[Skip to content](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html" \l "container)

* [*Safari*](https://www.safaribooksonline.com/home/)
* [Recommended](https://www.safaribooksonline.com/r/)
* [Queue](https://www.safaribooksonline.com/s/)
* [Search](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html)
* [Expand Nav](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html)
  + [Recent](https://www.safaribooksonline.com/recent/)
  + [Topics](https://www.safaribooksonline.com/t/)
  + [Highlights](https://www.safaribooksonline.com/u/p253751/)
  + [Settings](https://www.safaribooksonline.com/u/)
  + [Feedback](mailto:feedback@safaribooksonline.com)
  + [Sign Out](https://www.safaribooksonline.com/accounts/logout/)
  + [Settings](https://www.safaribooksonline.com/u/)**10** days left in your trial. [Subscribe](https://www.safaribooksonline.com/subscribe/).
  + [Feedback](mailto:feedback@safaribooksonline.com)
  + [Sign Out](https://www.safaribooksonline.com/accounts/logout/)

[Software Test Engineering with IBM Rational Functional Tester: The Definitive Resource](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html)

* [Search in book...](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html)

Top of Form

Bottom of Form

* [Toggle Font Controls](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html)
* [Share this](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html)
  + [Twitter](https://twitter.com/share?url=http://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch04.html&text=Software%20Test%20Engineering%20with%20IBM%20Rational%20Functional%20Tester%3A%20The%20Definitive%20Resource&via=safari)
  + [Facebook](https://www.facebook.com/sharer/sharer.php?u=http://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch04.html)
  + [Google Plus](https://plus.google.com/share?url=http://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch04.html)
  + [Email](mailto:?subject=Safari:%20Chapter%204.%20XML%20and%20Rational%20Functional%20Tester&body=http://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch04.html%0D%0Afrom%20Software%20Test%20Engineering%20with%20IBM%20Rational%20Functional%20Tester%3A%20The%20Definitive%20Resource%0D%0A)

[Prev](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch04.html)

[Chapter 4. XML and Rational Functional Tester](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch04.html)

[Next](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch06.html)

[Chapter 6. Debugging Scripts](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch06.html)

**Chapter 5. Managing Script Data**

**Chip Davis, Daniel Chirillo, Fariz Saracevic**

*In this chapter, you learn how Rational Functional Tester can manage script data. You start with in depth coverage of Rational Functional Tester datapools, followed by techniques to access databases from Java and .NET, and you learn how to handle user-specific runtime variables.*

**Accessing Rational Functional Tester Datapools**

The easiest and most common way of managing test data in your scripts is to use *datapools*. Datapools are essentially just tables of data that are represented in Rational Functional Tester as simple test elements, along with the scripts and maps. You can have multiple datapools and these can be shared by multiple test scripts.

Although simple datapool usage is easy, you can do more sophisticated things with datapools to handle test data in automated testing. For basic data-driven testing involving reading test data from a datapool and sending to the application under test during playback, you can rely on the wizards to create the datapool commands in the test scripts. For more powerful manipulation of datapools, you can use several methods in the Rational Functional Tester Application Programming Interface (API). These methods are in the package com.rational.test.ft.datapool, which is fully documented in the product Help.

You might want to use more than one datapool with a particular script. You might want to control the rows of data that are read while the script executes. You might even want to add data to a datapool while a test is running, instead of just reading data. All of this requires adding code into your test scripts and the examples in this section show you how to do this.

**Datapool Basics**

Datapools are essentially just tables of data that have special functions that enable Rational Functional Tester to work with them more easily. When you use datapools, you do not have to concern yourself with file locations or SQL statements to retrieve test data the way you would using files or database tables. You also do not need to use another application or editor, such as a spreadsheet or database tool, to work with your test data. You work with your datapools in Rational Functional Tester just like you work with scripts, object maps, and other test assets.

A datapool is made up of columns and rows of data. The columns are called *variables* and they have a type to represent the data in that column, such as a string or a float. Note that the default type is always a string. The rows are called *records*, and each row contains a value, which could be null, for every variable. You read a datapool row by row, meaning that there is always a pointer, called the *cursor*, to a particular record at a given point in time. When Rational Functional Tester reads from a datapool, it reads a variable from the current record.

**How Do I Read from a Datapool?**

When you use the data-driven wizards to automatically generate datapool script code, you see only one method in the script to read from the datapool. This is the dpString method, and you will probably see this often in scripts with datapools. An example of this is:

expirationDateText().setText(dpString("ExpirationDateText"));

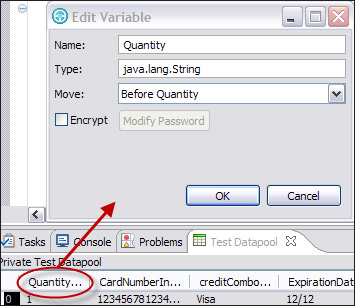
Other methods required to open and iterate through the datapool exist, but these are not shown in the test script. When you want to control and program these functions yourself, you add them into the test script.

Rational Functional Tester always reads a datapool value into a string by default, even for numbers. You can use several other methods to read different types of data, such as dpInt, dpFloat, and dpValue. You use these methods the same way you use dpString, for example:

image

For this code to work correctly, the datapool variable "Quantity" must be of type integer, as shown in [Figure 5.1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05fig01).

**Figure 5.1** Datapool variable—setting the variable type to integer



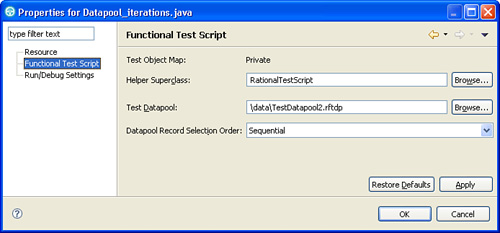
You normally use the name of the datapool variable, or column, with these methods to read a particular value. The variable name is contained in quotes as shown in the previous two examples. These methods always retrieve the current row of the datapool. Instead of using the variable name, you can use the index number to indicate which column value to read. The index starts at zero and increases from left to right. For example, if you wanted to read the fourth column of the current record, you would use:

creditCombo().setText(dpString(3));

**Multiple Scripts, Multiple Datapools**

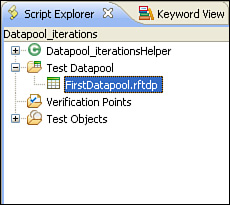
Typically, when you use the Rational Functional Tester data-driven wizards, you have a one-to-one relationship between a test script and a datapool. You are also likely to create a shared datapool at some point, which means multiple scripts use the same datapool. In both of these cases, a given test script is using only one particular datapool. You do not see any explicit lines of code in the scripts showing which datapool is accessed or how the records are incremented because the Rational Functional Tester handles this automatically. You can view the association between test scripts and their datapools by opening test scripts and looking in the Script Explorer. You can also view this association without opening test scripts by right-clicking on a test script from the Project Explorer and selecting Properties, and then clicking on Functional Test Script, as shown in [Figure 5.2](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05fig02).

