RunDbp (“**Run** **D**ata**B**ase **P**rocess”) is a command line program to run database process automation scripts. This document tells how to configure and use the RunDbp program on the MS Windows platform. Instructions would be similar on Linux.

# Syntax

Invoke RunDbp from the MS Windows command line as follows:

rundbp [ -c | --check ] *scriptName* [ *argument1* [ *argument2* … ] ]

*scriptName* is the name of a file with .dbp file extension. The .dbp extension must be omitted on the RunDdp command line.

The arguments are passed to script parameter values. See “DBPA Scripting Language.docx” for details of defining and using parameters in .dbp scripts.

The -c or –check option checks the script for valid syntax and checks that the specified arguments are compatible with the script parameter types, but does not run the script. It issues an error message and returns with non-zero return code if any problem is found. It runs silently and returns zero return code if the script is valid and the arguments are compatible.

Before you can use RunDbp, you must create several files that control its configuration.

# Prerequisites

RunDbp is implemented in Java as a runnable JAR file. You must have a Java 8 JVM installed on your machine.

The JAR contains all the classes needed to run the program, including classes to connect to MySQL and MS SQL Server databases. The class containing the main method is com.hauldata.dbpa.RunDbp.

You must install any MS Windows device drivers required to access your database.

If you are using MS SQL Server integrated security, you must download sqljdbc\_auth.dll from Microsoft.

The configuration of RunDbp is controlled by properties files. The contents of those files are described later in this document. You should select a directory where you will place the properties files. This is your DBPA\_HOME directory.

# Windows Batch File (shell script) Setup

You must create an MS Windows batch file named rundbp.bat that optionally sets the environment variable DBPA\_HOME to the directory where you placed the properties files and then invokes RunDbp.jar.

If you do not set the DBPA\_HOME environment variable, all properties files must be in the current directory when running RunDbp.

If the program java.exe is not already in your PATH, you must either add it to your PATH in rundbp.bat or specify the fully qualified file path of java.exe when invoking it.

When connecting to a MySQL database server or an MS SQL Server database server not using integrated security, suggested rundbp.bat file contents are as follows:

@echo off

set DBPA\_HOME=*pathToProperties*

REM If java.exe is not in your path, you could do this:

PATH=*pathToJavaExe*;%PATH%

java -jar *pathToJar*\RunDbp.jar %\*

When connecting to a SQL Server database server using integrated security, you must place the file sqljdbc\_auth.dll in the class path. Integrated security means that you login to SQL Server using your MS Windows credentials. When using SQL Server integrated security, or if connecting to a database that has its own .class file not bundled into RunDbp.jar, suggested rundbp.bat file contents are as follows:

@echo off

set DBPA\_HOME=*pathToProperties*

REM If java.exe is not in your path, you could do this:

PATH=*pathToJavaExe*;%PATH%

java -cp *pathToDllOrClass*;*pathToJar*\RunDbp.jar com.hauldata.dbpa.RunDbp %\*

# Properties Files

The properties file contents all follow Java conventions. In particular, the backslash is an escape character. Therefore, to use a literal backslash character in a file path in a properties file, you must specify double backslashes.

## RunDbp.path.properties

This file controls the directory paths where various files are found. Contents are as follows:

process=*pathToProcessScriptFiles*

log=*pathToLogFiles*

read=*pathToSourceFiles*

write=*pathToTargetFiles*

data=*pathToDataFiles*

properties=*pathToPropertiesFiles*

If any property is omitted or if this file is not found, the current directory is used as the path for the corresponding files.

The process property determines the directory where process script files are found. As noted, when process scripts are named on the command line, they must omit the .dbp extension. However, they can include a relative or absolute path prior to the file name. If a relative path is specified, it is relative to the process path. These same rules apply when naming process scripts invoked by a PROCESS task in a script.

The log property determines the directory where log files are written. The full naming of log files is controlled by the RunDbp.log.properties file. If the log file name is specified there with a relative path, it is relative to the log path.

The read property determines the directory where input files are located by default. Input files are files read *or written* by OPEN, LOAD, READ, UNZIP, or GET tasks. The assumption is that files written by UNZIP or GET will be read later in the same process, thus are written to the read directory.

The write property determines the directory where output files are located by default. Output files are files written *or read* by CREATE, APPEND, WRITE, ZIP, PUT, EMAIL, DELETE, RENAME, COPY, or MAKE tasks. The assumption is that files read by ZIP, PUT, EMAIL, DELETE, RENAME, or COPY will have been written earlier in the same process, thus will be located in the write directory.

The data property is supported for backward compatibility. It determines the directory where input and/or output files are located by default if the read and/or write property is omitted.

A data file name can always be specified on any task using an absolute path, in which case the data path has no effect on the location of that file.

The properties property determines the directory where files read by the PROPERTIES data source are located by default.

## RunDbp.log.properties

This file controls where script logging messages are written. Contents are as follows:

type=*listOfTargets*

level=*loggingLevel*

consoleLevel=*consoleLoggingLevel*

fileName=*logFileNamePattern*

fileRollover=*rolloverSchedule*

fileLevel=*fileLoggingLevel*

tableName=*databaseLogTableName*

tableLevel=*databaseLoggingLevel*

The type property allows a comma-separated list that can include console, file, and/or table in any order. Log information is directed to each of the specified target types.

If type is specified as null or if the type property is omitted or if this file is not found, log information is not captured.

The level property is the default logging level for any target that doesn’t explicitly specify a logging level. It can be info, warning, error, or message. The default of info logs the maximum amount of information. To minimize the amount of information logged, select message level, which only logs messages explicitly written to the log by LOG and GO tasks in the script.

