Apps > Manage Apps** and click **Install app from**

file.

3. Browse to the DB Connect installation package (.tgx) that you downloaded to a temporary location, and click **Upload**. If you are upgrading from an earlier version of the app, check the **Upgrade app** box. This overwrites the earlier version of the app with the newer version.

4. Click **Restart Splunk** when prompted; or restart Splunk via the command line, as shown:

```
./splunk restart
```

If you encounter problems with this standard upgrade approach, try this upgrade procedure.

Note: After upgrading DB Connect, you might encounter this error creating PersistentValueStore.

Setup Splunk DB Connect

After you install DB Connect and restart Splunk Enterprise, you must complete the following setup tasks:

Enable splunkd SSL

To run DB Connect, you must enable SSL for `splunkd`.

1. Go to `$SPLUNK_HOME/etc/system/local/server.conf`.

2. In the `[sslConfig]` stanza, set `enableSplunkdSSL` to `true`, as shown:

```
[sslConfig]
enableSplunkdSSL = true
```

Note: `splunkd` is enabled by default.

Complete the app setup from the UI

1. Go to **Apps > Splunk DB Connect**.

The Splunk DB Connect Setup page appears.

2. Enter your JAVA_HOME path. This is where your JRE (Java Runtime Environment) resides. For example:

```
echo $JAVA_HOME

/usr/lib/jvm/java-1.6.0-openjdk-1.6.0.0.x86_64/jre
```

3. Click **Save**.

This enables the Java Bridge Server.

Note: To verify that the Java Bridge Server is running, make sure that the scripted input `jbridge_server.py` is enabled. See step 3 of Command Line Setup.

Command Line Setup

You can setup DB Connect manually from the command line.

1. Create `$SPLUNK_HOME/etc/apps/dbx/local/app.conf`

```
[install]
is_configured = 1
```

2. Create `$SPLUNK_HOME/etc/apps/dbx/local/java.conf`

```
[java]
home = <JAVA_HOME path>
```

This is the path to your (JRE) Java Runtime Environment. For example:

```
home=/usr/lib/jvm/java-1.6.0-openjdk-1.6.0.0.x86_64/jre
```

3. Enable the Java Bridge Server (scripted input) in `$SPLUNK_HOME/etc/apps/dbx/local/inputs.conf`

```
[script://$SPLUNK_HOME/etc/apps/dbx/bin/jbridge_server.py]
disabled = 0
```

4. Create the sink for database inputs in `$SPLUNK_HOME/etc/apps/dbx/local/inputs.conf`

```
[batch://$SPLUNK_HOME/var/spool/dbmon/*.dbmonevt]
crcSalt = <SOURCE>
disabled = 0
move_policy = sinkhole
sourcetype = dbmon:spool
```

5. Restart Splunk

Advanced Setup Options

Use the following information to setup custom configurations for Splunk DB Connect in `$SPLUNK_HOME/etc/apps/dbx/local/java.conf`

Java Settings

```
[java]

home = <JAVA_HOME path>
```

- Path to your JRE (Java Runtime Environment) directory. Your JAVA_HOME environment variable retrieves this path.

```
options = <string>
```

- Java command line options. These (optional) parameters are called when you start your Java instance. You can specify multiple optional parameters, including:
  - `-Xmx`: Maximum memory usage. Change the value of this parameter if your Java application requires more or less memory. For example, you can increase the default value (`-XMx256m`) to a higher value, such as `-XMx512m` or `-Xmx1024m`.
  - `-Duser.language`: Default user language. For example, -Duser.language=en
  - `-Dfile.encoding`: Default file encoding. For example, -Dfile.encoding=UTF-8
  - `-Duser.region`: Default region. For example, -Duser.region=US (See Class Locale.)

Important: Incorrect formatting of this field can prevent Splunk DB Connect from starting correctly.

Java Bridge Server

[bridge]

`addr = <bind address>`

- The IP address of your Java Bridge Server (typically 127.0.0.1 (localhost)).

`port = <bind port>`

- The port of your Java Bridge Server. Default is 17865.

`Important:` `There must not be any firewall rules activated for this port.`

`threads = <n>`

- The size of the thread pool for Java Bridge command execution. Determines the number of commands that can run concurrently.

Note: Too many or too few threads can slow performance of the Java Bridge service.

`debug = true|false`

- Turns on debugging for the Java Bridge client. When enabled, the Java Bridge logs any debug information in `jbridge_client.log`.

Important: Enabling debugging can have a negative impact on performance. We do not recommend enabling debugging for the Java Bridge client in a production environment.

Logging settings

- level: Log severity level for Splunk DB Connect.
- file: The name of the Splunk DB Connect log file located in $SPLUNK_HOME/var/log/splunk. The default log file name is `dbx.log`

Database Connection Handling

- Factory Type
- Enable connection pooling
- Cache database and table metadata
- Preload database configuration

Database Inputs

- Scheduler Threads
- Output Type
- Default timestamp output format

Database Lookups

- Enable caching of database lookup definitions
- Cache invalidation timeout

Persistence

- Global Store type

Install database drivers

Many Splunk DB Connect supported databases are preconfigured and require only that you add a database connection and define inputs for that database.

But if you wish to connect a MySQL, Oracle, DB2, or Informix database to Splunk via Splunk DB Connect, you must download and install the correct JDBC drivers as shown below.

Step 1: Download driver

Download the appropriate JDBC driver for your database, as follows:

- **MySQL**
  - Download the MySQL Connector/J driver, version 5.1.24 or later (`mysql-connector-java-*-bin.jar`).
  - Download and uncompress either ZIP or TAR archive files. The uncompressed archive contains the JDBC driver (.jar).
  - Copy the `mysql-connector-java-version-bin.jar` file to the `$SPLUNK_HOME/etc/apps/dbx/bin/lib` directory.
- **Microsoft SQL Server**
  - There are two JDBC driver options for Splunk DB Connect and Microsoft SQL Server. The better option is Microsoft's own JDBC Driver. To enable Microsoft SQL Server connections, download and install the Microsoft JDBC Driver for SQL Server.
    - Go to the Microsoft JDBC Drivers 4.1 and 4.0 for SQL

Server download page and click **Download**.

◊ On the Choose the download you want page, select the checkboxes next to the appropriate download: **sqljdbc_4.0.2206.100_enu.tar.gz** for Linux, Unix, and OS X; **sqljdbc_4.0.2206.100_enu.exe** for Windows. Be sure to download version 4.0 of the driver, and then click **Next**.

◊ Expand the file you just downloaded.

◊ From inside the **sqljdbc_4.0** directory, copy or move the **sqljdbc4.jar** file to the **$SPLUNK_HOME/etc/apps/dbx/bin/lib** directory.

> **Note:** Do not copy the **sqljdbc.jar** file. It is not needed and will cause problems if installed alongside the **sqljdbc4.jar** file.

- **Oracle**
  - Download the Oracle JDBC driver (`ojdbc6.jar`).
- **DB2**
  - Go to the DB2 JDBC Driver Versions download site.
  - Login (register, if needed). As part of the login, check the **license agreement** checkbox, then click the **I confirm** button.
  - Check the **download** checkbox for:

    IBM Data Server Driver for JDBC and SQLJ (JCC Driver)
    ibm_data_server_driver_for_jdbc_sqlj_v10.5.zip (9 MB)
  - Click the **Download now** button, saving the file to a temporary directory.
  - Unzip the downloaded file.
  - Copy or move the `db2jcc4.jar` file to the `$SPLUNK_HOME$/etc/apps/dbx/bin/lib` directory.
  - **Note:** Only move the `db2jcc4.jar` file to the `.../dbx/bin/lib` directory, not `sqlj.zip`, to avoid conflicts.
  - See Installing and Connecting to Clients for additional information.
- **Informix**
  - Download the Informix JDBC driver (`ifxjdbc.jar`).

Step 2: Install driver

After you have downloaded the correct driver for your database and platform, install the driver as follows:

1. Copy the driver to the `$SPLUNK_HOME/etc/apps/dbx/bin/lib` directory.

2. Restart Splunk.

Adding a database that is not in the list of Supported Databases

If you want to connect to a database that is not in the list of Supported Databases, see Add a database for instructions.

Add a database

Splunk DB Connect provides built-in support for a variety of databases (see "About Splunk DB Connect"). You can also add custom support for any database that has a JDBC driver.

Note: At a minimum, Splunk DB Connect supports querying custom database connections. For some custom database connections, certain query-related features may not work. Also, depending on the database-implementation of the JDBC driver, the `dbinfo` search command may not work as expected.

Download and install the relevant JDBC driver

Before you can add a custom database connection, you must download the JDBC (Java Database Connectivity) driver for the database you want to add, and copy the .jar file to `$SPLUNK_HOME/etc/apps/dbx/bin/lib`.

Add the custom database to database_types.conf

When you add a custom database connection that Splunk DB Connect does not support by default, you must create a stanza to define the database connection in a copy of `database_types.conf`.

Important: Do not edit the `database_types.conf` file in `$SPLUNK_HOME/etc/apps/dbx/default`. Instead, copy the filed to `$SPLUNK_HOME/etc/apps/dbx/local` and perform your edits there. For more information, see About configuration files.

Example new database stanzas in database_types.conf

```
[generic_mssql]
displayName = MS-SQL Server Using MS Generic Driver
```

```
        jdbcDriverClass = com.microsoft.sqlserver.jdbc.SQLServerDriver
        testQuery = SELECT 1
        connectionUrlFormat =
jdbc:sqlserver://{0}:{1};databaseName={2};selectMethod=cursor

        [postgresql]
        displayName = PostgreSQL
        jdbcDriverClass = org.postgresql.Driver
        defaultPort = 5432
        connectionUrlFormat = jdbc:postgresql://{0}:{1}/{2}
        testQuery = SELECT 1 AS test
        defaultCatalogName = postgres
        defaultSchema = public
```

Connection Validation

Each time DB Connect uses a database connection, it tries to validate that the database connection is actually working. If validation fails, you might see an error message, such as "ValidateObject failed".

