

Create custom operators for WebSphere DataStage

Skill Level: Intermediate

[Blayne Chard](#)
Software Engineer
IBM

08 Feb 2007

Create a simple DataStage operator, then learn how to load the operator into DataStage Designer. An operator is the basic building block of a DataStage job. Operators can read records from input streams, modify or use the data from the input stream, and then write the results to a output stream.

Section 1. Before you start

About this tutorial

This tutorial gives you an introduction to creating a basic DataStage operator. You'll start by learning how to write a basic operator, and then walk step-by-step through the process of loading the operator into the DataStage Designer.

Objectives

In this tutorial you learn:

1. How to write a simple DataStage operator
2. How to set up the development environment to compile and run a DataStage operator

3. The basics of the Orchestrate Shell (OSH) scripting language for DataStage jobs
4. How to load your operator into the DataStage Designer so you can use it on any job you create

Prerequisites

This tutorial is written for Windows programmers whose skills and experience are at an intermediate level. You should have a solid understanding of IBM WebSphere DataStage and a working knowledge of the C++ language.

System requirements

To run the examples in this tutorial, you need a Windows computer with the following:

- Microsoft Visual Studio .NET 2003
- IBM WebSphere DataStage 8.0
- [MKS Toolkit](#)

Before you begin

Before you start this tutorial, refer to the [Download](#) section and download the source code for this tutorial. Extract this example to a simple location, as you will be accessing it frequently. In this tutorial, the directory `e:/osh/` has been used, so change any reference to this location to the location of your source code directory.

Inside the download code.zip archive, there are seven files:

- **/setup.bat** -- Batch script to setup the environment
- **/myhelloworld.osh** -- OSH script used to run the operator
- **/make.bat** -- Batch script to compile and link the operator
- **/operator.apt** -- Operator configuration file
- **/src/myhelloworld.c** -- The source code for the operator
- **/input/mhw.txt** -- The input file used for running the operator

- **/output/mhw.txt** -- The output file used for running the operator
-

Section 2. Create your first operator

MyHelloWorld

The first operator takes one input stream and one output stream. MyHelloWorld, takes a single column as input, an integer, this integer determines how many times "Hello World!" is printed into one of columns in the output stream. The output stream consists of two columns, a counter showing how many times "Hello World!" was printed, and the printed result. To go with the input and output streams, there is one option "uppercase." This option determines if the text "Hello World!" is printed in uppercase or not.

Describe the input and output

The input and output schemas can be described as follows:

- Input stream columns:
 - **inCount - int32** -- Number of times "Hello World!" should be output
- Output Stream Columns:
 - **outCount - int32** -- Number of times "Hello World!" was printed
 - **outHello - String** -- "Hello World!" printed outCount number of times
- Parameters
 - **uppercase - boolean** -- Whether "Hello World!" should be printed in uppercase or not (optional)

There are a few pieces of code to describe this inside the operator. The parameter description is usually defined first. This tells the operator the parameters to expect. This includes how many input streams and output streams, and all the parameters that can be passed into this operator.

The parameter description shown below, tells the operator that there is one parameter, "uppercase", which is optional, then goes on to tell it that there is only one input and one output stream.

```

#define HELLO_ARGS_DESC \
"{ uppercase={optional, " \
" description='capitalize or not'" \
" otherInfo={ " \
" inputs={ " \
" input={ " \
" description='source data for helloworld', " \
" once" \
" } " \
" }, " \
" outputs={ " \
" output={ " \
" description='output data for helloworld', " \
" minOccurrences=1, maxOccurrences=1 " \
" } " \
" }, " \
" description='myhelloworld operator:' " \
" } " \
"}"

```

Describe the operator

DataStage operators are all extensions of one of three base classes:

1. **APT_Operator** -- The basic operator
2. **APT_CompositeOperator** -- Contains one or more operators
3. **APT_SubProcessOperator** -- Used to wrap around third party executable's

You only need the functionality of the `APT_Operator`, as you only need one operator. `APT_CompositeOperator` is not needed, and you are not wrapping this operator around a third party process, so `APT_SubProcessOperator` is not required. To use this base class, you must implement two virtual methods, `describeOperator()` and `runLocally()`. Also as you have an input parameter `uppercase`, a third method, `initializeFromArgs_()`, is required.

