**Working with the JRDataSource Interface**

[‹ Using the Empty Record Data Adapter](https://community.jaspersoft.com/documentation/tibco-jaspersoft-studio-user-guide/v611/using-empty-record-data-adapter) [up](https://community.jaspersoft.com/documentation/tibco-jaspersoft-studio-user-guide/v611/data-adapters) [A Look at TIBCO Spotfire Information Links ›](https://community.jaspersoft.com/documentation/tibco-jaspersoft-studio-user-guide/v611/look-tibco-spotfire-information-links)

<https://community.jaspersoft.com/documentation/tibco-jaspersoft-studio-user-guide/v611/working-jrdatasource-interface>

**Working with the JRDataSource Interface**

All data adapters implement the JRDataSource interface. Some data adapters, such the JDBC data connections, do this indirectly using a connection and a query; other data adapters, such as adapters for CSV files, XML documents, and collections of JavaBeans, do this directly. This section is useful if you want to understand more about the direct data adapters, or if you are interested in creating a custom data adapter.

**Understanding the JRDataSource Interface**

Data supplied by a JRDataSource is ideally organized into records as in a table. Every JRDataSource must implement the following two methods:

public boolean next() – Returns true if the cursor is positioned correctly in the subsequent record, false if no more records are available.

public Object getFieldValue(JRField jrField) – Moves a virtual cursor to the next record

Every time JasperReports executes the public boolean next() method, all the fields declared in the report are filled and all the expressions (starting from those associated with the variables) are calculated again. Subsequently, JasperReports determines whether to print the header of a new group, to go to a new page, and so on. When next returns false, the report is ended by printing all final bands (Group Footer, Column Footer, Last Page Footer, and Summary). The method can be called as many times as there are records present (or represented) from the data source instance.

The method public Object getFieldValue(JRField jrField) is called by JasperReports after a call to next results in a true value. In particular, it's executed for every single field declared in the report. In the call, a JRField object is passed as a parameter. It's used to specify the name, the description and the type of the field from which to obtain the value (all this information, depending on the specific data source implementation, can be combined to extract the field value).

The type of the value returned by the public Object getFieldValue(JRField jrField) method has to be adequate for that declared in the JRField parameter, except when a null is returned. If the type of the field was declared as java.lang.Object, the method can return an arbitrary type. In this case, if required, a cast can be used in the expressions. A cast is a way to dynamically indicate the type on an object, the syntax of a cast is:

(type)object

in example:

(com.jaspersoft.ireport.examples.beans.PersonBean)$F{my\_person}

Usually a cast is required when you need to call a method on the object that belongs to a particular class.

**Implementing a New JRDataSource**

If the JRDataSource supplied with JasperReports doesn't meet your requirements, you can write a new JRDataSource. This is not a complex operation. In fact, all you have to do is create a class that implements the JRDataSource interface that exposes two simple methods: next and getFieldValue.

|  |
| --- |
| The JRDataSource interface |
| **package** net.sf.jasperreports.engine;  **public** **interface** JRDataSource  {  **public** **boolean** next() **throws** JRException;  **public** Object getFieldValue(JRField jrField) **throws** JRException;  } |

The next method is used to set the current record into the data source. It has to return true if a new record to elaborate exists; otherwise it returns false.

If the next method has been called positively, the getFieldValue method has to return the value of the requested field or null. Specifically, the requested field name is contained in the JRField object passed as a parameter. Also, JRField is an interface through which you can get information associated with a field—the name, description, and Java type that represents it.

Now try writing your personalized data source. You have to write a data source that explores the directory of a file system and returns the found objects (files or directories). The fields you create to manage your data source are the same as the file name, which should be named FILENAME; a flag that indicates whether the object is a file or a directory, which should be named IS\_DIRECTORY; and the file size, if available, which should be named SIZE.

You data source should have two constructors: the first receives the directory to scan as a parameter; the second has no parameters and uses the current directory to scan.

Once instantiated, the data source looks for the files and the directories present in the way you indicate and fills the array files.