**Figure 5.2** Test script properties—viewing the associated datapool



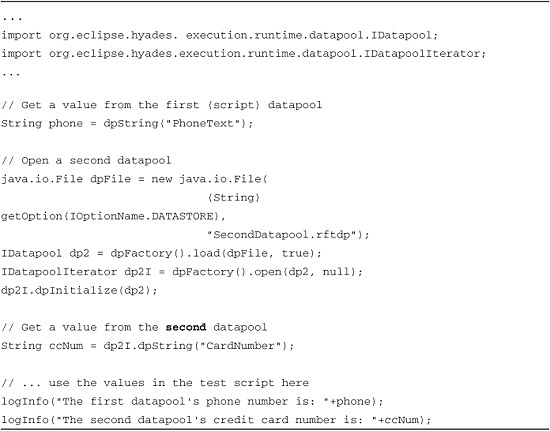
You might want to use more than one datapool with a given test script, in which case you will need to manually add code to the script. Whenever you have more than one datapool per test script, you need to explicitly state the name of the datapools to open them in the script. [Figure 5.3](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05fig03) shows a simple test script that reads data from two different datapools: FirstDatapool and SecondDatapool. Note that you see only the association to the first, or default, datapool in the Rational Functional Tester Script Explorer as shown in [Figure 5.3](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05fig03); the association to the second datapool is not shown graphically.

**Figure 5.3** Script Explorer showing the default associated datapool



To open a second datapool, you must add several lines of code to open the file, load the datapool, and open the iterator (controls the cursor or row pointer) for the datapool. After you create the new datapool and iterator in your script, you can use the same methods to work with the datapool as the default datapool by using the iterator or datapool object prefix. This is shown in [Listing 5.1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05ex01), which gets a phone number from the first datapool (shown in the script explorer) and then a credit card number from the second datapool.

**Listing 5.1** Example How to Add Second Datapool



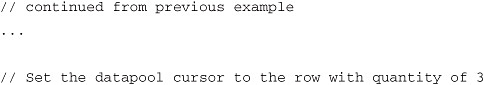
Test Assets in Rational Functional Tester

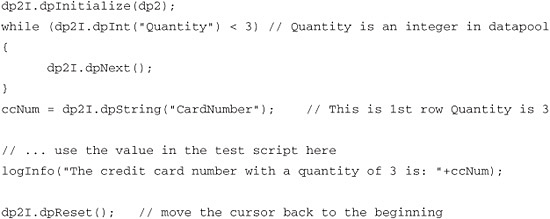
To clarify the test assets discussed here: test script contains the test procedure (the steps) and the verification (pass or fail) points, and the datapool contains data (numbers, names, values, and so on) that is used in the tests. Although some scripts might always use the same set of data (a one-to-one relationship), some might not (one-to-many, many-to-one, or many-to-many). For this reason, the scripts and datapools are kept as separate test assets.

**Setting the Datapool Row**

Typically, Rational Functional Tester automatically handles which record (row) of a datapool is being read. It also automatically increments the cursor that points to a row for each datapool iteration (see the section, “[Handling User-Specific Runtime Variables](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05sec1lev3),” later in this chapter) or when another script reads from the same datapool. However, there are times when you might need to manipulate this yourself through script code. You might want to count a certain number of times to the desired row, or you might want to look for a particular variable value.

The following example shows how to set the datapool cursor to a particular row. In this example, you loop through each row of the datapool until you find a specific value, and then you reset the cursor back to the beginning.



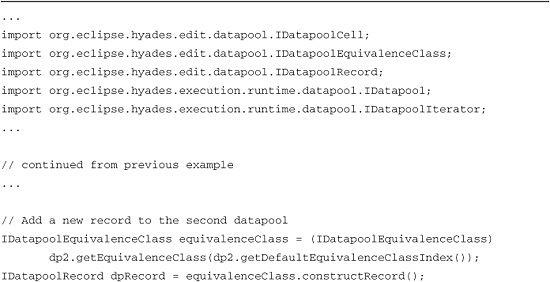


**Adding Rows to a Datapool During Test Playback**

So far, you examined several examples of how to read values from datapools. You might usually think of test data as being fixed at the time you are ready to execute tests. However, you might want to add value into a datapool while a test is running. You might do this to retrieve some value from the application under test and store it back into a datapool for another test to use. To do this, you create an equivalence class from the datapool and then use the constructRecord and appendRecord methods to create the new row. You then use dpFactory to save the record into the datapool.

In [Listing 5.2](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05ex02), you create a new record, copy the values from the first record into the new record, and then save the record into the datapool.

**Listing 5.2** Example of Adding Rows to a Datapool

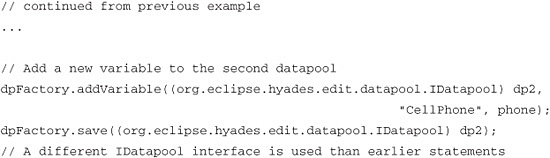




**Adding Columns to a Datapool During Test Playback**

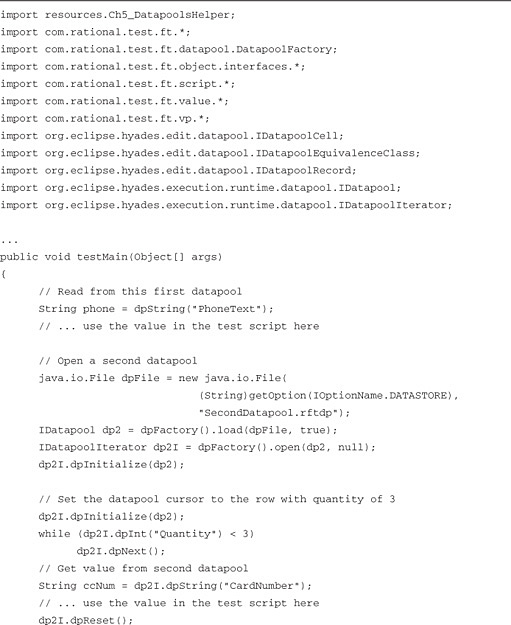
Like the values in a datapool, you normally define the columns in a datapool while you are developing the tests, and they are then fixed when you begin running the tests. However, you might want to add a new variable to a datapool during test execution. You might want to do this for similar reasons when adding values during test execution. To do this, use the dpFactory addVariable method to create a new variable for the datapool.