The consoleLevel property, if specified, sets the logging level for the console, overriding the default level.

If file logging is selected, the fileName property must be specified. It determines the name of the log file. The name may include one or more instances of %d{pattern}, which specifies a pattern to be used for formatting the current date and time to be substituted into the file name. The pattern must conform to syntax used by Java DateTimeFormatter.ofPattern(*pattern*). For example, %d{yyyy-MM-dd HH-mm-ss} substitutes current year, month 01 to 12, day 01 to 31, hour on a 24-hour clock, minute 00 to 59, and second 00 to 59 into the file name.

If the file name including optional date and time substitution evaluates to the same name as an existing log file, new log information is appended to the contents of the existing log file.

The fileRollover property only has an effect if file logging is selected. It specifies the schedule on which a new log file is started while the process runs. Each time a new log file is started, the file name is regenerated substituting the current date and time formatted according to the pattern in the fileName property.

The rollover schedule is a specified as a string conforming to the syntax accepted by the ON / ON SCHEDULE task. See “DBPA Scripting Language.docx” for full scheduling syntax. For example, DAILY, WEEKLY, MONTHLY, and HOURLY are acceptable rollover schedule strings.

The fileLevel property, if specified, sets the logging level for file logging, overriding the default level.

The tableName property only has an effect if table logging is selected. It specifies the database table to which log messages are written. The name must be fully qualified to work correctly with an INSERT INTO *databaseLogTableName* SQL statement. The table may have arbitrary column names but it must have exactly five columns with the following types in order: VARCHAR, VARCHAR, DATETIME / TIMESTAMP or compatible type, TINYINT / INT or compatible type, and VARCHAR. See Log Contents below for considerations when choosing the maximum length of the VARCHAR columns.

The tableLevel property, if specified, sets the logging level for table logging, overriding the default level.

### Log Contents

Log information is written in five columns. When written to file, it conforms to the CSV file specification, which means it can be viewed in MS Excel for example.

The first column hold process ID, which is generally the *scriptName* passed on the rundbp command line. The process ID will be appended with any nested process IDs if the script invokes other scripts. It will also be appended by a qualifier if nested scripts are invoked asynchronously.

The second column will hold task ID, which is generally the *taskname* on a script TASK. The task ID will be appended with any nested task IDs if the task encloses nested tasks.

The third column is the date and time stamp when the log information was written

The fourth column is the level of the log information as an integer 0 to 3 for levels info through message.

The last column holds the actual log message. It is recommended you declare this column at least VARCHAR(255) if logging to database table.

## RunDbp.jdbc.properties

This file configures the JDBC connection to your database. Required Contents are as follows:

driver=*driverJavaClass*

url=*databaseConnectionURL*

See the documentation for your database to determine the values to use for these properties. All other properties in the file are passed through to the driver itself. Again, see the document for your database to determine what other properties are required by the driver. Typically, at least the following are required:

user=*userName*

password=*userPassword*

For MySQL, you must set the allowMultiQueries property true in order to execute SQL that includes multiple statements separated by semicolons. Suggested properties to connect to a MySQL database running on your local machine are as follows:

driver=com.mysql.jdbc.Driver

url=jdbc:mysql://localhost/*yourDatabaseName*

allowMultiQueries=true

user=*yourUser*

password=*yourPassword*

For MS SQL Server without integrated security, suggested properties are as follows:

driver=com.microsoft.sqlserver.jdbc.SQLServerDriver

url=jdbc:sqlserver://*ipAddressOrServerName*

user=*yourUser*

password=*yourPassword*

For MS SQL Server using integrated security you must set the integratedSecurity property true. Suggested properties to connect to an MS SQL Server database using integrated security are as follows:

driver=com.microsoft.sqlserver.jdbc.SQLServerDriver

url=jdbc:sqlserver://*ipAddressOrServerName*

integratedSecurity=true

## RunDbp.mail.properties

This file configures the connection to your email server. Email is managed using the Java Mail API implemented in the javax.mail package. The connection is established using a password authenticator.

If your process does not use email, this properties file is not read and can be omitted. Otherwise, required properties as follows:

user=*yourMailUser*

password=*yourMailPassword*

All other properties in the file are passed through to the javax.mail.Session.getInstance( Properties, Authenticator) function using a password authenticator. See that function and documentation for your email server to determine what other properties are required.

To use the mail server for your gmail account, suggested properties are as follows:

user=*yourGmailAddress*

password=*yourGmailPassword*

mail.smtp.starttls.enable=true

mail.smtp.auth=true

mail.smtp.host=smtp.gmail.com

mail.smtp.port=587

## RunDbp.ftp.properties

This file configures the connection to your FTP server. FTP is managed using the Apache Commons VFS2 API implemented in the org.apache.commons.vfs2 package.

If your process does not use FTP, this file is not read and can be omitted. Otherwise, required properties are as follows:

protocol=*protocolName*

hostname=*hostName*

user=*userName*

password=*userPassword*

The *protocolName* can be ftp, ftps, or sftp. Optional properties are as follows:

strictHostKeyChecking=*keyCheckingYesNo*

timeout=*timeoutMilliseconds*

mode=*passiveModeYesNo*

dataTimeout=*dataTimeoutMilliseconds*

socketTimeout=*socketTimeoutMilliseconds*

Default for *keyCheckingYesNo* is no. Default for *passiveModeYesNo* is yes. If any timeout value is omitted, there is no timeout.

To connect to your corporate .com FTP server via SFTP, suggested properties are as follows:

protocol=sftp

timeout=10000

hostname=ftp.*yourCompanyDomain*.com

user=*yourFtpUser*

password=*yourFtpPassword*