The next method increases the index variable you use to keep track of the position reached in the array files, and returns true until you reach the end of the array.

|  |
| --- |
| Sample personalized data source |
| **import** net.sf.jasperreports.engine.\*;  **import** java.io.\*;  **public** **class** JRFileSystemDataSource **implements** JRDataSource  {  File[] files = **null**;  **int** index = -1;  **public** JRFileSystemDataSource(String path)  {  File dir = **new** File(path);  **if** (dir.exists() && dir.isDirectory())  {  files = dir.listFiles();  }  }  **public** JRFileSystemDataSource()  {  **this**("."); |
| }  **public** **boolean** next() **throws** JRException  {  index++;  **if** (files != **null** && index < files.length)  {  **return** **true**;  }  **return** **false**;  }  **public** Object getFieldValue(JRField jrField) **throws** JRException  {  File f = files[index];  **if** (f == **null**) **return** **null**;  **if** (jrField.getName().equals("FILENAME"))  {  **return** f.getName();  }  **else** **if** (jrField.getName().equals("IS\_DIRECTORY"))  {  **return** **new** Boolean(f.isDirectory());  } |
| **else** **if** (jrField.getName().equals("SIZE"))  {  **return** **new** Long(f.length());  }  *// Field not found...*  **return** **null**;  }  } |

The getFieldValue method returns the requested file information. Your implementation doesn't use the information regarding the return type expected by the caller of the method. It assumes the name has to be returned as a string. The flag IS\_DIRECTORY as a Boolean object, and the file size as a Long object.

The next section shows how to use your personalized data source in Jaspersoft Studio and test it.

**Using a Custom JasperReports Data Source with Jaspersoft Studio**

Jaspersoft Studio provides a special connection for your personalized data sources. It's useful for employing whatever JRDataSource you want to use through some kind of factory class that provides an instance of that JRDataSource implementation. The factory is just a simple Java class useful for testing your data source and filling a report in Jaspersoft Studio. The idea is the same as what you have seen for the collection of JavaBeans data adapter — you need to write a Java class that creates the data source through a static method and returns it. For example, if you want to test the JRFileSystemDataSource in the previous section, you need to create a simple class like that shown in this code sample:

|  |
| --- |
| Class for testing a custom data source |
| **import** net.sf.jasperreports.engine.\*;  **public** **class** FileSystemDataSourceFactory {  **public** **static** JRDataSource createDatasource()  {  **return** **new** JRFileSystemDataSource("/");  }  } |

This class, and in particular the static method that's called, executes all the necessary code for instancing the data source correctly. In this case, you create a new JRFileSystemDataSource object by specifying a way to scan the directory root ("/").

Now that you have defined the way to obtain the JRDataSource you prepared and the data source is ready to be used, you can create the connection through which it can be used.

Create a new connection as you normally would (see [Working with Database JDBC Connections](https://community.jaspersoft.com/documentation/tibco-jaspersoft-studio-user-guide/v611/working-database-jdbc-connections#jss-user_data_sources_3464778876_1026774)), then select Custom implementation of JRDataSource from the list and specify a data source name like TestFileSystemDataSource (or whatever name you want), as shown below.

|  |
| --- |
|  |
| Configuring a Custom Data Adapter |

Next, specify the class and method to obtain an instance of your JRFileSystemDataSource, that is, TestFileSystemDataSource and test.

|  |  |
| --- | --- |
|  | There is no automatic method to find the fields managed by a custom data source. |

In this case, you know that the JRFileSystemDataSource provides three fields: FILENAME (String), IS\_DIRECTORY (Boolean), and SIZE (Long). After you have created these fields, insert them in the report’s Detail band.

Divide the report into two columns and in the Column Header band, insert Filename and Size tags. Then add two images, one representing a document and the other an open folder. In the Print when expression setting of the Image element placed in the foreground, insert the expression $F{IS\_DIRECTORY}, or use as your image expression a condition like the following:

($F{IS\_DIRECTORY}) ? “folder.png” : “file.png”

In this example, the class that instantiated the JRFileSystemDataSource was very simple. But you can use more complex classes, such as one that obtains the data source by calling an Enterprise JavaBean or by calling a web service.

Version:

# How to create and use a JRDataSourceProvider adapter

<https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasourceprovider-adapter>

Table of Contents [[hide](https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasourceprovider-adapter)]

* + [Overview](https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasourceprovider-adapter#Overview)
  + [The interface JRDataSourceProvider](https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasourceprovider-adapter#The_interface_JRDataSourceProvider)
  + [Location of the class](https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasourceprovider-adapter#Location_of_the_class)
  + [Creation of the Data Adapter](https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasourceprovider-adapter#Creation_of_the_Data_Adapter)
  + [Creation of the Report](https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasourceprovider-adapter#Creation_of_the_Report)

## Overview

In the tutorial [How to create and use a JRDataSource adapter](https://community.jaspersoft.com/node/803226) we have seen how to create a simple data adapter that gets the data from a custom class. One of the problem of that adapter was the managing of the fields. Suppose to use a custom JRDataSource providden by someone, how do you know which fields are required? Obviously you can read documentation on the data adapter or ask the author, but we will see another way to provide a data adapter, and integrated with it all these informations like list of fields, with name, description and type. We can do this by definig a custom datasource provider, a java class that implements the interface JRDataSourceProvider.