DB Connect uses these two methods to validate a connection:

- If a testQuery is specified in `database_types.conf`, DB Connect executes that query, and receives a response that validates that the connection is working.
- If testQuery is not specified, DB Connect uses the Java method connection.isValid(), and relies on the JDBC driver to answer. Some JDBC drivers do not implement this API call (seems like Derby is build against Java 1.5 source, where JDBC doesn't have the method isValid). The workaround is to specify a manual testQuery, such as `SELECT 1`.

Note: You can disable connection validation by setting `validationDisabled=true` in `database_types.conf`.

Add the database connection in Manager

After you add the custom database, proceed to "Add or manage a database connection" to setup and manage your database connection.

Configure and use Splunk DB Connect

Manage a database connection

Before you can interact with a database through Splunk DB Connect, you must configure a connection to that database. You can then use the database in queries, lookups, inputs, and outputs.

Note: To setup a connection to a database that is not listed under Supported Databases, see Add a database.

Create a new database connection

1. In Splunk Web, select **Apps > Splunk DB Connect**.

2. Click **Database connections in Splunk Manager**.

The **External Databases** page shows a list of existing database connections.

3. Click **New**.

The **Add New External Databases** page opens.

4. Enter the following:

1. **Name:** Type a unique name that identifies your new database connection.
2. **Database Type:** The type of database to which you want to connect.
3. **Transaction Isolation Level:** The transaction isolation level determines the degree to which database transactions are isolated from other concurrent transactions. This typically involves the use of data locking to prevent concurrent transactions on shared database objects (such as rows, pages, etc.), which can cause undesirable read phenomena, data corruption, and data loss.

   Select a transaction isolation level for this database connection:

   - `DATABASE_SETTING`: Select this option to maintain your database's existing transaction isolation level. Splunk makes no changes to the existing isolation settings.
   - `TRANSACTION_NONE`: This option is not supported in the current release of DB Connect.

- `TRANSACTION_READ_UNCOMMITTED`: This is the lowest isolation level. This level allows "dirty reads," as a transaction may return a value that is not committed (and can be rolled back to a pervious value) and is thus invalid. This level is appropriate for queries of static tables whose data is not being modified. This is the only isolation level available to databases that do not have transactions.
- `TRANSACTION_READ_COMMITTED`: This isolation level locks a row until after it is committed, thus preventing dirty reads. This level is appropriate when each row of data is processed as an independent unit, without reference to other rows. Use this option to guarantee that all retrieved rows are committed when the row is retrieved. This isolation level does not place a lock on retrieved rows, however, so "phantom reads" can occur.
- `TRANSACTION_REPEATABLE_READ`: In this isolation level, transactions maintain read/write locks on all retrieved rows until the end of the transaction. This ensures that retrieved rows are not updated during the transaction. However, range-locks are not managed, so phantom reads can occur.
- `TRANSACTION_SERIALIZABLE`: This is the highest transaction isolation level. In this level, a shared lock is placed on every row selected during the transaction. Another transaction can also place a shared lock on a selected row, but no other process can modify any selected row during your transaction or insert a row that meets the search criteria of your query during your transaction. The shared locks are released only when the transaction is committed or rolled back. This is the only isolation level that prevents phantom reads.

  For more information on Informix Isolation Levels, see the IBM Informix documentation.

4. **Host:** The host name or IP address of the database server. For local database types (such as SQLite or ODBC) you can use any value (for example, "localhost"). For Microsoft SQL servers, you can use a fully qualified domain name, a short name, or an IP address. Do not use the Microsoft SQL convention of `<SERVERNAME>\<DATABASE>` for the host field.
5. **Port:** The TCP Port to connect to. You can leave this field empty if you are using the default port of the selected database type or if the database is local. Many Microsoft SQL Servers use dynamic ports instead of TCP/1433. Work with your database administrator to identify the correct port, or see "Verifying the port configuration of an instance of SQL Server" .
6. **Username** and **Password:** If the database connection requires username and password for authentication, provide them here. For Windows users, you can use the following notation in the **Username** field:

`<DOMAIN>\<USERNAME>. arg.useNTLMv2 = true` is implied if you use this notation. You can override this in the config file.

7. **Database:** You can leave this field empty to connect to the default database, if the selected database type supports this.

   Note: When you add a local database such as SQLite, specify the fully qualified path to the database file. You can place the SQLite file into `$SPLUNK_HOME/var/dbx` and name it `database_name.sqlitedb`. You can then use "database_name" instead of the fully qualified path.

8. **Additional JDBC Parameters:** Enter additional JDBC parameters to connect with your database.

   MS SQL database connections require this additional parameter:

   ```
   useCursors=true
   ```

   Informix database connections require an additional parameter, such as:

   ```
   informix.server=demo_on.
   ```

   Important: If you are connecting to an Informix database, make sure the Informix JDBC driver (`ifxjdbc.jar`) is installed in $SPLUNK_HOME/etc/apps/dbx/bin/lib.

9. **Read only check box:** You can set the database connection to read-only. If this check box is selected, Splunk DB Connect will not run any SQL statements that modify the database. And the </code>dboutput</code> command will not work.
10. **Validate Database Connection:** If this check box is selected, Splunk DB Connect tries to connect to the database before saving the connection information. If the connection does not succeed, an error message appears.

Modify a database connection

You can modify or delete a database connection using the same **External Databases** page.

1. On the **External Databases** page, click the name of the specific database connection that you want to modify.

The configuration page for that database connection opens.

2. Make changes to the database connection configuration and click **Save**.

Delete a database connection

1. On the **External Databases** page, click **Delete** to the right of the database connection name.

2. Click **OK**.

The database connection is deleted.

After you modify or delete the database connection, Splunk reloads the Java Bridge Server (JBS) database list.

Manage database connections using configuration files

You can manage your database connections by editing a copy of the `database.conf` file, or, if you're connecting to a database not supported out-of-the-box, the `database_types.conf` file.

Important: Do not edit these configuration files in `$SPLUNK_HOME/etc/apps/dbx/default`. You must create and edit a new copy of each configuration file in`$SPLUNK_HOME/etc/apps/dbx/local`. See "About configuration files".

After you edit `database.conf`, you can restart Splunk (which also restarts the JBS), or reload Splunk using this command:

```
splunk cmd python $SPLUNK_HOME/etc/apps/dbx/bin/reload.py
databases
```

The Java Bridge Server picks up the configuration file modifications and encrypts passwords in the configuration files.

Configure database input queries

A database input lets you fetch and index data from a SQL database. Unlike other input sources, database inputs are retrieved periodically by the DBmon scheduler.

Note: Because Splunk DB Connect queries your database, it can have an impact on database performance. This is likely if your initial tail query retrieves a large amount of data. Subsequent queries that retrieve new

data only are likely to have less impact on database performance.

To add a database input:

1. In Splunk Web, select **Apps > Splunk DB Connect**. The Splunk DB Connect app opens.

2. Select **Settings > Manage database inputs**. The **Database Inputs** page opens.

3. Click **New**. The **Add New Database Inputs** page opens. Use this page to configure database input, output, and schedule query intervals.

Configure input query

1. Assign a unique **Name** to your input.

2. Select your **Input Type** from the drop-down list.

- **Tail** finds the new records you want and returns only those records with each query.
- **Dump** invokes the same query each time and returns all results.

3. Select the **Database** from the drop-down list.

Important: On MySQL instances, the terms "database" and "schema" are interchangeable. Unlike other RDBMS systems, MySQL only supports a single schema per database. Therefore, when selecting from the database drop-down list, the schema dropdown will be always be set to "all."

4. (optional) Select the **Specify SQL query** check box if you want to run a custom SQL query against the database. This opens the **SQL Query** field where you type your query string. For example:

```
SELECT * FROM my_table {{WHERE $rising_column$ > ?}}
```

Place the `WHERE` clause in curly braces `{{...}}`. The literal `$rising_column$` is replaced with the name specified in the **Rising Column** field in step 6. The checkpoint value is substituted for the literal `?`.

For the initial run, when there is no checkpoint state, the query does not include the part inside the curly braces, `{{...}}`. On subsequent queries, the query includes the part inside the curly braces.

If **Rising column** is a date, wrap the checkpoint parameter in a `"to_date"` construct. For example: `{{AND $rising_column$ > to_date (?,'YYYY-MM-DD"T"HH:MI:SS')}}`. The correct `"to_date"` function to use depends on the database type. In MySQL, the to_date function is `STR_TO_DATE`.

For Oracle, use uppercase for the name of the **Rising column**.

For more information, see How dbmon-tail inputs work.

5. If you do not specify a SQL query, type the database table name that you want to query in the **Table Name** field. Splunk provides the appropriate query string.

6. If you selected the **Tail** input type in step 2, specify the **Rising Column** in the **Tail input settings** panel. Choose a column with an increasing value, such as the creation or modification timestamp, or a sequential identifier.

Caution: Do not rename the rising_column. Doing so can break your database input. See "Renaming rising_column breaks database input" in the Troubleshooting section of this manual.

7. Specify a data **Sourcetype**.

The following formats are associated with the sourcetypes:

- Key-Value format with dbmon:kv sourcetype
- Multi-line Key-Value format with dbmon:mkv sourcetype
- Template with dbmon:tpl sourcetype
- CSV format with CSV sourcetype

Note: If you leave the **Sourcetype** field blank, the pre-defined sourcetype associated with the format is used.

