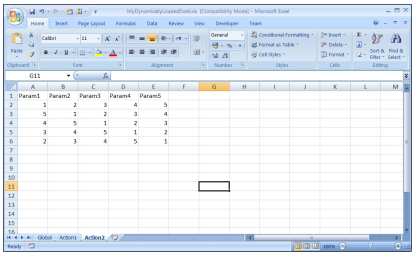
Importing an Excel file to a test

We can dynamically set the underlying Excel file that will serve as a data source for the whole run session, though this is probably a very rare case, and even switch between such files during the run session is possible to import a whole Excel workbook for a test or just a single worksheet for a specific action.

The classical case of importing an Excel file to a test is when the san flow needs to be executed on different environments, such as with multilingual systems. In such a case, the test would require an extern parameter to identify the environment, and then load the correct Exce file. Another possibility is that the test identifies the language dynamically, for example, by retrieving the runtime property value of **Test Object** (**TO**), which indicates the current language, or by retrievi the `lang` attribute of a web page or element.

Getting ready

Ensure that a new test is open and create a new action. Ensure that a external Excel sheet exists with one global worksheet and worksheets named after each action in the test. The Excel sheet will contain three worksheets, namely, `Global`, `Action1`, and `Action2`. The `Action2` worksheet will contain data shown in the following screenshot. In our example, we will use the Excel sheet named `MyDynamicallyLoadedExcel.xls`, and to simplify matters, we will put it under the same test folder (it should be placed in a separate shared folder):



In the **Flow** pane, make sure that the **Action Call** properties are set to **Run on all rows**.

How to do it...

In order to load the `MyDynamicallyLoadedExcel.xls` file to the test, perform the following steps:

1. We use the `DataTable.Import` method to load the Excel sheet. In `Action1` (the first to be run), we use the following code to ensure that the Excel file is loaded only once (to avoid loading Excel for each iteration in case the test is set to **Run on all rows**):

```
Print "Test Iteration #" & Environment("TestIteration") & " - " & E
if cint(Environment("TestIteration")) = 1 then
    DataTable.Import("MyDynamicallyLoadedExcel.xls")
end if
```

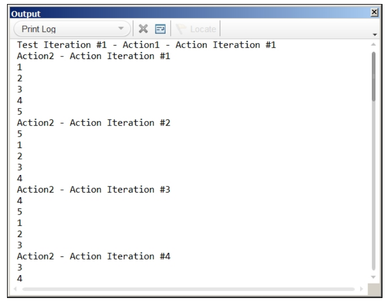
2. In `Action2`, we use the following code to retrieve the values for all parameters defined in the local datasheet for `Action2`. We first print the number of the current action iteration, so that we may

distinguish between the outputs in the console.

```
Print Environment("ActionName") & " - Action Iteration #" & Enviror
For p = 1 to DataTable.LocalSheet.GetParameterCount
    print DataTable.LocalSheet.GetParameter(p)
Next
```

3. When a test is set to **Run on all rows**, it means that it will be executed repeatedly for each row having data in `GlobalSheet`.

   The output to the console looks like the following screenshot:



How it works...

In `Action1`, the `DataTable.Import` method replaces `Default.xls` with the target Excel file. The code in `Action2` retrieves and prints the values for each parameter, and as the action was set to **Run on all rows**, the code repeats this for all rows with data.

There's more...

To import just a worksheet for an action, use the `DataTable.ImportSheet` method as follows:

```
DataTable.ImportSheet("MyDynamicallyLoadedExcel.xls", "Action1", "Action1
```

Here, the first parameter is the Excel filename and the last two are the source datasheet and target datasheet respectively.

See also

For information on saving values collected during a run session, refer to the next recipe, *Exporting a DataTable*.