## The interface JRDataSourceProvider

To create a custom datasource provider we need to implement the interface JRDataSourceProvider, lets see which method this interface require:

* boolean supportsGetFieldsOperation() : returns true if the provider supports the field retriving operation, using the method getField. By returning true in this method the data source provider indicate that it is able to introspect the data source and discover the available fields.
* public JRField[] getFields(JasperReport report) : Returns the fields that are available from the data source. The provider can use the passed in report to extract some additional configuration information such as report properties.
* public JRDataSource create(JasperReport arg0) : Creates and returns a new instance of the provided data source. The provider can use the passed in report to extract some additional configuration information such as report properties. In other words this return the class that implements a JRDataSource interface (like the class MyImplementation seen in [How to create and use a JRDataSource adapter](https://community.jaspersoft.com/node/803226)) that contains the real data.
* public void dispose(JRDataSource dataSource) : disposes the data source previously obtained using the create method. This method must close any resources associated with the data source.

Now we have a general idea on how this inderface does, but lets see a pratical example. In this example as data adapter for the create method we have used MyImplementation, seen in [How to create and use a JRDataSource adapter](https://community.jaspersoft.com/node/803226), inside the implementation of the method create:

You can find this example attached.

## Location of the class

This class to work must be placed in the same project folder where you have your custom data adapter (MyImplementation in the example). We can put the class in a new package, so right click in the project folder MyReports and select New->Package.

At this point right click on the CustomDataAdapter package and select New -> Class. On the dialog that will appear insert MyDataSource as class name. Then press the button Add to add a new interface and select JRDataSourceProvider, if you have done right the JRDataSourceProvider interface will appear in the list of the used interfaces. Finally hit the Finish button to create the new class. At this point you have to write the data adapter code, but for this example you can just copy and paste the code written before.

## Creation of the Data Adapter

Now you have to create the data adapter that use this class, from select the element File -> New -> Data adapter.

From the dialog select the same project folder where you put the class (in this case MyReports) and has name of the file put MyCustomSource and hit Next. Now you have to choose the type of the data adapter, select JasperReports DataSource Provider Class and hit Next. At this point you must provide the information to get the class previously written:

* As name of the datasource you can put anything you like, in this case we can use MyCustomSource.
* On the field JasperReports DataSource Provider Class Name hit the button with three dots "...", and in the new dialog search for MyCustomSource and hit Ok. At the end in this field you should find something like CustomDataSource.MyDataSource.

After you have compiled all the fields hit the Test button to check if it is all right, and if you obtain a successful response hit the Finish button.

## Creation of the Report

At this point you can use this new data adapter inside a report. Create a new report from the report wizard (File -> New -> Jasper Report), for this example use the template Coffee. Place it in the same Project folder of the data adapter (MyReports) and as data adapter select MyCustomSource. At this point the new datasource should be recognized as a fields provider and you will see the following image:

and hit Next. Now to the provider will be requested the available fields, and they will be shown in the left list (in this example Name and Age), add them all to the report and then hit Finish.

Now your report will be generated with the two fields already in it. Switch to the preview tab to see the result:

Attachments:

[Example of a custom JRDataSourceProvider](https://community.jaspersoft.com/sites/default/files/wiki_attachments/mydatasource.java)

**How to create and use a JRDataSource adapter**

<https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasource-adapter>

Table of Contents [[hide](https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasource-adapter)]

* + [Overview](https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasource-adapter#Overview)
  + [The Class of the Custom Data Adapter](https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasource-adapter#The_Class_of_the_Custom_Data_Adapter)
  + [Location of the Class](https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasource-adapter#Location_of_the_Class)
  + [Creation of the Data Adapter](https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasource-adapter#Creation_of_the_Data_Adapter)
  + [Creation of the Report](https://community.jaspersoft.com/wiki/how-create-and-use-jrdatasource-adapter#Creation_of_the_Report)

**Overview**

Sometimes it is necessary to reterive information from sources unsupported by any data adapters. In this case you can write your own java class to connect and query these sources, and then elaborate the extracted information to make them compatible with JasperReport. To do this you must define a custom data adapter specifing its class that implements the JRDataSource interface and some other information. Doing this, JasperReport will use this class to extract the information and will use it to fill the report.