The basic class definition for the `MyHelloWorld` operator looks like the code shown below:

```

class APT_MyHelloWorldOp : public APT_Operator {
    APT_DECLARE_PERSISTENT(APT_MyHelloWorldOp);
    APT_DECLARE_RTTI(APT_MyHelloWorldOp);
public:
    APT_MyHelloWorldOp();
    ~APT_MyHelloWorldOp();

    void setUpperCase(bool uppercase);

protected:
    virtual APT_Status initializeFromArgs_(const APT_PropertyList &args,

```

```

        APT_Operator::InitializeContext context);
    virtual APT_Status describeOperator();
    virtual APT_Status runLocally();
private:
    bool uppercase_;
};

APT_DEFINE_OSH_NAME(APT_MyHelloWorldOp, myhelloworld, (APT_UString)HELLO_ARGS_DESC);
APT_IMPLEMENT_RTTI_ONEBASE(APT_MyHelloWorldOp, APT_Operator);
APT_IMPLEMENT_PERSISTENT(APT_MyHelloWorldOp);

```

The macros in this definition play a important role in defining the operator.

- `APT_DEFINE_OSH_NAME` -- This macro defines the OSH name of this operator. The OSH name is the way DataStage references operators and is used whenever this operator is referenced.
- `APT_IMPLEMENT_RTTI_ONEBASE` and `APT_DECLARE_RTTI` -- These macros set the runtime type identification for your operator.
- `APT_IMPLEMENT_PERSISTENT` and `APT_DECLARE_PERSISTENT` -- These macros tell you that this operator can be serialized and moved to a processing node.

Use the input and output streams

Before the operator is able to use the input and output streams, the operator needs more information about them. The setup for the streams is done inside the `describeOperator()` function.

To set up the interface for the streams, the operator needs to know what type of data to expect in each of the streams. This is specified using `setInputInterfaceSchema()` and `setOutputInterfaceSchema()`. Both of these methods take two parameters, an `APT_String`, and an integer. The `APT_String` is a schema; the integer is an index indicating which input or output stream to apply the schema to. Below is the `describeOperator()` that is used inside the `MyHelloWorld` operator.

```

APT_Status APT_MyHelloWorldOp::describeOperator(){
    setKind(eParallel);

    // Set the number of input/output links
    setInputDataSets(1);
    setOutputDataSets(1);

    // Set the schema for the input link
    // inCount:int32 requires the first column of the input stream to be of type int32
    setInputInterfaceSchema(APT_UString("record (inCount:int32;)", 0);

    // setup the output link
    // sets the output link to have two columns a integer column outCount and a
    // string column outHello

```

```

    setOutputInterfaceSchema(APT_UString("record (outCount:int32;outHello:string;)", 0);
    return APT_StatusOk;
}

```

Use the input parameters

To access the input parameter you are using in this operator, there are three methods you must implement:

- `initializeFromArgs_()`
- `serialize()`
- `setUppercase()`

`initializeFromArgs_()` is called when the operator is first run. It receives a list of parameters that have been passed into the operator. Here you have to look for the parameter you are using, "uppercase." To do this, cycle through all the parameters that have been passed in. If you find the "uppercase" keyword, you then set it's value.

```

APT_Status APT_MyHelloWorldOp::initializeFromArgs_(const APT_PropertyList &args,
    APT_Operator::InitializeContext context){
    APT_Status status=APT_StatusOk;
    if (context == APT_Operator::eRun) return status;

    for (int i = 0; i < args.count(); i++){
        const APT_Property& prop = args[i];
        if (prop.name() == "uppercase"){
            uppercase_ = true;
        }
    };

    return status;
}