To use a custom sourcetype, specify line-break and timestamp settings in `$SPLUNK_HOME$/etc/apps/dbx/local/props.conf` or `$SPLUNK_HOME$/etc/system/local/props.conf` file.

For sourcetype line-break specifications, see `$SPLUNK_HOME$/etc/apps/dbx/default/props.conf` or `$SPLUNK_HOME$/etc/system/default/props.conf`.

8. Specify the name of the **Index** associated with this input.

9. Specify the name of the host as **Host Field value**, for this database.

How dbmon-tail inputs work

When you create a **Tail** input in the UI, DB Connect adds a `dbmon-tail` stanza to your `inputs.conf` file, in `$SPLUNK_HOME/etc/apps/dbx/local`.

Unlike `dbmon-dump` inputs, which index all data from the specified table each time the SQL query executes, a `dbmon-tail` input filters the table data input based on an increasing value specified in a "rising column." This lets you index only new data appended to the table since the last SQL query.

You can specify as rising column any column whose value increases over time, such as a timestamp or sequential ID. For example, a rising column could be last_update, employee_id, customer_id, transaction_id and so on.

For `dbmon-tail` inputs, the SQL query is broken into two parts: The main SQL, plus a filter condition, such as

```
{{WHERE $rising_column$ > ?}}
```

For example, the SQL statement:

```
SELECT customer_id, last_name, first_name FROM customer {{WHERE
$rising_column$  > ?}}
```

is executed as follows:

Note: `customer_id` is set as rising column.

1. When DB Connect runs this SQL statement for the first time, only the main part of the SQL statement executes:

```
SELECT customer_id, last_name, first_name FROM customer
```

Once the last record is retrieved from the main SQL query, DB Connect stores the highest value of the rising column in the `state.xml` file pertaining to this input, under `$SPLUNK_HOME/var/lib/splunk/persistentstorage/dbx`.

2. Upon subsequent executions of this `dbmon-tail`, DB connect emits the full SQL statement, which includes both the main SQL statement and the

tail portion containing the filter condition:

```
SELECT customer_id, last_name, first_name FROM customer {{WHERE
$rising_column$  > ?}}
```

So this would be the actual SQL DB Connect emits to the database:

```
SELECT customer_id, last_name, first_name FROM customer WHERE
customer_id  > 10
```

Note: In this case, the number 10 is the value that DB Connect stores in the state.xml file from the previous execution. "?" is the variable that represents the `state.xml` value.

Configure database output

Define your database output parameters.

1. Select an **Output Format** to determine how Splunk renders your output data.

- Key-Value format
- Multi-line Key-Value format
- Template
- CSV
- CSV (with headers)

If you select **Template** output format, specify the template with placeholders for the column values returned from the database. DB Connect applies the template to each row returned by the query, then writes the resulting text to the output, which is then indexed. For example, this template:

```
Event ID $ID$ from $HOST$ at $timestamp$
```

returns output like this:

```
Event ID 4712 from myhost.foobar.com at 2013-10-30T13:59:12.201Z
```

Note: Line-breaking settings in props.conf and transforms.conf can impact template output. In addition, template output format does not provide automatic field extractions, so you must extract fields manually. We recommend that you use **Template** output format only if you have a specific format that you want to use for your database input.

If you select **CSV** or **CSV (with headers)** output format, and you want a complete CSV file without line breaks, then you must specify the `SHOULD_LINEMERGE` attribute as "true" in `$SPLUNK_HOME/etc/apps/dbx/local/props.conf`. For example:

```
[source::$sourcename$]
SHOULD_LINEMERGE=True
```

Note: For CSV and CSV (with headers) options, if you have modified the default source name in `props.conf`, but want to maintain line breaking, specify the regular expression for the `LINE_BREAKER` attribute in `$SPLUNK_HOME/etc/apps/dbx/local/props.conf`.

2. Select the **Output timestamp** check box to prefix the event with a timestamp.

3. Specify the **Timestamp column** of the table/query to use as the timestamp. If you do not specify a column, the current time is used.

4. Specify the **Timestamp format**. This is a Java SimpleDateFormat pattern. The default format is configurable during setup.

About timestamps and database output

Splunk assigns **timestamps** to event data at **index time**. In most cases, Splunk automatically recognizes and extracts timestamps from your data. If an event does not contain an explicit timestamp, Splunk tries to assign a timestamp through other means, according to specific timestamp precedence rules.

In some cases, you might need to help Splunk recognize the timestamps in your database output.

For example, when Splunk indexes your data, it looks for a timestamp of the `DATETIME` datatype. If your timestamp is a string value (such as `VARCHAR`, `NVARCHAR`, etc.), you can try to convert the timestamp to the correct datatype using a custom SQL statement with `CAST`, `CONVERT`, or `TO_TIMESTAMP` functions.

Or, if your data does not have a time reference, you can configure Splunk to use an alternate timestamp source, such as the system time when Splunk indexes your data.

Note: Incorrect timestamp formatting can cause line-breaking issues when Splunk indexes your data. For help, see Issues with bad line breaking/line merging in the Troubleshooting section of this manual.

Timestamp best practice

Follow these steps to help ensure that Splunk assigns proper timestamps to your database output at index time.

1. Look at your data. Is the time established in your source data?

- If yes, then Splunk might be able to assign the timestamp from your data. Proceed to step 2.
- If no, then skip ahead to step 3.

2. Is there a column with the time in it?

- If yes, is that column configured as a `DATETIME` datatype?
  - If yes, then Splunk assigns the timestamp from this column to your data.
  - If no, then try to use an SQL statement to convert the timestamp to the `DATETIME` datatype (using `CAST` or `CONVERT`) functions. If this doesn't work, you can try specifying the timestamp parse format in `$SPLUNK_HOME/etc/apps/dbx/inputs.conf`, as shown in this workaround.
  - **Note:** Do not assign a timestamp to a rising column.

3. Is the time established outside of your data?

- If no, then you can configure Splunk to set the timestamp to the time Splunk indexes the data, as follows:
  - Go to **Database Inputs > Add New > Outputs > Output Format**, then check **Output timestamp** and leave the **Timestamp column** field blank. Splunk uses the current time as timestamp as it indexes your data.

You can configure how Splunk recognizes timestamps in your data by editing timestamp attributes in `$SPLUNK_HOME/etc/apps/dbx/local/props.conf`. See "Edit timestamp properties in props.conf".

Specify input query interval

Specify an **Interval** for your database queries. This is the amount of time Splunk DB Connect waits between queries. If you leave the **Interval** field

blank, DB Connect chooses a time interval based on the amount of data fetched.

You can specify the **Interval** using a relative time expression or a valid `cron` expression.

Relative time expressions

A relative time expression specifies the amount of time between each input query. The syntax for relative time expressions is:

```
<integer><time_unit>
```

Relative time units are specified as seconds (s), minute (m), hour (h), day (d), week (w), month (mon), quarter (q), and year (y). For example, if you want your input query to run every 15 minutes, you would enter `15m`.

Cron expressions

Cron expressions let you schedule your input queries to run on a recurring basis. Cron expressions typically consist of five or six fields that encode a time specification similar to this:

minutes (0-59) | hours (0-23) | date of month (1-31) | month of year (1-12) | day of week (0-6, 0=Sunday)

For example, if you want your input query to run every Sunday at 3:30 AM, you would enter a cron expression such as:

```
30 3 * * 0
```

This literally translates into: "the 30th minute, of the 3rd hour, of any date of the month, of any Month of the year, on Sunday."

Note: When you specify an **Interval** value, DB Connect stores that value in the `inputs.conf` file, under the `$SPLUNK_HOME/etc/apps/dbx/local` directory. You can edit the interval value inside `inputs.conf`, under the `dbmon` stanza that specifies the input. See inputs.conf.spec

Set up a lookup table

Splunk DB Connect lets you define a **lookup table** that uses an external database as its source. For more information on lookups, see "About lookups and field actions".

To set up a database lookup table:

1. In Splunk Web, go to **Settings > Lookups > Database Lookups.** Click **Add new**.

2. Enter a **Lookup Name**, then choose your **Database** from the menu.

3. Enter the **Database Table** name.

4. At this point you have two options:

- **Specify Lookup Fields directly**: Click the **Fill all columns** button to bring in all columns from the table. Or specify the individual fields/columns to be used in the lookup. (You can fill all the columns, then use the **Delete** button next to each field to trim the list.)
- **Use an SQL query to pull data in**: Check **Configure advanced Database lookup settings**, then define a SQL query, using `$input_field$` as a placeholder for each input field value.

5. Click **Save**.

This creates a scripted lookup definition, which you can use inside Splunk as if it were a regular lookup by using the `| lookup` command.

Note: By default, only one match is returned from the database for each lookup input row. If you want to return more than one row, you must edit `max_matches` in `dblookup.conf`, as shown in the following section: "Create a lookup by editing dblookup.conf."

You can also configure an automatic lookup. For information on automatic lookups, see this topic in the Splunk Enterprise platform documentation.

Create a lookup by editing dblookup.conf

You can also create a lookup by editing the dblookup.conf file. This is useful if you have a table with many columns that would be cumbersome

to select using Manager. This requires that you also create the lookup definition manually in `transforms.conf` with `external_cmd = dblookup.py <name from dblookup.conf>`

By default, only one match is returned from the database for each lookup input row. If you want to return more than one row, you must change `max_matches` in `dblookup.conf`.

Lookups and Splunk DB Connect in a distributed environment

Some constraints exist when using Splunk DB Connect to perform lookups in a distributed Splunk environment.

- If you are running DB Connect in a distributed environment, you must perform lookups on the search head where Splunk DB Connect is installed. To perform a lookup locally, add `local=1` after the `lookup` command.