In the following example, you create a new variable called "CellPhone" and copy the value from the "Phone" variable into the new record. Note that this simple example applies the same phone value into the new variable for all records in the datapool, which might not be realistic. This is equivalent to using the Copy and Paste commands in a spreadsheet to fill an entire blank column with the same value.



[Listing 5.3](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05ex03) is the complete test script combining each of the previous examples.

**Listing 5.3** Complete Test Script Combining Previous Examples





**Database Access from Java and .NET**

This section looks at how to create Rational Functional Tester test scripts for both Java and the Visual Basic .NET environment to access information from databases. There are a number of possible ways to implement database access in Visual Basic; however, you examine the most common way to do this in the .NET environment using ActiveX Data Objects (ADO).

This section shows the basic functions used to work with data in databases for use in Rational Functional Tester test scripts. This section does not explain SQL queries or other database-specific information. You have to know the database connection strings and the SQL queries to get the information you need for your tests. After you have this information, you can use the examples shown here as a reference to make your own specific Rational Functional Tester scripts.

**JDBC Overview**

Java Database Connectivity (JDBC) is an Application Programming Interface (API) that defines how a client can access a database. It provides methods for accessing databases, sending SQL statements, and processing results. There are two major sets of interfaces provided by JDBC API. One is for application writers, and the second is the lower-level JDBC driver API for driver writers. There are four different JDBC driver categories:

• JDBC Type 1—JDBC-ODBC Bridge plus Open Database Connectivity (ODBC) Driver

• JDBC Type 2—A native API partly Java technology-enabled driver

• JDBC Type 3—Pure Java Driver for Database Middleware

• JDBC Type 4—Direct-to-Database Pure Java Driver

You use JDBC APIs to create automated test scripts, and you might need to add lower-level JDBC driver developed by Relational Database Management Systems (RDBMS) or third-party vendors. If you are interested to learn more about JDBC, please visit <http://java.sun.com/products/jdbc/overview.html>.

**ADO Overview**

ADO, not to be confused with Data Access Objects (DAO), which was a previous technology used with Visual Basic, is a flexible way to develop database access code that can then be used for different data sources. This enables you to create an application, or in this case, automated test scripts, with a simple database such as Microsoft Access® and later change to an enterprise level database such as IBM DB2® or Microsoft SQL Server with barely any changes to the implementation. In Rational Functional Tester Visual Basic scripting, you can add classes for ADO.NET from the System.Data.dll library.

**Setting Up a Rational Functional Tester Project to Access a Database**

The first step to access a database from your Rational Functional Tester project is creating the connection and opening the database. This is similar regardless of the kind of database you have, although the details vary.

As a prerequisite, you must ensure that every workstation that executes these test scripts has access to the database(s). This might require the installation of database drivers for Rational Functional Tester to gain access. You might also have to determine if special database users or login credentials are used by the automated test scripts. Basically, if you cannot manually connect to the database from a particular workstation, then Rational Functional Tester is not able to either.

The general steps to establish a connection to a database in the test scripts follow:

**1.** Create the connection using a known connect string.

**2.** Open the database.

**3.** Interact with the database, which can include:

• Executing queries using known SQL statements

• Inserting or deleting rows of data

• Firing database triggers by executing SQL statements

**4.** Close the database.

Several similar but distinct classes are available in the .NET framework to connect to different types of databases. The following list shows the commands you can use in Visual Basic depending on the type of database to which you are connecting:

• If you have IBM DB2 or other databases with Object Linking and Embedding (OLE) access, you can use OleDbConnection and OleDbCommand in the System.Data.OleDb namespace.

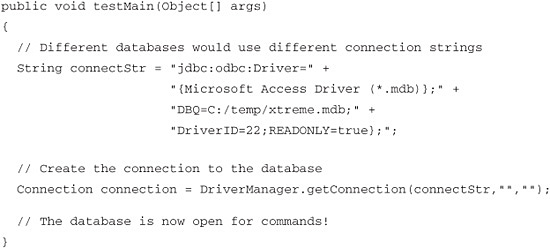
• If you have Microsoft SQL Server, you can use the SqlConnection and SqlCommand objects in the System.Data.SqlClient namespace.

• For Oracle, you use the OracleConnection and OracleCommand objects in the System.Data.OracleClient namespace.

You use a simple Microsoft Access database for the example in this section. You do not need Microsoft Access to execute this test script, you need only the database file (with the .mdb file extension) and the Access drivers, which are included in most Windows installations.

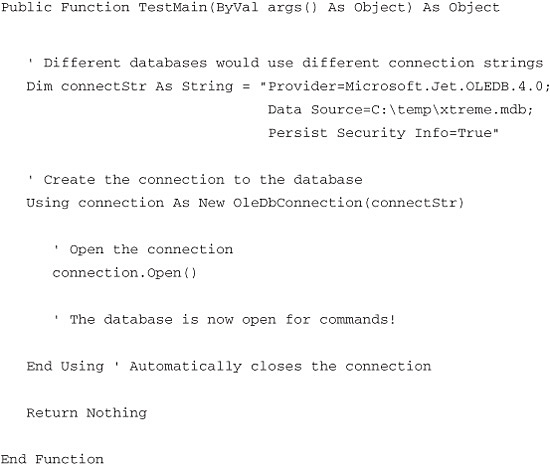
**Java**

In the next example, you use the DriverManager to load JDBC driver in memory and getConnection() method along with a specific connection string to connect to the database. If you use a different database type, your connection string is different. If you are interested in establishing a connection to DB2, see the developerWorks article “Establishing an IBM DB2 database connection in IBM Rational Functional Tester” at <http://www.ibm.com/developerworks/rational/library/07/1120_saracevic-narasappa/index.html> and for Oracle, see the article “Establishing a Database Connection in IBM Rational Functional Tester Scripts” at <http://www-128.ibm.com/developerworks/rational/library/05/saracevic/>.