```

The method `serialize()` is used when the operator is going to be moved to a processing node. It can also be used when the operator is getting parallelized over multiple nodes so that each operator contains the same information about the parameters. To serialize your parameter "uppercase", you have to use an overloaded method on the `APT_Archive` class, by OR'ing (`||`) a variable with a `APT_Archive`, the variable is then stored inside the archive, as shown in the code below.

```

void APT_MyHelloWorldOp::serialize(APT_Archive& archive, APT_UInt8) {
    archive || uppercase_;
}

```

You need an initialization method `setUppercase()`

```
void APT_MyHelloWorldOp::setUpCase(bool uppercase) {
    uppercase_ = uppercase;
}
```

Finally, you need to add the default value to the constructor.

```
APT_MyHelloWorldOp::APT_MyHelloWorldOp() : uppercase_(false) {
    ...
}
```

After all these steps have finished, you can access `uppercase_` like any other local variable.

MyHelloWorld's main method

The `runLocally()` method is called when the operator is run. This method houses most of the logic behind the operator.

Warning: This method can be invoked in parallel if multiple instances of this operator are spawned. They can interfere with each other if they interact with external objects such as files or databases.

Access input and output streams

To allow the `runLocally()` method to read the input stream and write to the output stream, you have to setup some access cursors; one for each stream, as shown below.

```
APT_Status APT_MyHelloWorldOp::runLocally() {
    ...

    // Allows access to the input dataset, read only can only move forward
    APT_InputCursor inCur;
    setupInputCursor(&inCur, 0);
    // allows access to output dataset
    APT_OutputCursor outCur;
    setupOutputCursor(&outCur, 0);

    ...
}
```

After setting up the cursors, you can then have direct access to the data in the streams. To get this access you use an accessors class, there is one class per data type. These accessors classes have most of their basic operator's overloaded (+,=,*,-,/) so you can assign and change their values. A few examples are given below

```
*fieldlout = *fieldlin;
*fieldlout = *fieldlin * 2;
*fieldlout = *fieldlin + 5;

*field2out = APT_String("Hello");
```

To set up the accessors, you initialize them by referencing the column name and the input or output cursor that the column is in. The accessors for your input and output columns and for your operator are shown below.

```
APT_Status APT_MyHelloWorldOp::runLocally() {
    ...

    APT_InputAccessorToInt32 fieldlin("inCount",&inCur);
    APT_OutputAccessorToInt32 fieldlout("outCount",&outCur);
    APT_OutputAccessorToString field2out("outHello",&outCur);

    ...
}
```

Before attempting to access the data behind these accessors, you have to start accessing the data inside the input streams. You do this by using the input cursor's `getRecord()`. This method gets the next record from the input stream and loads all the record's values into the accessors. You can then begin using the accessors.

Once you have finished with a row on the output accessors, you need to call `putRecord()`. This flushes the output accessor's record into the output stream.

Add logic

Finally, by adding the logic behind the `runLocally()` method, you can finish off your operator.

First, you should add some logic based on the input parameter "uppercase." Here you have a simple if statement to decide if you should use an uppercase version of "Hello World!".

Next, you loop through all the data on the input stream. You can use the input cursor's `getRecord()` to exit the loop as it returns a boolean of true if there is still more data on the link, and false when it has reached the end of the records.

For every record you loop through you output one record into the output stream using the output stream's `putRecord()`.

```
APT_Status APT_MyHelloWorldOp::runLocally(){
    ...
    APT_String hello;
    if(uppercase){
        hello = APT_String("HELLO WORLD!");
    }else{
```

```
        hello = APT_String("Hello World!");
    }
    // loop through all the records
    while(inCur.getRecord()) {
        // output the input
        *field1out = *field1in;

        // print out Hello World!
        APT_String fout = APT_String("");
        for(int i =0; i< *field1in; i++){
            fout += hello;
        }
        *field2out = fout;

        //output the row the output cursor
        outCur.putRecord();
    }

    return status;
}
```

Section 3. Compile your operator

Configure environment variables

Because most of the development and testing is done outside of DataStage, Windows is unable to find specific files required to compile and execute the operator. In the provided source code zip file, there is a bat file, setup.bat. This file contains all the environment setup required to compile and run an operator from your current command prompt. There are a few areas inside this file that need to be changed based on your environment.