Example:

```
index=test | lookup local=1 mysql_table ip_address as clientip
OUTPUT host | table clientip, host
```

- Automatic lookups are not supported.

Note: To perform database lookups in a distributed search environment, you must install the DB Connect app on a search head. For instructions on installing apps in a search head pooling environment, see "Create a search head pool". For instructions on configuring search head pooling for Splunk DB Connect, see "Set up search head pooling".

Lookups and Datatypes

Splunk typically only sends CSV data to a lookup, so dblookup receives everything as a string. DB Connect can do datatype conversion under one of these two conditions:

- the database/JDBC driver you're using supports parameter metadata. This means that it can analyze the SQL you're about to execute and can tell which datatype is expected for each placeholder beforehand. Some JDBC drivers don't support this. Even if the JDBC driver supports it, in most cases it requires another round trip to the database for analyzing the SQL and therefore costs performance.

- the datatype is specified in the SQL template for the lookup. DB connect will generically convert the values it receives from Splunk to the datatype specified in the parameter placeholder. The syntax for specifying those datatypes is as follows:

  `$<fieldname>[:<DATATYPE>]$`

The datatype portion is optional. If it's not supplied, then DB Connect will try to use parameter metadata from the JDBC API. If it's not possible it will fallback to simply supply String values.

For example:

```
SELECT FOO FROM BAR WHERE _time = $_time:TIMESTAMP$ and src_ip =
$src_ip:VARCHAR$
```

This will force DB Connect to supply an actual timestamp value and to not rely on datatype conversion of the database or JDBC driver. A list of datatypes that can be used can be found here: http://docs.oracle.com/javase/6/docs/api/java/sql/Types.html (not all of them are fully supported). For the TIMESTAMP datatype, the value is expected to be in epoch format (for fields other than _time, it might be necessary to strptime them in the search).

Security and access controls

Splunk's role-based access controls let you setup access permissions for individual users in DB Connect. You can grant a user global access to all DB Connect features and available database connections, or limit a user's access to a specific database connection only. (You should have already set up and configured your database, including defining database users.)

If you have not yet setup your database connection in Splunk DB Connect, follow the steps described in Manage a database connection earlier in this manual.

Note: To set the database connection to read-only access for all users that have permission to use the connection, check **Read only** at the bottom of the **Add new external database** page. Check **Validate Database Connection** to verify a successful connection to the database.

Read/Write permissions refer to configuration file access, not resource access. Set permissions to "read" to give a role permission to access a resource. Without read permissions, the role cannot use DB Connect or its commands. That is, for the `dbquery` command, read permission simply

means you can use the command, including `INSERT`, not that you can only do `SELECT`.

Set permissions to "write" to give the administrator permission to modify the database configuration.

For more information, see Set Permissions in the Splunk Enterprise platform documentation.

Admin only control

In Splunk DB Connect 1.1.2 and later, only the Admin role can:

- Create and see the **Database Inputs** and **Database Lookups** pages;
- Run the `dbmonpreview` command.

To provide user permissions for database lookups, see Set up user lookup permissions.

Set up user access permissions

DB Connect provides a dbx_user role. Admins can assign the dbx_user role to non-admin users to allow the user to access all features and database connections inside the app.

To set up individual user access permissions for DB Connect:

1. Create a role for the specific user, for example "new_role_1". Do not inherit any roles for this role.
2. Click **Save**. The new role appears in the list.
3. Create a new user, for example "new_user," and assign the following roles: **user, dbx_user,** and **new_role_1.**
4. Click **Save**. The "new_user" appears in the list. The user can now access the DB Connect app in general, other apps, and depending on permissions, specific database connections (see the following section, "Set up user access to a specific database").

Note: Old user permission settings from Splunk DB Connect 1.1.0 do not work with DB Connect 1.1.2. Follow the instructions on this page to properly configure user permissions for DB Connect 1.1.2.

Restrict user access to a specific database

By default, users assigned the dbx_user role can access all database connections inside the app. To restrict user access to a specific database connection, you must assign each user to a unique role for each database connection.

For example, if you have setup individual connections to MySQL and MSSQL databases, you can restrict user access to the respective database connections as follows:

1. Create a new role for the MySQL database, for example "role_mysql_1."

2. Create a new user, for example "user_1". Assign user_1 the following roles: user, dbx_user, and role_mysql_1.

This allows user_1 to access other apps, the DB Connect app, and the MySQL database connection.

3. Create a new role for the the MSSQL database, for example, "role_mssql_1."

4. Create a new user, for example "user_2." Assign user_2 the following roles: user, dbx_user, and role_mssql_1.

This allows user_2 to access other apps, the DB Connect app, and the MSSQL database connection.

5. In Splunk Web, go to **Apps > Manage Apps > Splunk DB Connect > View Objects**.

6. Locate the mysql database and click **Permissions**. Uncheck Read/Write for dbx_user. Check Read/Write for role_mysql_1.

7. Locate the mssql database and click permissions. Uncheck Read/Write for dbx_user. Check Read/Write for role_mssql_1.

Verify database access

1. Log into Splunk as user_1.

2. Go to **Apps > Splunk DB Connect > Manage Database Connections**.

user_1 now sees the mysql database connection only. The mssql database connection is no longer visible.

3. Repeat steps 1 and 2 above to verify restricted access for user_2.

Note: Any database connection that does not have a unique role and assigned user will remain visible to all users assigned the dbx_user role.

For more information, see "About users and roles" in the Splunk Enterprise documentation.

Restrict user access to a specific index only

If you create a database input and the data is indexed by Splunk, any user that has access to the index can see the data, regardless of the user's database access permissions. You can however restrict user access to a specific index, as follows:

1. Create a new index, for example "new_index."
2. Create a new role, for example "new_role." Do not inherit any roles for this role.
3. On the **Add new role** page, add new_index to **Indexes searched by default**.
4. By default, "user" and "dbx_user" roles have access to all non-internal indexes. To restrict access to the appropriate index only, you must edit these roles, as follows: Go to **Settings > Access Controls > Roles**. For both user and dbx_user roles, under **Indexes** remove "all non-internal indexes", then add "new_index." Click **Save**.
5. Create a new user, for example "new_user." Assign the following roles to new_user: user, dbx_user, and new_role.
6. Go to **Settings > All configurations**. Locate the specific database connection for which you wish to provide the user access. Click **Permissions**. Uncheck **Read/Write** for dbx_user role, then check **Read** for new_role.

The new_user can now access the new_index only.

For more information, see Set up multiple indexes and Set Permissions in the Splunk Enterprise platform documentation.

Set up user lookup permissions

While only Admins can create database lookups in the Splunk DB Connect UI. Admins can enable users to access and use specific database lookups in searches by assigning the user role (with appropriate read/write permissions) to the database lookup.

You can set up user lookup permissions as follows:

1. In Splunk Web, go to **Settings > All configurations**.

2. Locate the specific lookup in the list of items.

3. Click **Permissions**. This opens the **Permissions** window for the lookup.

4. Assign **Read**/**Write** permissions to the user role.

Note: The Admin might first need to create a new role for the user, then assign the user to that role, and assign that role to the lookup, as shown above.

Set up search head pooling

> **Note:** The search head pooling feature has been deprecated as of Splunk Enterprise version 6.2. This means that although it continues to function, it might be removed in a future version.

If you're using search head pooling with Splunk DB Connect, you must run the `dbx_shpinst.py` script against each search head for each database connection to ensure that the database password is encrypted with the Splunk secret key for that particular search head.

While you can install and configure Splunk DB Connect on a single search head, in a pooling environment, the app state is written to shared storage and is visible to all search heads.

In addition, the Java Server Bridge will work only on the search head on which the database connection is configured (because the password is encrypted with that particular search head's secret key).

To make the Java Bridge Server work on all search heads, you must run the `dbx_shpinst.py` script against each search head and each database connection. See example below.

Set up search head pooling for Splunk DB Connect

These instructions assume you have already created a search head pooling environment. If you have not yet done so, see Create a search head pool for complete instructions.

To setup search head pooling for Splunk DB Connect:

On each search head:

1. Install JRE in the same location. The java path must be the same on each search head. (To see if your database requires JDBC driver installation, see Install database drivers.)
2. Make sure the Java Bridge Server port is open in `$SPLUNK_HOME/etc/apps/dbx/local/java.conf`

On any search head:

1. Install and configure the Splunk DB Connect app. This includes creating a database connection for each database. For instructions on how to install apps in a search head pooling environment, see Create a search head pool. **Note:** The database connection name will be the same on each search head.
2. Run the `dbx_shpinst.py` script against each search head, as follows:

```
./splunk cmd python
<path_to_shared_storage>/etc/apps/dbx/bin/dbx_shpinst.py
<searchHeadHost>:<searchHeadHostPort> --user <userName>
--targetuser <targetUserName> --db <databaseName>

splunk password:
database password:
```

The *`userName`* must belong to the **Admin** role and *`targetUserName`* must have permission to access *`databaseName`*.

If `--user` is `admin`, you don't need to specify `--targetuser`. The `--targetuser` is the Splunk user (not database user) under whose context a database configuration is stored. By default, `--targetuser` is "nobody." Specify `--targetuser` only if you need to locate a db configuration that is stored in a specific Splunk user context. For example:

```
./splunk cmd python
<path_to_shared_storage>/etc/apps/dbx/bin/dbx_shpinst.py
localhost:8089 --user julian --targetuser admin --db oracle

splunk password:
database password:
```

Note: You must rerun the `dbx_shpinst.py` script against each search head for every subsequent database password change.

Example

This example shows you how to setup search head pooling for a Splunk DB Connect deployment that includes 3 search heads and 2 database connections.

First, we setup search head pooling for our 3 search heads, as shown in Create a search head pool.