**VB.NET**

In the next example, you use the OleDbConnection class along with a specific connection string, followed by the Open() method to connect to the database. If you use a Microsoft SQL Server database, then you use the SqlConnection class instead. Likewise, if you use an Oracle database, then you would use the OracleConnection class. If you use an IBM DB2 database, then the only difference is the content of the connection string.



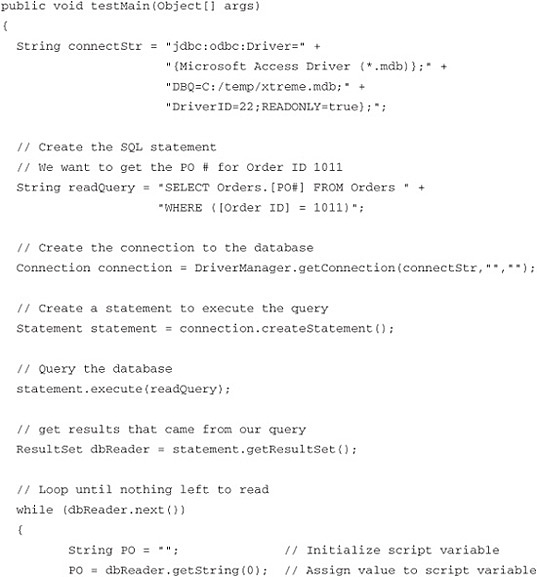
**Querying a Database**

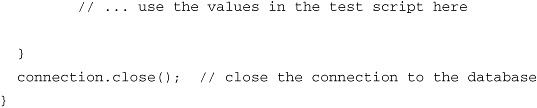
Often, one of the first things you do after connecting to a database is execute a query. This is not required, but it is common because querying is how you get information out of a database. The results from a query can be a set of values or it can be a single value. The data you get from each query can then be used in the Rational Functional Tester script as part of the automated test. You can use a database for test data instead of, or in combination with, datapools.

A query is made up of SQL statements, which are stored in a string in the Rational Functional Tester script. The value of the query string can be hard coded in the script, generated from other values, passed as an argument, or come from another source into the script. Putting together a SQL query programmatically can be accomplished using simple string functions. In the example, you just hard code the SQL query string that you use.

**Java**

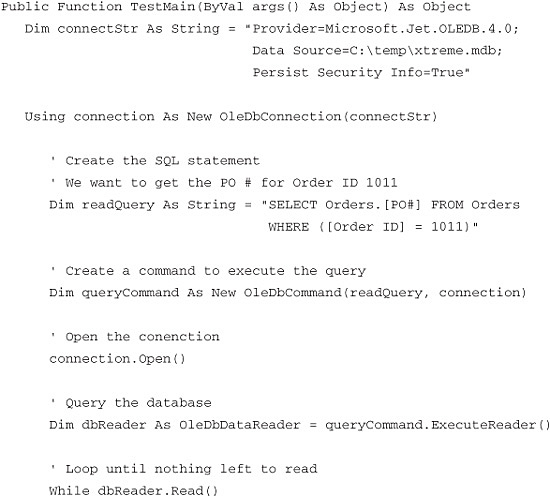
The next example shows you how to execute a SQL query against a database and read the result into the script. You use the createStatement method to execute a SQL query or statement, which is linked to the database connection. You then use the execute method to execute a query that returns rows of data to a ResultSet object. You use the next() method to extract values from the return set. Because you know that the one column value you are retrieving (PO#) is a string value, you can use the getString method to assign this to a local script variable. You can use many methods and techniques to get values from the reader object to use in your test script.

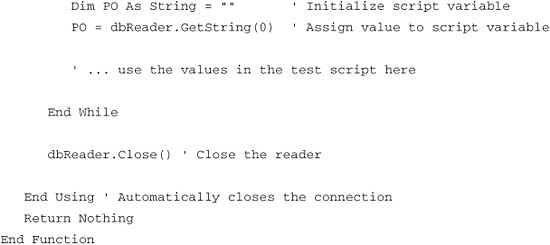




**VB.NET**

The next example shows you how to execute a SQL query against a database and read the result into the script. You use the OleDbCommand method to execute a SQL query or statement, which is linked to the database connection. You then use the ExecuteReader method to execute a query that returns rows of data to an OleDbDataReader object. You then use the Read method to extract values from the return set. Because you know that the one column value you are retrieving (PO#) is a string value, you can use the GetString method to assign this to a local script variable. You can use many methods and techniques to get values from the reader object to use in your test script.



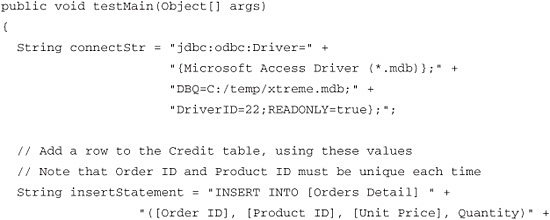


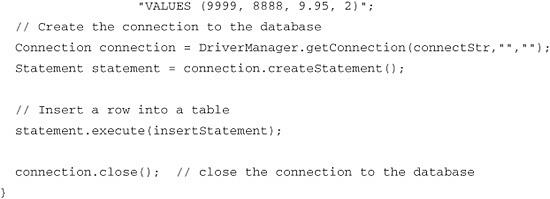
**Inserting Rows into a Database Table**

One common reason for linking a Rational Functional Tester script to a database is to store data produced or obtained from the test procedure. The information might be stored for logging purposes or the data might be used in other automated tests. You put information into a database by inserting new rows of data into a table. To do this, you must already know which table to use and its design (the data types of each table column). To modify the particular table, you must have the required database permissions obtained from the connection string user name and password. After you know the table you want to modify, adding a row is simple in Rational Functional Tester.