In this example will see how to create a custom JRDataSource adapter. The code in this example will be pretty simple, because all the information provided by the data adapter is hardcoded in the adapter itself. This means that the information provided is embedded and for this reason there is no need of other resources. This is a simplistic approach to show how to create a custom data adapter without creating too complex an example.

**The Class of the Custom Data Adapter**

The first thing to do is decide which data is returned and how it is structured. Lets suppose that we have a series of records where every record is a name and an age. So we need to return this information. Now we need to understand how a data adapter is composed. A custom data adapter is a piece of java code with some characteristics:

* A class that implements the interface JRDataSource. This is necessary to guarantee the presence of the methods required to retrieve the data. To implement this interface we need to define two methods:

1. **public boolean next():** this method tell to JasperReport if there are still records to read. If it returns true, there are other records, otherwise it must return false.
2. **public Object getFieldValue(JRField jrField):** We already know that a record can have any number of fields. This method is called for every field in the report, and it must return a value for that field. The parameter received by this method is the field that needs to be valorized, and it contains the name of the field, the description and, in general, information about the field. Knowing this information could be useful to identify the field and return the appropriate data.

Keep in mind that these methods are called for every record. At first the method next is called and if its result is true then the getFieldValue is called for every field in the report. Then the method next is called again and the cycle will be repeated until it returns false.

* Then you have to define a static method that returns an instance of the class defined in the first point. This is necessary to provide to JasperReport a way to get an already built instance of the class.

 Now that the concept behind this custom data adapter is explained, we can see the code (the code is also attached to this tutorial):

**Location of the Class**

At this point we need only to understand where put this class and how use it. From the designer, right click on a JasperReport project folder and select New -> Package (if you don't see the element package search it under others), then use the name CustomDataAdapter for the new package.

At this point right click on the CustomDataAdapter package and select New -> Class. On the dialog that will appear insert MyImplementation as class name. Then press the button Add to add a new interface and select JRDataSource, if you have done right the JRDataSource interface will appear in the list of the used interfaces. Finally hit the Finish button to create the new class. At this point you have to write the data adapter code, but for this example you can just copy and paste the code written before.

**Creation of the Data Adapter**

Now you have to create the data adapter that use this class, from select the element File -> New -> Data adapter.

From the dialog select the same project folder where you put the class (in this case MyReports) and has name of the file put MyCustomAdapter and hit Next. Now you have to choose the type of the data adapter, select Custom Implementation of JRDataSource and hit Next. At this point you must provide the information to get the class previously written:

* As name of the datasource you can put anything you like, in this case we can use MyDataSource.
* On the factory class hit the button with three dots "...", and in the new dialog search for MyImplementation and hit Ok. At the and in the Factory Class field you should find something like CustomDataAdapter.MyImplementation.
* In the second textfield you must insert the static method that return an instance of your class, in this case simply type getDataSource.

After you have compiled all the fields hit the Test button to check if it is all right, and if you obtain a successful response hit the Finish button.

**Creation of the Report**

At this point you can use this new data adapter inside a report. Create a new report from the report wizard (File -> New -> Jasper Report), for this example use the template Coffee. Place it in the same Project folder of the data adapter (MyReports) and as data adapter select MyDataAdapter and hit Finish. Now create two fields (In the outline view right click on the element Fields and select  Create Field, one time for each field), and drag and drop them in the detail band. Then **you have to change the Class property of the second field to java.lang.Integer**, since by default it is String, but our data adapter return a string for the first value and an integer for the second one. If you don't do this then you will get a ClassCastException from JasperReports when you will run the report, becuase it is not able to do this conversion automatically. Probably you will have to adjust the size of the band and of the frame inside it (you could also remove this frame). Anyway ad the and you will obtein something like this:

If you don't have the Automatic Build of your projects enabled (and by default it is disabled), you need to build your class file. To do this simply right-click on your project and select the option "Build Prjoect", if there aren't errors then you can proceed with the preview.

NOTE: If after building your project you still get ClassNotFound exeption when executing the report try to close and reopen Jaspersoft Studio and rebuild the project, because in some rare cases it could be possible that the Java Classloader is not able to load correctly a just created class.

Finally switch to the Preview tab to compile the report and see the result:

Attachments:

[Code of the class my implementation](https://community.jaspersoft.com/sites/default/files/wiki_attachments/myimplementation.java)

[sampleproject.zip](https://d2553lapexsdrl.cloudfront.net/sites/default/files/wiki_attachments/sampleproject.zip)