Next, we create 2 database connections on search head 1. The other two search heads pick up the database configuration from shared storage used by search head pooling.

Here we see the configuration of each database connection in `/<path_to_shared_storage>/etc/apps/dbx/local/database.conf`:

```
[MSSQL]
database = dbxtest
host = 10.75.0.50
password = enc:CDr9SiQKgXhss4JDRxb7vQ==
readonly = 1
type = mssql
username = sa

[mysql]
database = orders
host = 10.75.0.50
password = enc:pDWbcIFP7iPt11cDHMe9Zw==
port = 9408
readonly = 1
type = mysql
username = mktadmin
disabled = 0
```

Complete the following tasks on each search head where the DB Connect app will be used to connect to the above databases:

1. Execute the `dbx_shpinst.py` script from search head `$SPLUNK_HOME/bin`, as shown:

```
./splunk cmd python
/<path_to_shared_storage>/etc/apps/dbx/bin/dbx_shpinst.py
<searchHeadHost>:<searchHeadHostPort> --user <userName> --db
<databaseName>
```

For example:

```
./splunk cmd python
/<path_to_shared_storage>/etc/apps/dbx/bin/dbx_shpinst.py
splunksh01:8089 --user admin --db MSSQL
```

You will then be prompted to enter passwords for `splunk user (--user)` and `database user`.

Once `dbx_shpinst.py` has successfully executed, the following message appears:

```
Password at <searchHeadHost> set successfully.
```

Note: You must repeat step 1 on each search head for each database connection.

2. Verify that a `distributed.conf` file has been created under the search pool location `/<path_to_shared_storage>/etc/apps/dbx/local/`.

After repeating step 1 for all 3 search heads and 2 database connections, our `/<path_to_shared_storage>/etc/apps/dbx/local/distributed.conf` should look like this:

```
[MSSQL@splunksh01]
password = enc:CDr9SiQKgXhss4JDRxb7vQ==
readonly = 1

[MSSQL@splunksh02]
password = enc:pDWbcIFP7iPt11cDHMe9Zw==
readonly = 1

[MSSQL@splunksh03]
password = enc:WE44Q8qoC8Lm8roTDE5SvQ==
readonly = 1

[mysql@splunksh01]
password = enc:QOcS2rA2GcfSjq+3HoHtTw==
```

```
readonly = 1

[mysql@splunksh02]
password = enc:EnYgSEcf+dfskTOSlcAzWw==
readonly = 1

[mysql@splunksh03]
password = enc:RbQQeXTUPYYL/FBHPMCBjQ==
readonly = 1
```

This `distributed.conf` file will be used by the db connect instances on the search heads to decrypt the database connection passwords. Since each search uses its own secret key, the password strings should be different.

Use database search commands

Splunk DB Connect provides the following commands for reading and writing to your database:

- `dbquery`
- `dbinfo`
- `dboutput`
- `dbmonpreview`

These commands are typically invoked as part of a search string with the following format:

```
index=<myIndexName> | <DBConnectCommand>
```

You can also invoke the `dbquery` and `dbinfo` commands via the **Explore database schema** panel in the Splunk DB Connect app UI.

Note: For the commands described below, use the backslash (`\`) character to escape a literal backslash. For example, to escape the backslash in `C:\home`, use `C:\\home`. You do not need to escape backslashes in the `\n`, `\r`, and `\t`, and similar, formatting character codes.

dbquery

The `dbquery` command queries the specified database and returns table rows as Splunk search results.

Note: The `dbinput` command is an alias for the `dbquery` command.

Note: This command is for previewing database search results, and is not intended for regular use.

Syntax

```
dbquery <sql-database> <dbquery-sql>
```

Arguments

Argument	Description
`sql-database`	Name of a configured database listed in the `database.conf` file.
`dbquery-sql`	The `SELECT` query string to execute. Format options: "databaseName" `db`=databaseName `database`=databaseName

Example

```
| dbquery "mysql" "SELECT * FROM hosts WHERE active = 1" limit=25
```

Note: There is no "limit" argument (or default limit) for the number of rows the `dbquery` command returns. The `limit` command in the above example is a Splunk search command, which demonstrates how to return a more manageable (smaller) set of rows, in case your database table has many hundreds or thousands of rows. (You can use DB Connect and Splunk commands in the same search string.)

dbinfo

The `dbinfo` command retrieves database/table schema information as search results. The command relies on the JDBC generic metadata mechanism.

Note: The DB Connect UI includes a **DB Info** view that lets you interactively inspect the structure of your database. Use the **DB Info** view as a convenient alternative to running the `dbinfo` command directly.

Syntax

```
dbinfo type=<dbinfo-type> database=<sql-database>
(table=<sql-table>) [flags...]
```

Arguments

Argument	Description
`type`	Type of information to retrieve: `tables` = All specified database table names; (see `flags` qualifier) `columns` = All column information for specified table; (see `flags` qualifier) `schemas` = All database schema names `size` = Specified table size (TBD) information
`database`	Name of a configured database listed in the `database.conf` file.
`table`	Name of database table.
`flags`	The `type` option is qualified by by the following flag options: `tables`: `fetchSize=` get table size (`true` or `false`) `schemas=` All (`"*"`), or table schema name (Optional) `includeViews=` `true` or `false` `columns`: `table=` database table name (Optional) `forceRefresh=` Refresh first (`true` or `false`) `schemas`: (No additional qualifying arguments) `size`: `table=` database table name (Optional) `forceRefresh=` Refresh first (`true` or `false`)

Example

```
|  dbinfo type=columns database=mysql table=general_log
```

Note: The `dbinfo` command does not take an index (e.g. index=_internal) as the first part of the search string. Rather, you must specify the database you wish to access, as shown in the above example.

dboutput

Caution: The `dboutput` command overwrites existing database entries so use with caution.

The `dboutput` command provides database write capability. The command

updates or creates records in the specified database table, for each result.

Note: The `dboutput` command is compatible with historical (non real-time) search queries only. Historical search queries use time ranges that are not real-time, such as -15m or -1d. The `dboutput` command is not compatible with real-time searches.

Syntax

```
dboutput type=<dboutput-mode> [streaming=<false|true>]
(database=<sql-database>) ("<sql-statement>") |
(table=<sql-table>) (key=<field>|keyField=<field>
keyColumn=<sql-column>)? (notFound=<string>)? ((<field>( as
<sql-column>)?)+|"*")
```

Arguments

Argument	Description
`type`	Output mode: `insert` = Insert record `update` = Update existing record
`streaming`	(Optional) Stream data flag: `true` = Stream data. Transfers data in multiple segments, so the process runs in multiple transaction with each segment representing a transaction. If any segment fail, transfer continues without recovering the failed segment, which may result in the loss of a record in one transaction. Use this for less critical data, such as performance metrics. `false` = (Default, the same as not specifying the argument) Output is limited to 50,000 search results.
`database`	Name of a configured database listed in the `database.conf` file.
`"<sql-statement>"`	SQL write string.
`table`	SQL table name
`key`	(Optional) Only applies to `type=update`. The key columns to use for the `SQL UPDATE` statement.
`keyField`	(Optional) SQL table key field, used with `keyColumn` option (if not using `key` option)

`keyColumn`	(Optional) SQL table key column, used with `keyField` option.
`notFound`	(Optional) Behavior if key is not found: `insert` = Insert result, instead `ignore` = Do nothing `fail` = Rollback changes, if possible, and fail the execution
`<field> as`	(Optional) Fields used for the update in the form of <fieldName> [AS <columnName>]. Use * as a wildcard.

Examples

Example 1

```
index=_internal | dboutput type=insert database=ASSET_DB
table=jobs key=jobname host=myAssetsHost
```

Example 2

```
index=_internal | dboutput ASSET_DB table=jobs columns
```

Equivalency examples

These two examples are equivalent:

```
index="_internal" | dboutput database=mydb type=sql "INSERT INTO
udata_test (host, sourcetype) VALUES ($host$, $sourcetype$)"

index="_internal" | dboutput database=mydb type=insert
table=udata_test host sourcetype
```

These two examples are also equivalent:

```
index="_internal" | dboutput database=mydb type=sql "UPDATE
udata_test SET host=$host$, sourcetype=$sourcetype$ WHERE
uid=$uid$"

index="_internal" | dboutput database=mydb type=update
table=udata_test key=uid host sourcetype
```

Example 3

Note: Use care when piping database queries through search commands as some commands might result in invalid SQL character sequences. For example, in the following search, for the query to work properly, you must

`rename` the `stats` field, as shown:

```
index=_internal | stats dc(source) dc(sourcetype) | rename
dc(source) AS dcs dc(sourcetype) AS dct | dboutput database=mysql
type=sql "INSERT INTO t1 (a, b) VALUES ($dct$, $dcs$)"
```

dbmonpreview

The `dbmonpreview` command simulates the output of a database input.

1. Create a database input in `inputs.conf`, as shown in Configure input query parameters.

This creates a `dbmon-` stanza in the `inputs.conf` file.

2. Use the `dbmonpreview` command to preview the input.

Syntax

```
dbmonpreview <stanza> (<key>=<value>)
```

Arguments

Argument	Description
`stanza`	Input stanza name, prefixed with: `dbmon-`
`key`	SQL database key

Example

```
| dbmonpreview dbmon-dump://mysql/t1
```

For more information on using Search, see the Splunk Search Manual in the Splunk Enterprise documentation.

Troubleshooting

Troubleshooting

This topic describes how to troubleshoot common Splunk DB Connect issues.

Answers

Have questions? In addition to these troubleshooting tips, visit Questions related to Splunk DB Connect to see what questions and answers the Splunk community has about using Splunk DB Connect.