**Java**

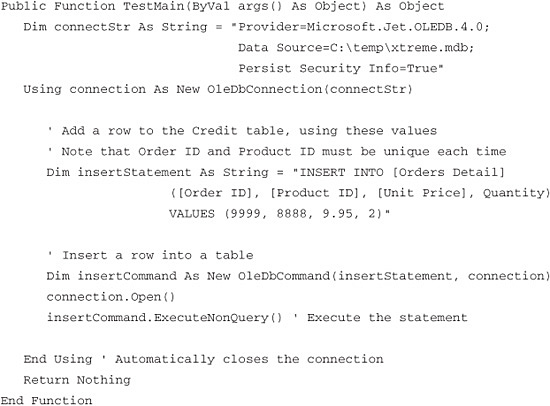
The following example shows you how to insert a row into a table. You do this by using SQL statements just like the query you created to read data from the table. The difference is that the SQL statement you execute does not return any data from the database.





**VB.NET**

The following example shows you how to insert a row into a table. You do this by using SQL statements just like the query you created to read data from the table. The difference is that the SQL statement you execute does not return any data from the database. Because of this, you use the ExecuteNonQuery method instead of ExecuteReader. You do not have to assign return data into the script, but you can use the return value of this method, which is the number of rows affected by the statement execution, for error checking.



Inserting Rows into Tables with Unique Keys

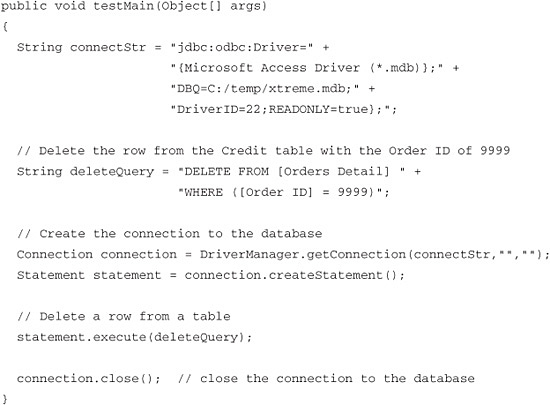
Many database tables have keys, or specific column values, which must be unique for each row. You have to manage the values in an SQL INSERT statement to ensure that you are not attempting to add duplicate keys. The example shown here might not be realistic because the values are hard-coded in the script.

**Deleting Rows from a Database Table**

In addition to inserting rows to a database table, you can also make a Rational Functional Tester script delete rows. You likely have to do this to manipulate the test data used by your automated tests. Just like inserting rows, you must already know which table and have the required database permissions. After you have this information, deleting a row is easy to do in Rational Functional Tester.

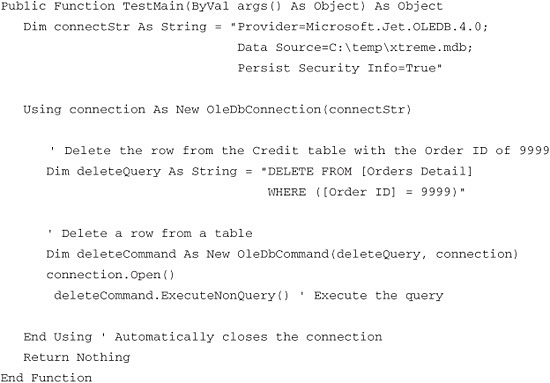
**Java**

The following example shows you how you can delete a row from a table. You do this by using SQL statements just like in the previous examples. The only difference between inserting and deleting rows is the SQL statement used. You also have to consider which data and rows you want to mark for deletion from the database. In the example shown, you delete the row that was inserted from the previous example, using a hard-coded filter ([Order ID] = 9999).



**VB.NET**

The following example shows you how you can delete a row from a table. You do this by using SQL statements just like in the previous examples. Like inserting rows, the SQL statement you execute does not return any data from the database. The only difference between inserting and deleting rows is the SQL statement used. You also have to consider which data and rows you want to mark for deletion from the database. In the example shown, you delete the row that was inserted from the previous example, using a hard-coded filter ([Order ID] = 9999).



**Fire a Database Trigger**

Database triggers automatically make something happen in the database when a particular event occurs. These are special types of stored procedures, which are like functions that run inside the database. There are different kinds of events that can fire (cause it to execute); the trigger and the trigger can do various things in the database. For example, a database might have a trigger that fires when a row is inserted into a particular table, which then causes the trigger to update other tables.

Although possible, you are not likely to create or modify database triggers from a Rational Functional Tester script. You might, however, want to fire a trigger. Causing a trigger to fire is just a matter of making whatever change is necessary in the database through SQL statements, such as an INSERT, an UPDATE, or a DELETE command. Therefore, you can fire a trigger from a test script by invoking the appropriate SQL statement with the right data and possibly at the right time or sequence. This obviously requires knowledge of the triggers you might want to fire and the database schema design, which you can get from a database administrator (DBA). One possible scenario in which you would fire a trigger in an automated test procedure follows:

**1.** Get some values from the application under test as part of the test procedure.

**2.** Put these values into the database that is part of, or linked to, the application under test. Do this by executing a SQL statement, such as an INSERT, which fires a database trigger to update data in other tables (which is part of the application’s functionality).

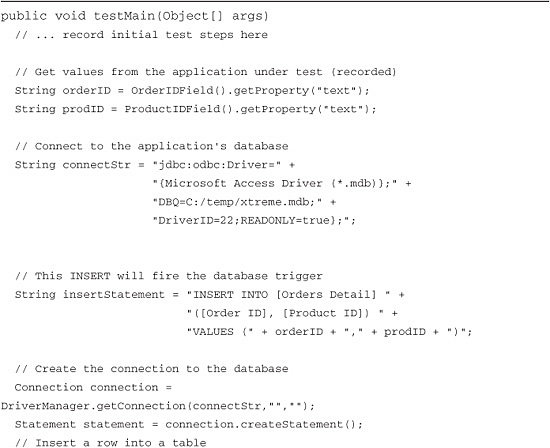
**3.** Read the new information from the database by executing a query to read the tables updated by the trigger.

