QlikView

QlikView Expressor Extensions SDK Tutorial: Using Schema import options

QlikView Expressor Version 3.10

Newton, Massachusetts, August 2013

Authored by James Siwila

This tutorial exercise demonstrates how to create import options for Extension Schemas. As in the previous exercise in this series, "Tutorial: Creating Write operators and using special fields," this exercise uses the Record Special and Field Special settings in the Extension Object Schema Descriptor to enable the Schema to specify the character to recognize as the field delimiter when reading a CSV file.

This tutorial requires you to use QlikView Expressor 3.10 because the new version contains a fix to a bug that would prevent the exercise from working successfully.

The tutorial builds on the first two tutorial exercises published on the Community site, "Schema from Type Tutorial Exercise" and "Tutorial: Creating Write operators and using special fields," and on the exercises in the online help for QlikView Expressor 3.9 and 3.10. Before attempting to complete this tutorial, you should have a working knowledge of the Extensions SDK and complete at least the first three tutorial exercises in the online help. You must also have worked through the Schema-from-Type and the Write-operators-with-special-fields tutorials.

Values in CSV files can be separated by a number of different characters, though only one character can be used as a field delimiter in a given file. In this exercise, we add a special value to the Extension Object Schema Descriptor used in the previous tutorial ("Tutorial: Creating Write operators and using special fields") that allows a user creating an Extension Schema to specify a field delimiter different from the default delimiter.

1. Add special values to the Schema Descriptor

As discussed in the previous tutorial, the QlikView Expressor Extension SDK provides for record and field values that store special metadata, and among other things, those values can be used to store information about the field delimiter used to read certain files.

1. Open the Schema Descriptor named "CsvSchemaDescriptor."

   This is the Object Schema Descriptor created in *3. Read Operator with Schema Exercise* in the QlikView Expressor 3.9 and 3.10 online help and modified in the "Creating Write operators and using special fields" tutorial.

2. Select the Metadata Import and select the  button in the Field import options section to add an option.

3. Use the following values for the new option:

   Name: csv_separator
   Display Name: Field separator
   Description: The separator used in the selected CSV file.
   Data Type: string
   Multi Values: *leave blank*
   Default Value: , (*a comma*)
   ***:** *leave blank*
   Update Group: G1

   Update Group is the name assigned to the Schema properties that can be used by an operator as the default values for some properties. The name used in the Operator Descriptor's field must match the name assigned in an Object Schema Descriptor in the Extension. For example, The QlikView Expressor Excel Extension's Read Excel operator has a Use Default Values button in its properties panel that takes the values for properties such as Top left cell for header from the Schema used by the operator.

4. Save the Schema Descriptor.

5. Use the **Go To Function** button on the Function tab of the Descriptor Editor to open the `CsvReadMetadata_DSM` Datascript Module.

6. Modify the `GetFieldList` function as follows (changes in **bold red**):

```
function GetFieldList(session, objectNames, schemaImportOptions, contex
t, constructionMetadata)
  local schemas = {}
  local discoveredFields = {}
  local fullFileName = session.fullFileName
  local shortFileName = string.substring(fullFileName, #fullFileName -
string.find(string.reverse(fullFileName),"\\")+2, #fullFileName)
  local file = io.open(fullFileName, 'r')
  if file ~= nil then
    --read the first line with a header
    local headerLine = file:read()
    if headerLine ~= nil then
      --split the record by comma
      local separator=''
      --schemaImportOptions is a collection of options for each
separate object
          --since we are using 1 file - 1 schema mode, there is only
one object and only one set of import options.
      local importOptionsOfCurrentObject = schemaImportOptions[1]

      if Common.Utilities.trim(importOptionsOfCurrentObject.csv_separat
or) == ',' then
          separator = ','
      elseif Common.Utilities.trim(importOptionsOfCurrentObject.csv_sep
arator) == ';' then
          separator = ';'
      else
          return 3, 'Incorrect separator value, only comma or semicolon
are supported.'
      end

      headerColumns = Common.Utilities.Split(headerLine, separator)
      for i, v in ipairs(headerColumns) do
        table.insert(discoveredFields,
          {
           name = Common.Utilities.trim(v),
           dataTypeName = 'CSV_STRING',
           size = 0,
           allownulls = false,
           customizable = false,
           fieldSpecial = {originalPosition=i}
          }
          )
      end
      --create a single schema
      local csvSchema = {
        sourceName = session.fullFileName,
        customizable = false,
        recordSpecial = {createdFromFile=true, csv_separator=Common.Uti
lities.trim(importOptionsOfCurrentObject.csv_separator) },
        fields = discoveredFields
       }
       --add schema to schema set
       table.insert(schemas, csvSchema)
       --close file
       file:close()
       --return success code and schemas set
       return 0, schemas
     else
```

```
            return 2, "Error reading CSV header from the CSV file."
          end
        else
          return 1, "Error: the selected file cannot be opened."
        end
      end
```