1. The `APT_OPERATOR_REGISTRY_PATH` needs to be the base directory where all your files are for your OSH operator.

```
SET APT_OPERATOR_REGISTRY_PATH=E:\osh\
```

2. `APT_ORCHHOME` is the location of the PXEngine. This is found under your Information Server install directory.

```
SET APT_ORCHHOME=E:\IBM\InformationServer\Server\PXEngine
```

3. `APT_CONFIG_FILE` is the location of a configuration file for the PXEngine, this is also located under your Information Server install

directory.

```
SET
APT_CONFIG_FILE=E:\IBM\InformationServer\Server\Configurations\default.apt
```

4. The Windows PATH variable needs to be updated to include the bin directory of the PXEngine and your current directory.

```
SET PATH=%PATH%;E:\IBM\InformationServer\Server\PXEngine\bin;.;
```

5. The Windows INCLUDE variable needs to be updated to include the PXEngine, Visual Studio .Net 2003, and the MKS Toolkit.

```
set INCLUDE=E:\Program Files\Microsoft Visual Studio
.NET 2003\VC7\PlatformSDK\include;
E:\IBM\InformationServer\Server\PXEngine\include;
C:\Program Files\MKS Toolkit\include;
E:\Program Files\Microsoft Visual Studio .NET 2003\VC7\ATLMFC\INCLUDE;
E:\Program Files\Microsoft Visual Studio .NET 2003\VC7\INCLUDE;
E:\Program Files\Microsoft Visual Studio
.NET 2003\VC7\PlatformSDK\include\prerelease;
E:\Program Files\Microsoft.NET\SDK\v1.1\include;
```

6. The Windows LIB variable needs the PXEngine's lib directory and the MKS Toolkit.

```
set LIB=%LIB%;E:\IBM\InformationServer\Server\PXEngine\
lib;c:\Program Files\MKS Toolkit\lib;
```

After making these changes, start a command prompt, navigate to your directory then run **setup.bat**. Your output should look like the following:

```
E:\osh>setup.bat
Setting environment for using Microsoft Visual Studio .NET 2003 tools.
(If you have another version of Visual Studio or Visual C++ installed and wish
to use its tools from the command line, run vcvars32.bat for that version.)
E:\osh>
```

To test to see if everything was setup correctly, type **osh** into the command window. The output should look similar to the following:

```
E:\osh>osh
##I IIS-DSEE-TFCN-00001 20:07:33(000) <main_program>
IBM WebSphere DataStage Enterprise Edition 8.0.0
Copyright IBM Corp. 2001, 2005
```

```

##I IIS-DSEE-TOSH-00002 20:07:33(001) <main_program> orchgeneral: loaded
##I IIS-DSEE-TOSH-00002 20:07:33(002) <main_program> orchsort: loaded
##I IIS-DSEE-TOSH-00002 20:07:33(003) <main_program> orchstats: loaded
##E IIS-DSEE-TCOS-00005 20:07:33(004) <main_program> Orchestrate Parallel Command Shell
usage: osh <options> "<step specification>"
      or: osh <options> -f file [-stdin file]
<options> include: [-echo] [-explain] [-dump] [-schema] [-check]
                  [-collation_sequence <collation file>]
                  [-collation_strength <collation strength>]
                  [-pdd <performance data directory>]
                  [-pf <parameter file>]
                  [-nopartinsertion] [-nosortinsertion]
                  [-params <parameter string>]
                  [-errorconfig <configstring>] [-jobid <jobid>]
                  [-nomonitor]
                  [[-monitorport <monitor port>] | [-monitorfile <file>]]
                  [-monitortype <'summary'>]
                  [-input_charset <map name>] [-output_charset <map name>]
                  [-string_charset <map name>] [-impexp_charset <map name>]
                  [-os_charset <map name>]
                  [-escaped]
                  [-default_date_format <uformat>]
                  [-default_time_format <uformat>]
                  [-default_timestamp_format <uformat>]
                  [-default_decimal_separator <separator>]
                  [-string_fields]
      or: osh -usage operatorname
      or: osh -fullusage operatorname
.
E:\osh>