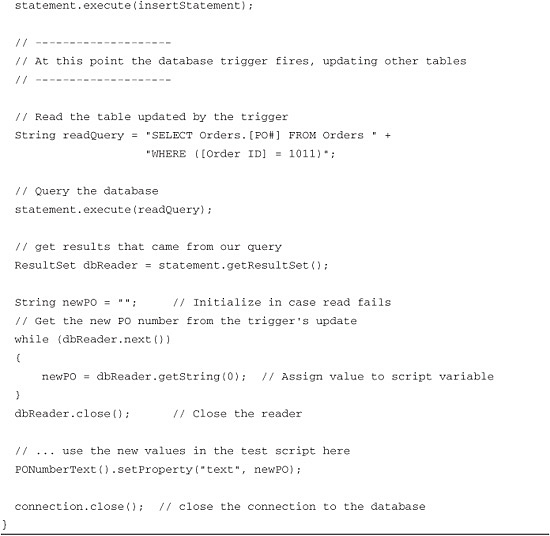
**4.** Use the new values from the query as test data in the test procedure.

**Java**

In [Listing 5.4](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05ex04), Java code shows one possible example how to fire a trigger as described in the previous section.

**Listing 5.4** Java Example of How to Fire a Database Trigger

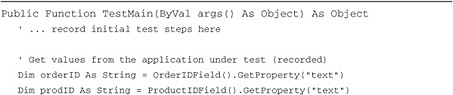


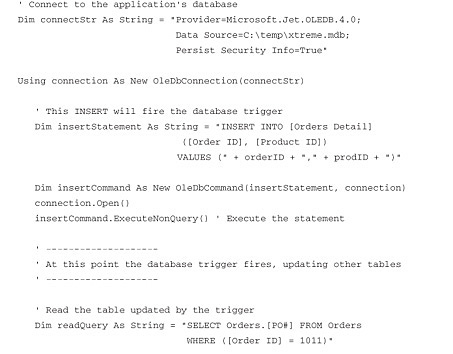


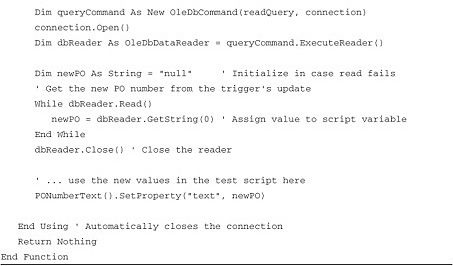
**VB.NET**

In [Listing 5.5](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05ex05), VB.NET code shows you one possible example how one would fire a trigger as described earlier in this section.

**Listing 5.5** VB.NET Example of How to Fire a Database Trigger







**Handling User-Specific Runtime Variables**

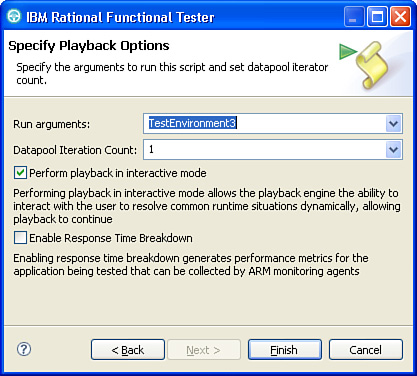
Often, when you run sets of automated tests, you need to run tests with varying options. These options might set different test environments or configurations, or they might set the number of times various parts of the tests are run. There are many other possible uses of these values. The difference between the test values discussed in this section from test data stored in datapools is that these parameters are typically assigned or picked at the time of execution. However, there might not always be a clear “border” or rule where you might put a value in a datapool or pass a value as a user-specific runtime variable.

The following four sections describe and give examples of four different ways to set these user-specific runtime variables. Also related to this is passing variables between scripts as arguments, which is covered in [Chapter 7](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch07.html#ch07), “[Managing Script Execution](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch07.html#ch07).”

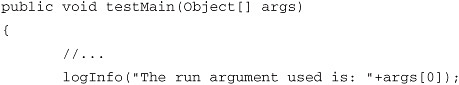
**Playback Options**

The simplest way to pass runtime parameters to a test script is using the Rational Functional Tester playback options. There are two different playback options: the run arguments and the datapool iteration count. You can assign values to these during the second step of the playback wizard, shown in [Figure 5.4](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05fig04).

**Figure 5.4** Playback options—passing arguments to a test script



The run argument is passed to the test script’s testMain() as the args value. This argument is always declared as an Object type, but you can cast it as a String or other type as needed. You can then use the args variable, containing the runtime argument, in the script. The following simple example illustrates a run argument used in a script.

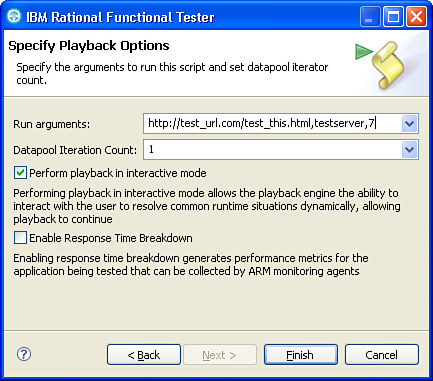


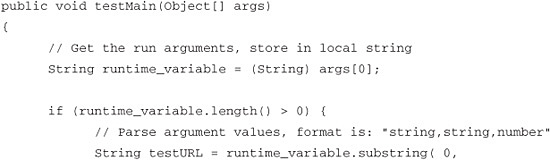
Passing a String with Spaces

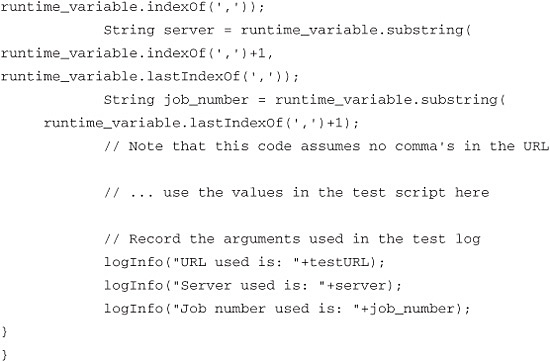
If the run argument you want to pass to the test script is a string with spaces, you must enclose the argument with double quotes.

If you need to pass multiple values, you can enter them into a string separated by spaces or other separator character, and then parse the string in the script as shown in [Figure 5.5](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05fig05). The following code shows how to pass a string and a numeric value to a script.