7. Save the Datascript Module.

These changes cause the Extension Schema to check the separator to see which is used and validate that it is either a comma or semicolon. It then places the specified separator (`csv_separator`) into the `recordSpecial` section where it can be retrieved by an operator if necessary.

2. Create a CSV Schema from the modified Schema Descriptor

Now that the new field import option has been added to the Schema Descriptor and the `recordSpecial` values have been modified in the Datascrpt Module, we can create a new Schema that uses the new feature.

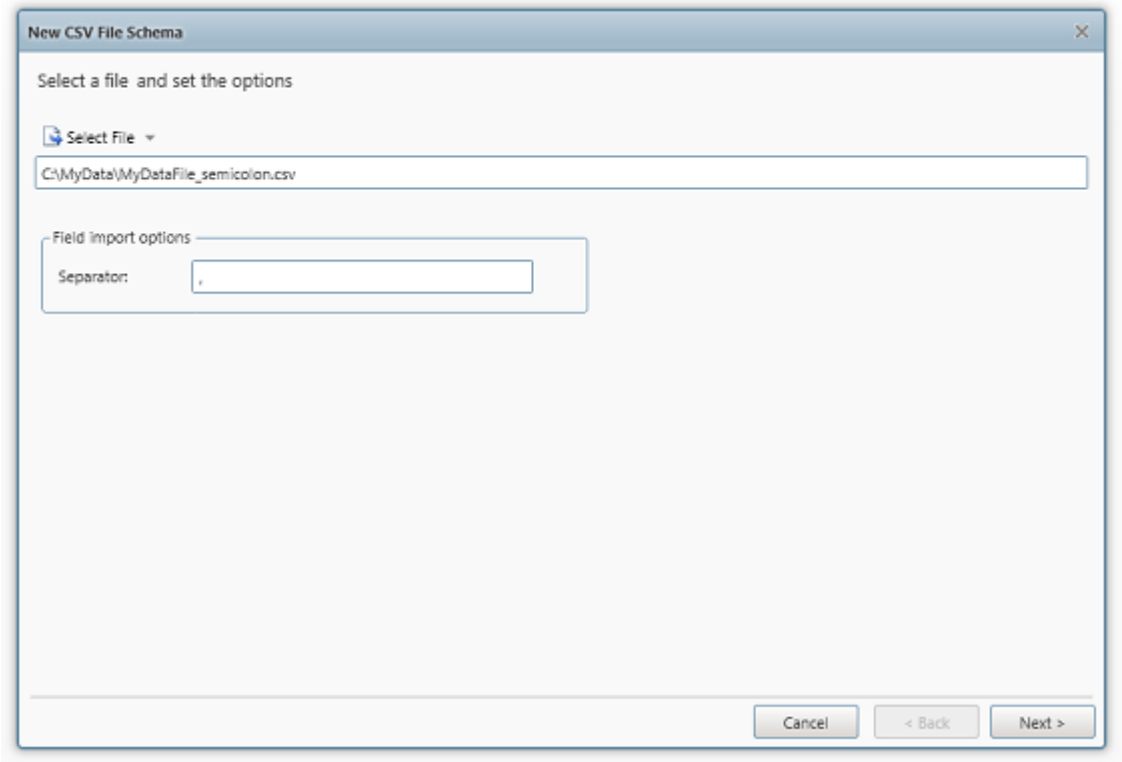
First, alter the data file MyDataFile.csv by replacing the commas used as record separators with semicolons:

Item;Cost;Sold;Profit
Keyboard;10;16;6
Monitor;80;120;40
Mouse;5;7;2

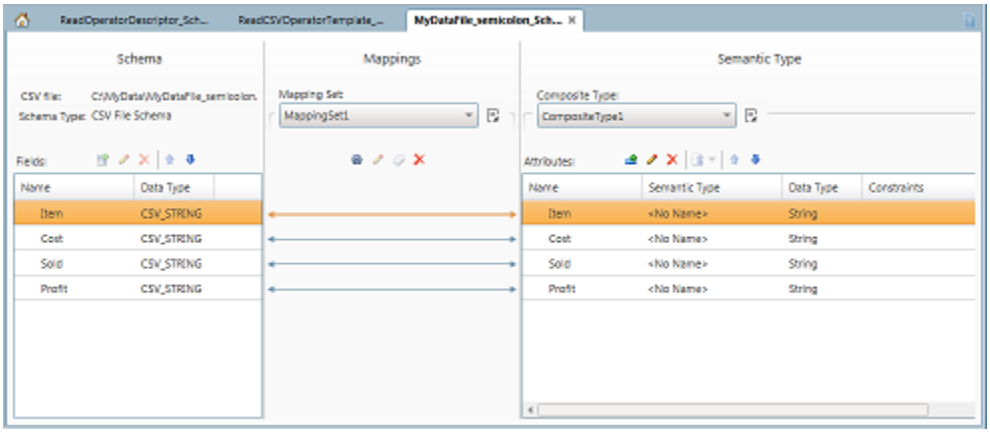
Save the changed file as "MyDataFile_semicolon.csv."

1. In Project1 in the Hello_World_Sample Workspace, right-click on the Schemas folder and select **New>CVS File Schema>From Data Source**.
2. Select the new CSV file "MyDataFile_semicolon.csv."
   Note that the New CSV File Schema wizard now has a Field import options box in which to

specify the field separator character in the data source file.



3. Replace the comma in the Separator text field with a semicolon.
4. In the next wizard panel, name the Schema "MyDataFile_semicolon_Schema."
5. Open MyDataFile_semicolon_Schema to see that it was created as expected from the data in "MyDataFile_semicolon.csv."



3. Add a CSV separator property to the Read Operator Descriptor

1. Open the "ReadOperatorDescriptor_SchemaConnection1" Descriptor created in the third exercise in the online help titled "3. Read Operator with Schema Exercise."
2. Select the Properties tab and select the button to add a property.
3. Use the following values for the new property:

   Name: csv_separator
   Display Name: Separator
   Description: The separator used in the selected CSV file.

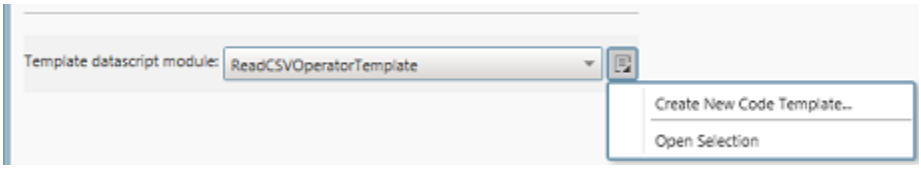
Data Type: string
Multi Values: *leave blank*
Default Value: , (*a comma*)
***:** *leave blank*
Update Group: G1

Using the same Update Group for `csv_separator` in both the Schema Descriptor and Operator Descriptor allows the operator to use the setting for the separator in the Schema instead of the operator's default setting.

3. Save the "ReadOperatorDescriptor_SchemaConnection1" Descriptor.

4. Modify the Read operator template DSM

1. Switch to the General tab of the "ReadOperatorDescriptor_SchemaConnection1" Descriptor.
2. Click the Action icon next to the drop-down list labeled **Template datascript module** and select **Open Selection**.