Java Bridge Server doesn't start after an upgrade of Java

An error appears in jdbrige.log indicating that Java cannot find the correct cipher suite

2015-03-18 09:37:40,269 ERROR Java process returned error code 1! Error: Initializing Splunk context... Environment: SplunkEnvironment{SPLUNK_HOME=D:Program FilesSplunk,SPLUNK_DB=D:Program FilesSplunkvarlibsplunk} Configuring Log4j... Exception in thread "main" com.splunk.config.SplunkConfigurationException: IO Error while reading configuration from Splunkd: javax.net.ssl.SSLHandshakeException: No appropriate protocol (protocol is disabled or cipher suites are inappropriate) at com.splunk.config.rest.RESTAdapter.request(RESTAdapter.java:195) at com.splunk.config.rest.RESTAdapter.readConfig(RESTAdapter.java:203) at com.splunk.config.cache.CachedConfigurationAdapter.readConfig(CachedConfigurationAda at com.splunk.config.cache.CachedConfigurationAdapter.readStanza(CachedConfigurationAd at com.splunk.env.SplunkContext.getConfigStanza(SplunkContext.java:313) at com.splunk.env.SplunkContext.initialize(SplunkContext.java:128) at com.splunk.bridge.JavaBridgeServer.main(JavaBridgeServer.java:34) Caused by: javax.net.ssl.SSLHandshakeException: No appropriate protocol (protocol is disabled or cipher suites are inappropriate) at sun.security.ssl.Handshaker.activate(Unknown Source) at

sun.security.ssl.SSLSocketImpl.kickstartHandshake(Unknown Source) at sun.security.ssl.SSLSocketImpl.performInitialHandshake(Unknown Source) at sun.security.ssl.SSLSocketImpl.startHandshake(Unknown Source) at sun.security.ssl.SSLSocketImpl.startHandshake(Unknown Source) at sun.net.www.protocol.https.HttpsClient.afterConnect(Unknown Source) at sun.net.www.protocol.https.AbstractDelegateHttpsURLConnection.connect(Unknown Source) at sun.net.www.protocol.https.HttpsURLConnectionImpl.connect(Unknown Source) at com.splunk.rest.Splunkd.request(Splunkd.java:212) at com.splunk.rest.Splunkd.request(Splunkd.java:98) at com.splunk.config.rest.RESTAdapter.request(RESTAdapter.java:193) ... 6 more

Fix: Starting with JDK 8u31 release, the SSLv3 protocol (Secure Socket Layer) has been deactivated and is not available by default. See the java.security.Security property jdk.tls.disabledAlgorithms in <JRE_HOME>/lib/security/java.security file.

If SSLv3 is absolutely required, the protocol can be reactivated by removing "SSLv3" from the jdk.tls.disabledAlgorithms property in the java.security file or by dynamically setting this Security property to "true" before JSSE is initialized.

http://www.oracle.com/technetwork/java/javase/8u31-relnotes-2389094.html

Java Bridge Server not running

A status error appears indicating that the Java Bridge Server is not running, and the `dbx.log` contains errors relating to `REST keep-alive failed`. Other symptoms might include searches that return column names with no or incomplete data. This typically occurs when Splunk DB Connect is running in a VM that has been suspended and restarted. Also, a stale `state.xml` file can prevent the Java Bridge Server from running.

Fix: If you suspend and restart a VM on which the Java Bridge Server is running, make sure to restart Splunk. Also, remove any stale `state.xml` files from `$SPLUNK_HOME/var/lib/splunk/persistentstorage/dbx`.

After upgrading to DBX 1.1.6, Java Bridge Server does not appear to be running. `jbridge.log` shows `ERROR Java process returned error code 1! Error: Initializing Splunk context... Environment: SplunkEnvironment{SPLUNK_HOME=/opt/splunk,SPLUNK_DB=/opt/splunk/var/lib/spl`

```
unk} Configuring Log4j... Exception in thread "main"
com.splunk.rest.SplunkdException: Unable to connect to Splunkd
REST Service: Connection refused
```

Fix: Ensure the supported Oracle JRE is in use as specified in our requirements.

Note: Only the Oracle JRE is certified and supported for use with Splunk DB Connect. Customers have reported problems when starting the Java Bridge Server under alternate JREs or JDKs such as OpenJDK or IBM Java.

Input not updating

For a `dbmon-tail`, check the latest checkpoint value, which is stored in the `$SPLUNK_DB/persistentstorage/dbx` directory. (`$SPLUNK_DB` is the `$SPLUNK_HOME/var/lib/splunk` directory, if not otherwise defined.) Each input has its own directory, which is a hash of its name and a 32-character hexadecimal string. This directory typically contains these files:

- `manifest.properties` has meta-information, such as the input name.
- `state.xml` has the actual state in XML format.

1. Identify the state directory.
2. inspect the XML file.

The state file looks like this:

```
<list>
  <value key="latest.record_update">
    <value class="sql-timestamp">2012-12-07 04:22:25.703</value>
  </value>
</list>
```

Putting DBX.log in DEBUG mode

To enable debug-level logging for DBX, edit the [logging] stanza in java.conf:

```
[logging]
level = DEBUG
file = dbx.log
console = false
```

```
logger.com.splunk.dbx = DEBUG
```

Error creating PersistentValueStore

Note: You might encounter this error after upgrading from earlier DB Connect versions.

If this error appears in the `jbridge.log` file, you might have a corrupted persistent store file.

```
ERROR Java process returned error code 1! Error: Initializing
Splunk context...
Environment:
SplunkEnvironment{SPLUNK_HOME=/opt/splunk,SPLUNK_DB=/opt/splunk/var/lib/splunk}
Configuring Log4j... [Fatal Error] :1:1: Premature end of file.
Exception in thread "main"
com.splunk.config.SplunkConfigurationException: Error creating
PersistentValueStore type xstream:
com.thoughtworks.xstream.io.StreamException:  : Premature end of
file.
```

To resolve this issue, remove `$SPLUNK_DB/persistentstorage/dbx/global`, recursively.

Issues with bad line breaking/line merging

The problem is caused by Splunk linebreak heuristics. Typically, log file data includes event timestamps, which Splunk understands. If you have timestamps in your database rows, you'll avoid linebreak issues. Be sure to set output timestamp and specify that the timestamp column is the actual timestamp column.

If you don't have timestamps in your db rows

If you don't have timestamps in your database rows, you have two options:

- Click output timestamp and leave the timestamp column blank. Splunk outputs the current time when indexing.
- Use the default sourcetype in the input config. Leave it blank because Splunk DB Connect uses `dbmon:kv` as the sourcetype (in the normal case where you're using the key-value output format). But, if you put something custom in the `sourcetype` field, you must tell Splunk how to linebreak for that sourcetype. Copy the `props.conf` settings for the default stanzas - specifically, add `"SHOULD_LINEMERGE = false"`.

If your timestamp is not of type datetime/timestamp

Splunk DB Connect expects the timestamp column in your database to be of type `datetime/timestamp`. If it is not (for example, it is in format char/varchar/etc.), you can first try to convert the SQL statement into the correct type using `CAST` or `CONVERT` functions. If this method doesn't work, you can use the following workaround:

Check the **Output timestamp** box and specify the `output.timestamp.parse.format` so DB Connect can obey the timestamp output format setting. For example, if the database column `EVENT_TIME` has strings, such as CHAR, VARCHAR, or VARCHAR2, with values like `01/26/2013 03:03:25.255` you must specify the parse format in the appropriate copy of `inputs.conf`.

```
output.timestamp = true
output.timestamp.column = EVENT_TIME
output.timestamp.parse.format = MM/dd/yyyy HH:mm:ss.SSS
```

Unexpected session key expiration

A system clock change or suspend/resume cycle can cause unexpected session key expiration. To remedy the problem, restart the Splunk system using DB Connect. If it does not come back cleanly, delete the state file, `$SPLUNK_DB/persistentstorage/dbx/global/state.xml`, and restart the Splunk system, again.

Java bridge log file settings

By default the Python Java bridge process logs INFO-level events to the `jbridge.log` file, using a rolling file appender over five files for a maximum of 100M bytes.

You can create a `jbridge_server.conf` file in the `$SPLUNK_HOME/etc/apps/dbx/local` directory to override those settings.

Example `jbridge_server.conf` file entry:

```
[log]
filename=jbridge_conf.log
maxCount=10
```

```
fileSize=1000000000000
logLevel=debug
```

The DB Connect homepage keeps refreshing

This issue is caused by a known bug (DBX-317), which affects users that are logged in with the capabilities of a role that inherits from the default dbx_user role.

To workaround, give users the dbx_user role directly, rather than assign them a role that inherits from dbx_user

Cannot connect to any database: SSL Errors

If you cannot connect to a database and see similar errors in jbridge.log to the following, check to ensure that you are not running in FIPS mode. FIPS mode is not compliant with the jbridge service and is not supported. About FIPS Mode

ERROR Java process returned error code 1! Error: Initializing Splunk context... Environment: SplunkEnvironment{SPLUNK_HOME=/u01/splunk,SPLUNK_DB=/u01/splunk/var/lib/splunk} Configuring Log4j... Exception in thread "main" com.splunk.config.SplunkConfigurationException: IO Error while reading configuration from Splunkd: javax.net.ssl.SSLHandshakeException: Received fatal alert: handshake_failure at com.splunk.config.rest.RESTAdapter.request(RESTAdapter.java:195) at com.splunk.config.rest.RESTAdapter.readConfig(RESTAdapter.java:203) at com.splunk.config.cache.CachedConfigurationAdapter.readConfig(CachedConfigurationAda at com.splunk.config.cache.CachedConfigurationAdapter.readStanza(CachedConfigurationAd at com.splunk.env.SplunkContext.getConfigStanza(SplunkContext.java:313) at com.splunk.env.SplunkContext.initialize(SplunkContext.java:128) at com.splunk.bridge.JavaBridgeServer.main(JavaBridgeServer.java:34) Caused by: javax.net.ssl.SSLHandshakeException: Received fatal alert: handshake_failure at sun.security.ssl.Alerts.getSSLException(Alerts.java:192) at sun.security.ssl.Alerts.getSSLException(Alerts.java:154) at sun.security.ssl.SSLSocketImpl.recvAlert(SSLSocketImpl.java:1781)

Cannot connect to Microsoft SQL server

If you cannot connect to a Microsoft SQL server, verify that you are using the correct driver, host, and port.