```

Compile your operator

Compiling your operator is a straight forward process once the environment has been setup. To compile the operator, make sure that you have run setup.bat in the current command window, then type the following commands.

```

E:\osh>cl %APT_COMPILEOPT% src/myhelloworld.c
Microsoft (R) 32-bit C/C++ Standard Compiler Version 13.10.3077 for 80x86
Copyright (C) Microsoft Corporation 1984-2002. All rights reserved.

```

```
myhelloworld.c
```

```
E:\osh>
```

```

E:\osh>link %APT_LINKOPT% myhelloworld.obj liborchcorent.lib liborchnt.lib Kernel32.lib
Microsoft (R) Incremental Linker Version 7.10.3077
Copyright (C) Microsoft Corporation. All rights reserved.

```

```
E:\osh>
```

This leaves you with a compiled DataStage operator named myhelloworld.dll in your base directory.

Section 4. Run your first operator

OSH scripts

An OSH script represents a DataStage job. Instead of displaying it in a graphical window like DataStage Designer, it is just a text representation of the operators, their parameters, and the links between operators.

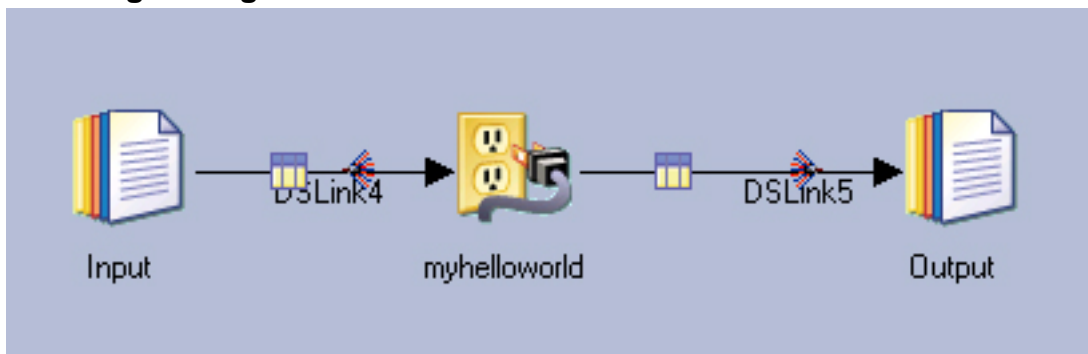
Inside the OSH script, there is a simple format to describe the basic structure of an operator. The first line is the name of the operator. The following lines start with `-`. These are the parameters for the operator. The streams between operators have a prefix of `<` or `>`, based on the direction of the stream. Input streams start with `<` and output streams start with `>`, then all the streams are suffixed with `.v`. Finally, a `;` is added to signify the end of the description for this operator. Additional operators are appended after the semicolon.

```
operatorname
-parameter1
-parameter2 'hello world'
...
-parametern

< 'inputstream.v'
> 'outputstream.v'
;
```

For your myhelloworld operator, you have a very simple OSH flow: one input, one myhelloworld operator, and one output.

Figure 1. Flow using sequential files as the input and output as shown the in DataStage Designer



The input operator for your example is a file reading operator called import. The

import operator has one parameter that needs to be changed based on your environment.

