Using Interpreted Data Sources

An interpreted data source is a tabular data source that is regarded by SOAtest as a relational repre-sentation of a Java object graph. An interpreted data source can be used to facilitate creation of multiple Java objects and object graphs that can be used as test parameter inputs by the EJB Client Tool and other SOAtest tools. For more information on the EJB Client tool, see EJB Client.

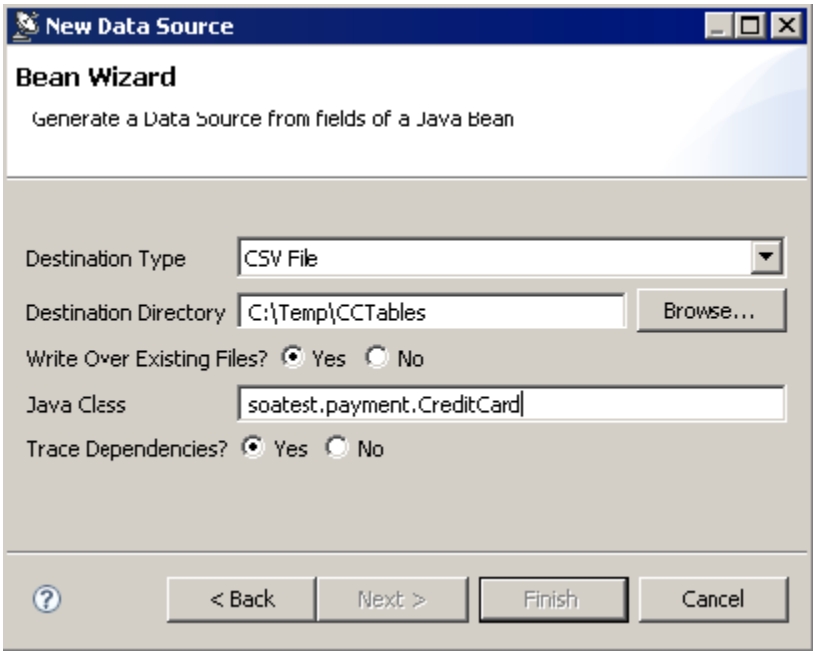
Generating a Data Source from the fields of a Java Bean

To generate a Data Source from the fields of a Java Bean:

1. Do one of the following:
   - For test suite level data sources, select the desired test suite node and click the **Add Database** toolbar button.

   - For project level data sources, right-click the related project's Test Case Explorer node, then choose **Add New> Data Source**.
   - For global level data sources, right-click the Test Case Explorer's Global Data Sources node, then choose **Add New> Data Source**.
2. Select **Bean Wizard** and click **Next.** The **Bean Wizard** dialog displays.



3. Complete the following options in the Bean Wizard dialog:
   - **Destination Type:** Select the type of table template you would like to create from the drop-down menu.
   - **Destination Directory:** Specify where the tables will be written.
   - **Write Over Existing Files:** Specify whether you want to overwrite existing files.
   - **Java Class**: Specify the class for which you would like to create a tabular representation.
   - **Trace Dependencies:** Select either **Yes** or **No** for trace dependencies. Selecting **Yes** prompts SOAtest to generate tables for types of class member variables "reachable" from the "root" class that you specified in the Java Class field.
4. Click the **Next** button. The **Dependencies** dialog displays.
5. Select the desired type dependencies from the **Generate Table for Classes** list.
6. Click the **Finish** button to generate the tables. The tables will be created in the designated directory and a data source for each file will be added to the test suite you selected.

Interpreted Data Source Table Format

The following are the main concepts of the relational to object mapping used by SOAtest:

- An object is a row in a table.
- There are two types of tables:
  - Class Tables
    - Column for each class member variable
    - Row for each instance
  - Collection Tables
    - Column for table and row of actual object
    - Row for each instance

Object References

The first column in each table is an identifier for object instances. An object may be referenced by the table name followed by a space, followed by the object identifier. Identifier column need not necessarily be row numbers, so long as the identifier values are unique within each table.

Values and References

Field values of non-primitive classes are generally specified through object references as described above. However, the extra level of indirection is unnecessary and cumbersome for values of primitive types. To accommodate an abbreviated form, a built-in support is included for inlining commonly used types with well-defined string representations. An empty cell in a reference column is interpreted as the null value. An empty cell for a value column is interpreted as an empty string.

Collections

In order to support variable sized collections, a collection table is introduced. A collection table has a single reference column. An object in the collection is referenced by the table name followed by a space, followed by the object identifier.

Example

As an example, let us consider an object graph with `CreditCardDO` as a root. `CreditCardDO` contains instances of `PersonDO`and `AddressDO` and a `Vector of ActivityDO` type objects.

```
public class CreditCardDO extends PaymentMethodDO implements Serializable
{
    protected String ccNumber;
    protected Date expirationDate;
    protected PersonDO ccHolder;
    protected AddressDO billingAddress;
    protected Vector recentActivity = new Vector();

    // set…()/get…() methods omitted
}
public class PaymentMethodDO implements Serializable {
    protected String bankName;

    // set…()/get…() methods omitted
}
public class AddressDO implements Serializable {
    protected String streetAddress;
    protected String city;
    protected int zipCode;
    protected String state;

    // set…()/get…() methods omitted
}
public class PersonDO implements Serializable {
    protected String firstName;
    protected String lastName;

    // set…()/get…() methods omitted
}
public class ActivityDO implements Serializable {
    private float amount;
    private String description;

    // set…()/get…() methods omitted
}
```

The following tables illustrate how the above object graph example can be represented in tabular format:
Table CreditCardDO.csv

soatest.examples.CreditCardDO	bankName	billingAddress ref	ccHolder ref	ccNumber	expirationDate	recentActivityref
1	SampleBank	AddressDO 1	PersonDO 1	1234123412341234	8/31/2005	RecentActivities-1

Table AddressDO.csv

soatest.examples.AddressDO	city	state	streetAddress	zipCode

1	Los Angeles	CA	101 E. Huntington Dr.	91016

Table PersonDO.csv

soatest.examples.PersonDO	firstName	lastName
1	Donald	Duck

Table ActivityDO.csv

soatest.examples.ActivityDO	amount	description
soatest.examples.ActivityDO	amount	description
1	10	10 Charge-10
2	20	20 Charge-20
3	30	30 Charge-30
4	40	40 Charge-40
5	50	50 Charge-50
6	60	60 Charge-60
7	70	70 Charge-70
8	80	80 Charge-80
9	90	90 Charge-90
10	100	100 Charge-100

Table RecentActivities-1.csv

ActivityDO ref
ActivityDO 1
ActivityDO 2
ActivityDO 3