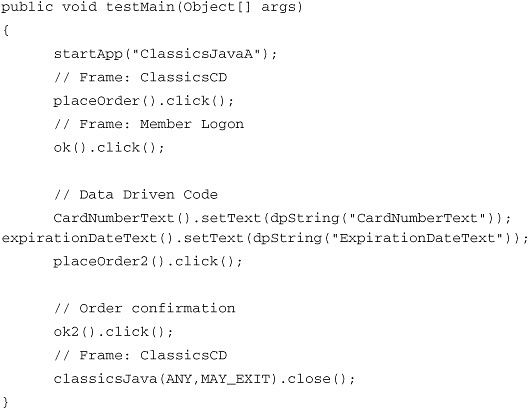
**Figure 5.5** Example playback options—passing three arguments







The second Rational Functional Tester playback option is the datapool iteration count. This option is available only if a test script has datapool associated with it; otherwise, it is disabled. This value controls how many rows of the datapool are retrieved when the test is run. The test script repeats the lines of code based on datapool iteration count. For example, the following code illustrates a test script getting values from a datapool (from a single row) and entering them into the application under test.



If you run this test script and enter a datapool iteration count of 5, the script runs from beginning to end five times to retrieve five datapool rows. If the datapool has less than five rows, it repeats some rows. Another playback option for the datapool iteration count is to select Iterate until done, which repeats as many times as needed to get to the end of the datapool, without repeating any rows.

Using the playback options is the easiest way to pass user runtime variables to a test script because it requires minimal lines of code and no files to manage. The disadvantage is that you have to manually enter the values every time you execute the script.

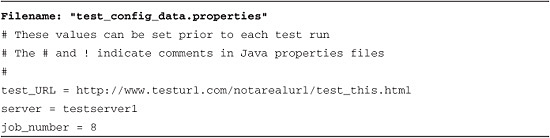
**Java Properties Files**

A Java properties file is an easy way for Java programs to get data, typically application configuration parameters, that is in stored files. The advantage of using properties files in Java is that classes and methods make it much easier to read and parse the files without having to write detailed code for file input and output. These files are simple text files, and they always have the file extension .properties. The basic format is a list of data, each having a name and the value, referred to as key/value pairs. The classes to use Java properties files are in the package java.util.Properties.

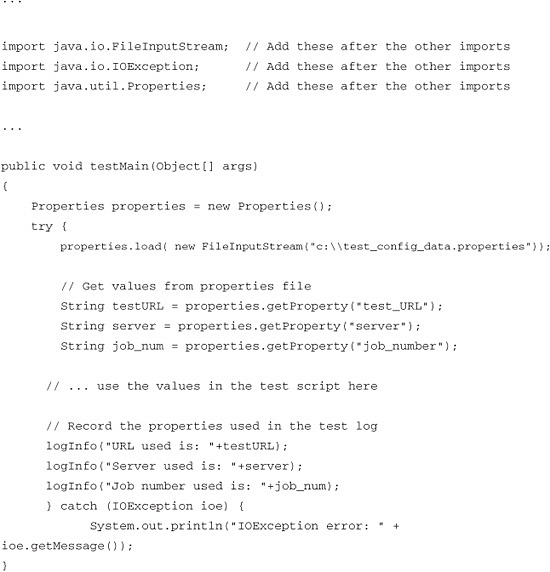
If you use Rational Functional Tester with Java scripting, you can use Java properties files to store configuration data for your test scripts. As long as you follow the correct format for these files, you can get the values into your test script using only a few lines of code. You can learn more about the file format and classes that implement this at [http://java.sun.com](http://java.sun.com/).

The following is an example of a Rational Functional Tester script using a Java properties file. In this example, you start with a properties file containing three values, which are used as test parameters. The properties file contents are shown in [Listing 5.6](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05ex06).

**Listing 5.6** Example Java properties file contents



Next, you add lines of code to retrieve these values from the file into the test script. In this example, the file is named test\_config\_data.properties and it is saved at the root of the C: drive. There are only three lines of code required to do this: declaring a new Properties object, opening the file with the load() method, and getting the value using the key name with the getProperty() method.



The previous example simply assigns these values to local script variables and writes them to the test log. Realistically, you use the variables as part of a test procedure in the middle of the previous example.

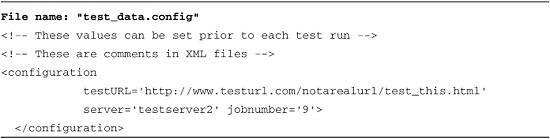
**XML Files**

Similar to the way Java programmers use properties file, .NET programmers can use XML files to store configurable parameters for applications. Like properties files, the advantage of using XML configuration files in .NET is that there are classes and methods that make it much easier to read and parse the files without having to write detailed code for file input and output. XML files are a common file format used to transfer data between many system components and applications. XML files can have any extension, but the .config extension is frequently used for storing application configuration parameters. You can use XmlTextReader from the System.Xml package in the .NET Framework class library to work with XML files.

If you use Rational Functional Tester with Visual Basic scripting, you can use XML files to store configuration data for your test scripts. As long as you follow XML syntax for these files, you can get the values into your test script using only a few lines of code. You can learn more about the file format and classes that implement this at <http://www.w3.org/>.

The following is an example of a Rational Functional Tester script using an XML configuration file. Like the previous example, you start with a configuration file containing three values that will be used as test parameters. The file contents are shown in [Listing 5.7](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05ex07).

**Listing 5.7** Example XML configuration file contents



You now add lines of code to retrieve these values from the XML file into the test script. In this example, the file is named "test\_data.config" and it is saved at the root of the C: drive. There are only three lines of code required to do this: declaring a new reader (file) object, accessing the file contents with the MoveToContent() method, and getting the value using the key name with the GetAttribute() method.