3. Modify the `initialize` function as follows (changes in **bold red**):

```
function initialize()
  file = io.open(_expCurrentOperatorParameters.Path ..'\\'.. _expCurren
tOperatorParameters.file_name)
  if (file == nil) then
    --error(_expCurrentOperatorParameters.Path)
    --error(_expCurrentOperatorParameters.file_name)
    error('the specified file can not be opened')
  end
  -- read the CSV header line (first line in a file)
  local csvHeaderLine = file:read()
  if csvHeaderLine == nil then error('error reading file header')
  end
  csvHeader = Common.Utilities.Split(csvHeaderLine, _expCurrentOperator
Parameters.csv_separator)
end;
```

4. Modify the `read` function as follows (changes in **bold red**):

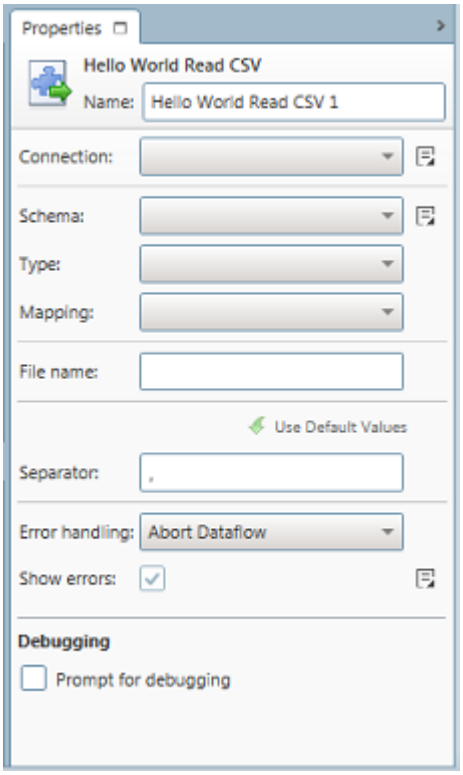
```
function read()
  -- Provide the main read loop to "read" input data and produce
  -- records or otherwise indicate operator status.

  -- NOTE: A nil return value indicates an error condition. If left
  -- unchanged, this will result in a runtime error.
  -- A true return value terminates the operator.

  csvLine = file:read()
  if csvLine ~= nil then
    local resultRow = {}
    --split values by comma
    csvRow = Common.Utilities.Split(csvLine, _expCurrentOperatorParamet
ers.csv_separator)
    for i, v in ipairs(csvRow) do
```

5. Save the operator template DSM.

5. Test the new Update Group setting

1. In Project1 in the Hello_World_Sample Workspace, right-click on the Dataflows folder and select **New**.
   Name the new Dataflow: `SchemaImportOption_test`.

2. Drag a Hello World Read CSV operator onto the Dataflow pallete.
   Note that the operator Properties panel now includes a Separator field and a Use Default Values button. The Separator field contains a comma because that is the default value specified in the "ReadOperatorDescriptor_SchemaConnection1" Descriptor.



3. Configure the Hello World Read CSV 1 in the Properties panel as follows:

   Connection: Choose the `FileConnection1` Connection created in an earlier exercise. The `MyDataFile_semicolon.csv` file must be in the directory pointed to by `FileConnection1`.

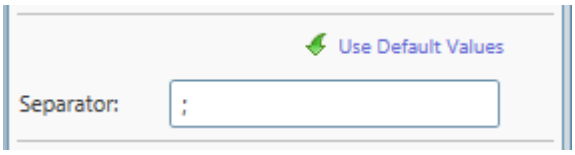
   Schema: Choose the "MyDataFile_semicolon_Schema" created above.

   Type: Leave the default type, CompositeType1

   Mapping: Leave the default, MappingSet1

   File name: Enter `MyDataFile_semicolon.csv`.

   Separator: Here is where you want to get the separator used by the "MyDataFile_semicolon_Schema." Click on Use Default Values to insert the separator used by the Schema and the semicolon is placed in the Separator field.