- **Driver:** MSSQL is the correct driver to use for Microsoft SQL servers. ODBC does not work as effectively.
- **Host:** To specify a host for Microsoft SQL, use a fully qualified domain name, a short name, or an IP address. Do not use the Microsoft SQL convention of `<SERVERNAME\DATABASE>` for the host field.
- **Port:** Many Microsoft SQL Servers use dynamic ports instead of TCP/1433. Work with your database administrator to identify the correct port, or see "Verifying the port configuration of an instance of SQL Server" here.

Cannot connect to Oracle SQL Server

If you receive an error attempting to connect to an Oracle DB, note the following:

Oracle Error Codes The most common error codes are:

- ORA-12504: TNS:listener was not given the SID in CONNECT_DATA

This error means that the SID was missing from the CONNECT_DATA configuration. To troubleshoot, check that the connect descriptor corresponding to the service name in TNSNAMES.ORA also has an SID component in the CONNECT_DATA stanza.

- ORA-12505: TNS:listener does not currently know of SID given in connect descriptor

You are receiving this error because the listener received a request to establish a connection to the Oracle DB, but the SID for the instance either has not yet dynamically registered with the listener or has not been statically configured for the listener. Typically, this is a temporary condition that occurs after the listener has started, but before the database instance has registered with the listener. To troubleshoot, try waiting a few moments and try the connection again. You should also check which instances are currently known by the listener by executing: lsnrctl services <listener name>

- ORA-12514: TNS:listener does not currently know of service requested in connect descriptor

This error is because the listener received a request to establish a connection to the database. The connection descriptor received by the listener specified a service name for a service that either has not yet dynamically registered with the listener or has not been statically configured for the listener. To troubleshoot, try waiting a few moments and try the connection again. You should also check which instances are currently known by the listener by executing: lsnrctl services <listener name>

Explanation of Oracle TNS Listener and Service Names

TNS is a proprietary protocol developed by Oracle. It provides a common interface for all industry-standard protocols and enables peer-to-peer application connectivity without the need for any intermediary devices.

DBX utilizes Java (via the JDBC driver) to connect Splunk to a TNS Listener, which in turn connects to the Oracle Database. You can configure DBX to connect via the Service Name or the Oracle SID. Typically, most connectivity issues with DBX and Oracle Databases are caused by misconfiguration of the TNS Listener.

Database login error from search head pool

If you receive a database login error when using search head pooling with DB Connect:

- Hit the distributed REST endpoints and confirm that each database connection on each search head has a unique password set. The REST endpoints are as follows:

```
https://<search_head_hostname>:8089/servicesNS/nobody/dbx/dbx/databases
https://<search_head_hostname>:8089/servicesNS/nobody/dbx/dbx/distributed
```

Problem upgrading from earlier DB Connect versions

To ensure a successful upgrade of DB Connect, we recommend that you stop the Java Bridge Server and make a backup of your local directory, as follows:

1. Stop the Splunk instance on which the Splunk DB Connect app is running.

2. Make a backup of your `$SPLUNK_HOME/etc/apps/dbx` directory.

3. Delete the original `$SPLUNK_HOME/etc/apps/dbx` directory.

4. Start the Splunk Enterprise instance.

5. Perform a fresh install of the latest version of Splunk DB Connect, as shown in steps 1-4 of Install the Splunk DB Connect App.

6. Once you have successfully installed the DB Connect app, copy all `.conf` files from your `$BACKUP_DIR/etc/apps/dbx/local` directory into your new `$SPLUNK_HOME/etc/apps/dbx/local` directory.

7. Start Splunk.

Note: After upgrading DB Connect, you might encounter this error creating PersistentValueStore.

Renaming rising_column breaks database input

Renaming the `rising_column` causes a "catch-22." If you rename the `rising column`, DB Connect returns an exception stating that no such column exists in the original table. If you set the `rising_column` to the unrenamed column name that is in the table, DB Connect returns an exception stating that there is no such field in the final output.
Workaround: Do not rename the `rising_column` field.

Java Bridge Server does not work after upgrading JDK

If you update the version of JDK running on your Linux server without updating the version of JDK that Splunk DB Connect references in **Java Home**, the Java Bridge Server will not work, and an "Error getting database connection: Pool not open" message appears in the `dbx.log` file.
Fix:

1. Open the Splunk DB Connect App.

2. Go to **Settings > Splunk DB Connect configuration**.

The Java setup page opens.

3. In the **Java Home** field, update the version of JDK in the path name, so that it matches the version of JDK currently running on the server.

inputs.conf stanzas must match in default and local directories

`inputs.conf` stanzas in `/default` and `/local` directories must match for DB Connect to function properly. This is because entries in `default/inputs.conf` are disabled by default and overridden by the corresponding stanza in `local/inputs.conf`.

For example, on Windows, if a user has script stanzas, such as `[script://.\bin\jbridge_server.py]` in `default/inputs.conf` and `[script://D:\Splunk\etc\apps\dbx\bin\jbridge_server.py]` in `local/inputs.conf` the passAuth parameter specified in `default/inputs.conf` is not inherited to `local/inputs.conf` and a failure occurs.

Timestamp column displays in epoch time instead of datetime

When using dbquery to query a database table in preview mode, the timestamp column displays in epoch time, instead of human readable and Splunk-recognized datetime (MM/DD/YYYY HH:MM:SS).

Workaround: Use a SQL statement in your query to convert epoch time to datetime. The specific SQL command depends on the database. For example, to convert epoch time to datetime, use the following statement:

```
SELECT FROM_UNIXTIME(epoch timestamp, optional output format)
```

The default output is YYYY-MM-DD HH:MM:SS (DBX-748)

For more information on configuring timestamps in DB Connect, see About timestamps and database output.

Queries containing AS do not change column names as expected

When using the AS keyword in a query such as the following:

```
SELECT xyz AS abc.xyz FROM jkl
```

DB Connect returns the column name as xyz instead of as abc.xyz. This is because the JDBC specification states that a column name is not changed

by the AS keyword; it always returns the actual name of the column.

Configuration file reference

Configuration file reference

Splunk DB Connect includes the following custom configuration spec files:

- database.conf.spec
- database_types.conf.spec
- dblookup.conf.spec
- java.conf.spec
- inputs.conf.spec

The most current versions of these spec files are located in `$SPLUNK_HOME/etc/apps/dbx/README`.

database.conf.spec

```
# Copyright (C) 2005-2012 Splunk Inc. All Rights Reserved.
# The file contains the configured database connections

[<name>]

host = <string>
* The IP address or the hostname of the database.

port = <integer>
* The port number of the database. If omitted the default port
number for the given database type is used.

username = <string>
* The username which is used for authenticating against the
database.

password = <string>
* The password which is used for authenticating against the
database. It will be automatically encrypted if it is set in
* clear-text.

database = <string>
* The database name or SID.

type = <database_type>
* The database type. References a stanza in database_types.conf
```

```
readonly = true|false
* Whether the database connection is read-only. If it is readonly,
any modifying SQL statement will be blocked

database.sid = true|false
* Only applies to Oracle database connections (ie. type=oracle).
Set to *true* if the Oracle database is only reachable
* using an SID. By default the the service name format is used.

default.schema = <string>
* Sets the default schema for the database connection if the
database type supports it (Currently only Oracle supports
* it).

testQuery = <string>
* Supply a specific test query for validating connections to this
database
* If defined it overrides the testQuery of the database type (see
database_types.conf)

validationDisabled = true|false
* Turn off connection validation for this database connection
* If defined it overrides the validationDisabled of the database
type (see database_types.conf)
* Caution: disabling validation can lead to unpredictable results
when using it with connection pooling

arg.ssl = request|require
* Allows SSL connection to MSSQL database
```

database_types.conf.spec

```
# @copyright@
# This file contains the database type definitions

[<name>]

displayName = <string>
* A descriptive display name for the database type.

typeClass = <string>
* The FQCN (fully qualified class-name) of a class implementing
the com.splunk.dbx.sql.type.DatabaseType interface.

jdbcDriverClass = <string>
* The FQCN of the JDBC Driver class. Only used when no typeClass
is specified.
```

```
defaultPort = <integer>
* The default TCP port for the database type. Only used when no
typeClass is specified.

connectionUrlFormat = <string>
* The JDBC URL as a MessageFormat string. The following values
will be replaced:
* {0} the database host
* {1} the database port (the port specified in database.conf or
the default port)
* {2} the database name/catalog or SID
* Only used when no typeClass is specified.

testQuery = <string>
* A simple SQL that is used to validate the database connection.
Only used when no typeClass is specified.

supportsParameterMetaData = [true|false]
* Whether the given JDBC driver supports metadata for
java.sql.PreparedStatement.
* Only used when no typeClass is specified.

quoteChars = <string>
* Override the quote characters for the database type. If not
specified the default ANSI-SQL quote characters will be used.
* Only used when no typeClass is specified.

defaultCatalogName = <string>
* Configure the default catalog name for a generic database type.
Used for querying the catalog names (ie. databases)

local = true|false
* This flag marks a database type as local (ie. it is accessed
via the filesystem instead of TCP)

defaultSchema = <string>
* Set the default schema prefix for the database type (defaults to
null)

streamingFetchSize = <n>
* Number of results to be fetched at a time when streaming is
enabled for a JDBC statement.

streamingAutoCommit = [true|false]
* Turn auto-commit on or off for java.sql.Connection instances in
streaming mode

validationDisabled = [true|false]
* Turn off connection validation for database connections of this
type
* Defaults to false
* Caution: this can lead to unpredictable results when using this
```

```
with connection pooling
```