You can use the System.Xml library; however, Rational Functional Tester does not support importing the .NET System package into a Visual Basic test script. Instead, you must copy the required dynamic link library (DLL) file into the Rational Functional Tester customization directory. Copying the DLL files automatically adds the methods references to the Rational Functional Tester projects. This customization directory is located by default in the following locations:

• Windows Vista at C:\Users\All Users\IBM\RFT\customization

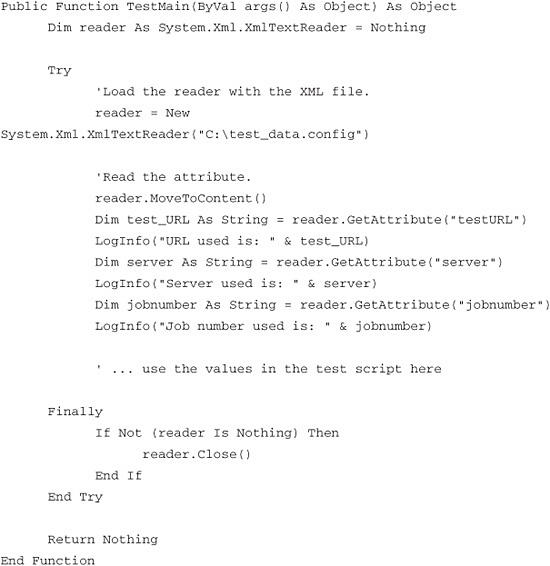
• All other Windows at C:\Documents and Settings\All Users\Application Data\IBM\RFT\customization

The DLL files you need are located by default in the framework installation directory, for example: C:\WINDOWS\Microsoft.NET\Framework\v3.0.

You need to copy the files that contain the classes you need and any dependent assemblies (more files). For example, the XmlReader function shown in the following code is contained in the system.xml.dll, so this file gets copied to the Rational Functional Tester customization folder. Refer to the Microsoft .NET framework documentation and references for more information on the .NET System package contents.

Requirement for This Customization

This customization is per machine, so you have to repeat this for every machine that executes the test scripts requiring the System packages.



The previous example simply assigns these values to local script variables and writes them to the test log. Realistically, you use the variables as part of a test procedure in the middle of the script.

**Simple Flat Files**

In this last example, you examine passing runtime parameters to a test script that involves using simple flat files. This method refers to storing data in any file format and adding code to the test script to read from the file. This method enables the greatest flexibility, because you do not have to use a particular format or file extension, but it can also be the most complex and require the most amount of work to implement. You are most likely to use this approach in situations where you have to use existing files containing test parameters and cannot change the file format.

There are many different possibilities for file formats and even more possibilities for how to read data from these files. You look at only one possible example to illustrate the strategy. The general strategy to get data into a test script from a file is as follows:

**1.** Open the file.

**2.** Read a line from the file.

**3.** Parse the line for the specific data you need.

**4.** Repeat getting lines to get the particular data you need.

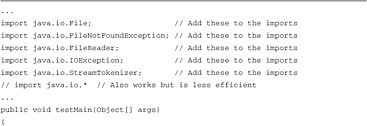
**5.** Close the file.

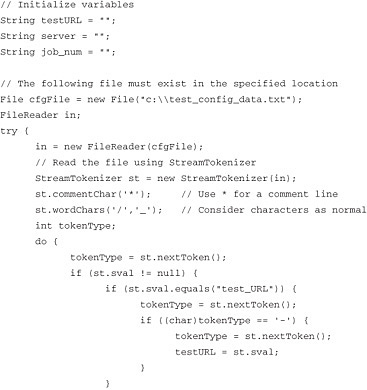
Java and .NET, Properties, and XML

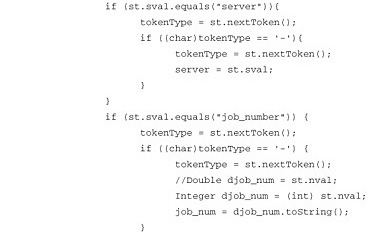
Although it is not typical, it is possible to use a .properties file with a Visual Basic test script or you can use an XML file with a Java test script. In these cases, you do not use the built-in classes, but instead, you use a strategy similar to reading simple flat files, which is described in the following section.

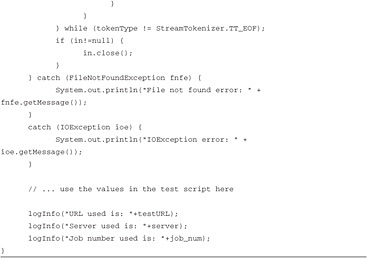
In [Listing 5.8](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05ex08), the simple example reads a file that has three parameters of data on each line of the file. The methods used in this example are typical functions in Java for reading and parsing files. The comments in the test script code explain many of the specific steps.

**Listing 5.8** Example of How to Read a File

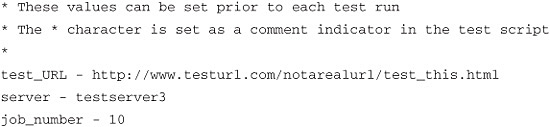








The file used for this example can have data as follows:



Like the previous examples, this example simply assigns these values to local test script variables and writes them to the test log. Realistically, you should use the variables as part of a test procedure in the middle of the test script.

**Summary**

As you can see, Rational Functional Tester gives you various options to manage data. Using datapools enhances your test scripts with an elegant and a simple mechanism. The openness of Rational Functional Tester enables you to access databases and how it handles user-specific runtime variables. This chapter pointed to the most common implementation for managing data. Either you can embrace one of these implementations or you can create a hybrid that would fit the needs of your organization.

[Prev](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch04.html)

[Chapter 4. XML and Rational Functional Tester](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch04.html)

[Next](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch06.html)

[Chapter 6. Debugging Scripts](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch06.html)

**Welcome to Safari**. Remember, your free trial will end on March 7, 2015, but you can [subscribe at any time](https://www.safaribooksonline.com/subscribe/)

* [Recommended](https://www.safaribooksonline.com/r/)
* [Queue](https://www.safaribooksonline.com/s/)
* [Recent](https://www.safaribooksonline.com/recent/)
* [Topics](https://www.safaribooksonline.com/t/)
* [Settings](https://www.safaribooksonline.com/u/)
* [Blog](http://blog.safaribooksonline.com/)
* [Support](http://support.safaribooksonline.com/?prod=flow)
* [Support](http://msupport.safaribooksonline.com/?prod=flow)
* [Feedback](mailto:feedback@safaribooksonline.com)
* [Sign Out](https://www.safaribooksonline.com/accounts/logout/)

© 2015 [Safari](http://www.safaribooksonline.com/).   
[Terms of Service](https://www.safaribooksonline.com/terms/) / [Membership Agreement](https://www.safaribooksonline.com/membership-agreement/) / [Privacy Policy](https://www.safaribooksonline.com/privacy/)

[**Make font larger Make font smaller**](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html)