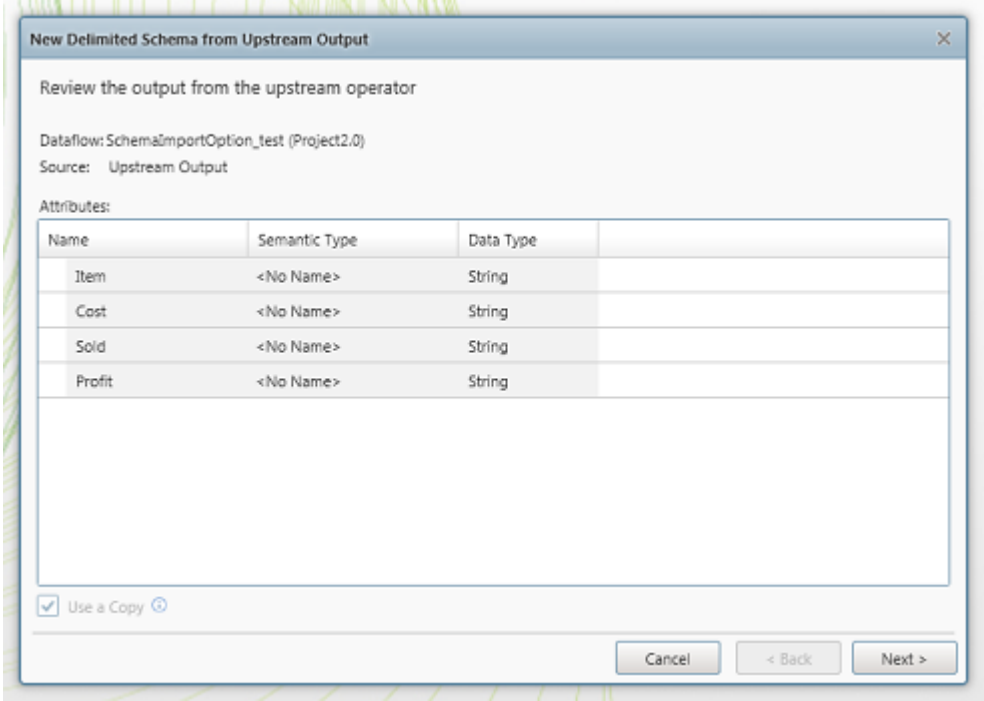
   Error handling: Leave the default: Abort Dataflow.

   Show errors: cannot be changed.

   Debugging: *Leave unchecked.*

4. Drag a Write File operator onto the Dataflow pallete and connect it to the Hello World Read CSV 1 operator.

5. Click the Action icon next to the Schema drop-down list in the Write File operator Properties panel and select **New CSV File Schema from Upstream Output**.
The New CSV File Schema from Upstream Output dialog box shows the fields that will be created for the new Schema. They are the same as the "MyDataFile_semicolon_Schema" created above.



6. Use the default name for the new Schema: DelimitedSchema1.
Note that the default Field Delimiter for Delimited Schemas is the comma. So the output of the Dataflow will have commas as field separators even though the input file uses semicolons as field delimiters.

7. Complete the remaining properties for the Write File operator as follows:

   Connection: Choose the `FileConnection1` Connection used above for the Hello World Read CSV operator.

   Type: Leave the default type: CompositeType1.

   Mapping: Leave the default: MappingSet1.

   File name: Enter `MyDataOutput_semicolon.csv`.

   Quotes: Leave the default: No quotes.

   Include header: Check to include a header in the output file.

   Append to output: *Leave unchecked.*

   Append timestamp to filename: *Leave unchecked.*

   Error handling: Leave the default: Abort Dataflow.

   Show errors and **Partition count** cannot be changed.

8. Save the Dataflow.

9. Select the **Start** button on the Dataflow Build tab of the Desktop ribbon bar.
If the Dataflow does not run successfully, review the error messages for indications of where the errors occur. Check your work in the previous steps, especially in the Datascript Modules.

10. If the Dataflow completes successfully, open the newly created file, `MyDataOutput_semicolon.csv`.
    The output should look the same as the input, except that the field separators are commas instead of semicolons because the Schema used with the Write File operator uses the default separator for Delimited Schemas, which is the comma:

    Item,Cost,Sold,Profit
    Keyboard,10,16,6
    Monitor,80,120,40
    Mouse,5,7,2

    If you want to change the output file delimiter to be the same as the input, open DelimitedSchema1 and select semicolon from the Field Delimiter drop-down list. Save DelimitedSchema1 and rerun the Dataflow. This time you see the output file has the semicolon delimiter.

    Item;Cost;Sold;Profit
    Keyboard;10;16;6
    Monitor;80;120;40
    Mouse;5;7;2