dblookup.conf.spec

```
# Copyright (C) 2005-2012 Splunk Inc. All Rights Reserved.
# This file contains the configured database lookup definitions

[<name>]

database = <database>
* The database. References a stanza in database.conf

table = <string>
* The database table name. Only used in simple mode (advanced =
0).

fields = <csv-list>
* A list of fields/columns for the lookup
* You can specify the field only, or both the field and the
column in the form: <field> as <sql-column>

advanced = [1|0]
* Whether to perform a simple lookup against the table or use a
custom SQL query

query = <string>
* A SQL query template. Expressions in the form of $fieldname$
are replaced with the input provided by splunk.

input_fields = <csv-list>
*  list of fields/columns for as input for the SQL query template

max_matches = <n>
* Maximum number of results fetched from the database for each
lookup input row
* Defaults to 1
```

java.conf.spec

```
# Copyright (C) 2005-2015 Splunk Inc. All Rights Reserved.
# The master configuration file. Global settings for Java and
Splunk DB Connect are
# configured in here.
```

```
#################
# Java settings #
#################
[java]

home = <path>
* Path to the Java JRE or JDK installation directory

options = <string>
* Arbitrary Java command line options
* For example memory settings or system properties

[bridge]

addr = <bind address>
* The address/interface the Java Bridge server should listen for
* connections on.
* In most cases only localhost makes sense.

port = <bind port>
* The port the Java Bridge server should listen for connections
on.

threads = <n>
* The size of the thread pool for Java Bridge command execution.
* This defines the number of commands that can run concurrently

debug = true|false
* Turn on debugging for the Java Bridge client

[logging]

level = INFO|DEBUG|WARN|ERROR|FATAL
* The global logging severity

file = <filename>
* The filename for the Splunk DB Connect logfile which is placed
at
* $SPLUNK_HOME/var/log/splunk


console = true|false
* Enable or disable STDOUT output of log events. (only for
debugging).

logger.<logger_name> = INFO|DEBUG|WARN|ERROR|FATAL
* Override the global log level for a specific logger

[splunkd]
sslVersion = <SSLv3, TLSv1>
* The ssl algorithm for the splunkd connection
```

```
[persistence]

global = <store type>
* The type used for the global persistent store.
* - xstream: Data is stored in XML flatfiles. These files are
readable
*            and easy to inspect and change
* - jdbm: Data is stored in a btree key-value database. The
performance
*         is better for big amounts of data but the files are
binary.

type.<type_name> = <fqcn>
* A type definition for a implementation of the
PersistentValueStore interface

[config]

adapter = <config_adapter>
* A class implementing the com.splunk.config.ConfigurationAdapter
interface

cache = true|false
* Enable or disable caching of configuration values

[output]

default.channel = <string>
* A channel is a way on how to get data into Splunk. Currently
this is
* done using the "spool" channel. There will be other options in
future.

default.timestamp.format = <string>
* The default format of timestamp values for events generated by
DBmon.
* The can be overridden on per-input definition basis. The format
is
* expressed as Java SimpleDateFormat pattern.

type.<type> = <string>
* Allows the registration of a output channel
*(a class implementing com.splunk.output.SplunkOutputChannel)

format.<format> = <string>
* Allows the registration of a output format
* (a class implementing
com.splunk.dbx.monitor.output.OutputFormat)

[cache]
default.type = <softref|lru>
```

```
cleaner.interval = <relative_time_expression>

[rest]
keep-alive.timeout = <relative_time_expression>

####################
# Database settings #
####################
[dbx]

database.factory = persistent|default
* The database connection factory to use

database.factory.pooled = true|false
* Enable database pooling

pool.maxActive = <n>
* The maximum number of active database connections

pool.maxIdle = <n>
* The maximum number of idle database connections

cache.tables = true|false
* Turn on caching of table metadata information

cache.tables.size = <n>
* The size of the table metadata cache

cache.tables.invalidation.timeout = <relative_time_expression>
* The amount of time before the cached metadata information of a
table is
* considered invalid and fetched again

preload.config = [true|false]
* When enabled, the database factory will fetch and check all
configured
* database on startup. Otherwise there fetched when they are used
for the
* first time.

query.stream.limit = <n>
* Force streaming results for queries with a max. result limit
greater than this (default is 10000).
* This setting affects only certain database types.

jdbc.streaming.fetch.size = <n>
* Number of results to be fetched at a time when streaming is
enabled for a JDBC statement. It can be overridden on
* a per-database-type basis using the "streamingFetchSize"
parameter in database_types.conf.
* This setting affects only certain database types
* Default is 500
```

```
[dbmon]

threads = <n>
* The size of the thread pool for database inputs

output.channel = <string>
* The output channel to use
* - spool: Temporary files, that are moved into a file monitor
sinkhole
* - rest: Events are uploaded via REST to Splunkd

output.buffer.limit = <file-size-expression>
* Only applies to the spool output channel. The max. size of the
temp files, before they are moved to the sinkhole.
* Defaults to 5MB

output.time.limit = <n>
* Only applies to the spool output channel. The time limit for
moving files to the sinkhole in milliseconds.
* Defaults to 5000

[dblookup]

cache = true|false
* When set to true, database lookup definitions are cached in
memory

cache.size = <n>
* The cache size for database lookups definitions (number of
entries)

cache.invalidation.timeout = <relative_time_expression>
* The amount of the before a database lookup definition is
considered invalid
* and removed from the cache.

[startup]
init.<n> = <FQCN>

[dboutput]

batch.size = <n>
```

inputs.conf.spec

Note: Modifying `inputs.conf` file stanzas outside of the DB Connect app, such as in the search app or manager context, is not supported.

```
# Copyright (C) 2005-2012 Splunk Inc. All Rights Reserved.
# This file contains the database monitor definitions

[dbmon-<type>://<database>/<unique_name>]

interval = auto|<relative time expression>|<cron expression>
     * Use to configure the schedule for the given database
monitor.
     * Schedule types:
          * auto - The scheduler automatically chooses an
interval based on the number of generated results.
          * relative time expression - The number of seconds or a
relative time expression.
             Examples:
               * interval = 60 (runs every 60 seconds)
               * interval = 1h (runs every hour)
          * cron expression
             Examples:
               * interval = 0/15 * * * *       (run every 15
minutes)
               * interval = 0 18 * * MON-FRI * (run every
weekday at 6pm)

query = <string>
     * The query option defines the exact SQL query executed
against the database

table = <string>
     * If a query is not specified, DBmon automatically creates a
SQL query from the given table name.
        Example: SELECT * FROM <table>.

output.format = [kv|mkv|csv|template]
     * The output format.
     * Format types:
       * kv: Simple key-value pairs.
       * mkv: Multiline key-value pairs. (Each key-value pair is
printed on its own line.)
       * csv: CSV-formatted events.
       * template: Specify the generated events using the
<output.template> or <output.template.file> options.

output.template = <string>
```

```
output.template.file = <string>

output.timestamp = [true|false]
     * Controls whether or not the generated event is prefixed
with a timestamp value.

output.timestamp.column = <string>
     * The column of the result set from which the timestamp is
fetched. If this is omitted, the monitor execution time
     * is used as the timestamp value.

output.timestamp.format = <string>
     * The format of the output timestamp value expressed as a
Java SimpleDateFormat pattern.

output.timestamp.parse.format = <string>
     * Used when the timestamp in the column defined by
<output.timestamp.column> is a string value, such as varchar or
nvarchar.
     * Lets you define a (SimpleDateFormat) pattern for parsing
the timestamp.

output.fields = <list>
     * The fields to print in the generated event.

# A Tail Database monitor remembers the value of a column in the
result and only fetches entries with a higher value
# in future executions.

[dbmon-tail://<database>/<unique_name>]

tail.rising.column = <string>
     * A column with a value that is always rising. The best
option is to use an auto-incremented value or a sequence.
     * A creation or last-update timestamp is a good choice.

tail.follow.only = [true|false]
     * If this options is set to true nothing is indexed on the
first run (default is false).
     * This only affects the first execution of the monitor.
```

dbmon examples

Example: Monitoring a database table.

```
[dbmon-tail://mySQL/MyTableTail]
output.format = kv
output.timestamp = 1
output.timestamp.column = last_update
```

```
table = actor
tail.rising.column = actor_id
# both actor_id and last_update are fields in table actor.

Example: Advanced SQL with joins and ORDER BY.

[dbmon-tail://mySQL/myTail]
output.format = kv
output.timestamp = 1
query = SELECT A.address_id, A.address, C.city FROM address A, city C
WHERE C.city_id=A.city_id {{ AND $rising_column$ > ? }} ORDER BY
A.address_id
sourcetype = mysource
tail.rising.column = address_id

[dbmon-dump://<database>/<unique_name>]

Example: Advanced SQL with joins and ORDER BY.

[dbmon-dump://mySQL/MyDump]
output.format = kv
output.timestamp = 1
output.timestamp.column = last_update
query = SELECT A.address_id, A.address, A.last_update AS last_update,
C.city FROM address A, city C WHERE C.city_id=A.city_id ORDER BY
A.city_id
sourcetype = mysourcetype

[dbmon-change://<database>/<unique_name>]

change.hash.algorithm = MD5|SHA256

[dbmon-batch://<database>/<unique_name>]
```