Note: This `-file` parameter must point to the text file located in the input directory.

```
## Operator Name
import

## Operator options
-schema record
  {final_delim=end, delim=',', quote=double}
  (
    inCount:int32;
  )
-file 'e:\osh\input\mhw.txt'
-rejects continue
-reportProgress yes

## Outputs
> 'inputFile.v'
;
```

The input file required by this operator is a text file using double quotes to surround strings, commas to separate columns, and each row is one line. As you only require one integer column, the file looks like the following:

```
1
2
3
4
```

Your operator myhelloworld OSH representation is relatively simple, it has one parameter `uppercase`, which is set to true. To set `uppercase` to false, remove the `-uppercase` parameter from the OSH script.

```
## Operator Name
myhelloworld

## Operator options
-uppercase

##Inputs
< 'inputFile.v'

##Outputs
> 'outputFile.v'
;
```

The output operator for your example is a file writing operator called "export." The export operator also has one parameter that needs to be changed, the file parameter needs to point to a file in the output directory. This file is overwritten every time this script is called, it is also created if the file does not exist at runtime.

```
## Operator Name
export
```

```

## Operator options
-schema record
  {final_delim=end, delim=',', quote=double}
  (
    outCount:int32;
    outHello:string;
  )
-file 'E:\\osh\\output\\mhw.txt'
-overwrite
-rejects continue

## Inputs
< 'outputFile.v'
;

```

Given this OSH script, using the input file specified above, there are three columns to look at: the input `inCount` and the two outputs: `outCount` and `outHello`. The expected column's values are shown below:

<code>inCount</code>	<code>outCount</code>	<code>outHello</code>
1	1	HELLO WORLD!
2	2	HELLO WORLD!HELLO WORLD!
3	3	HELLO WORLD!HELLO WORLD!HELLO WORLD!
4	4	HELLO WORLD!HELLO WORLD!HELLO WORLD!HELLO WORLD!

Operator mapping

The `operator.apt` file tells the PXEngine the mappings between operator names in a OSH script and the dll or executable located in your Windows PATH variable. Below is an example `operator.apt` for your operator.

```
myhelloworld myhelloworld 1
```

The first `myhelloworld` is the operator name that is defined inside the source code and is used inside an OSH script. The second `myhelloworld` is the name the PXEngine is looking for in its PATH search. To find the actual operator, it looks for executables first (.exe), then looks at dll files, so make sure you don't have a `myhelloworld.exe` sitting somewhere in the directories specified by your PATH variable. The 1 in the third column indicates that this mapping is enabled, if this is set to 0 the PXEngine ignores this mapping.

Run the operator

To run the operator, you give your OSH script you just created to the PXEngine. You do this by calling `osh.exe -f myhelloworld.osh`

```
E:\osh>osh -f myhelloworld.osh
##I IIS-DSEE-TFCN-00001 13:27:10(000) <main_program>
IBM WebSphere DataStage Enterprise Edition 8.0.0
Copyright IBM Corp. 2001, 2005

##I IIS-DSEE-TOSH-00002 13:27:10(001) <main_program> orchgeneral: loaded
##I IIS-DSEE-TOSH-00002 13:27:10(002) <main_program> orchsort: loaded
##I IIS-DSEE-TOSH-00002 13:27:10(003) <main_program> orchstats: loaded
##I IIS-DSEE-TFSC-00001 13:27:11(004) <main_program> APT configuration file:
E:/IBM/InformationServer/Server/Coons/default.apt
##I IIS-DSEE-TOIX-00163 13:27:12(000) <import,0> Import complete; 4 records
imported successfully, 0 rejected.
##I IIS-DSEE-TOIX-00059 13:27:12(002) <APT_RealFileExportOperator in Sequential_File_2,0>
Export complete; 4 reported successfully, 0 rejected.
##I IIS-DSEE-TFSC-00010 13:27:12(004) <main_program> Step execution finished with
status = OK.
##I IIS-DSEE-TCOS-00026 13:27:12(005) <main_program> Startup time, 0:00; production
run time, 0:00.

E:\osh>
```

After running the operator, you are left with a file in the output directory `e:\osh\output\mhw.txt`. Inside this file, you see a list of comma-separated values. The first column is `outCount` and the second column is `outHello`. The contents of this file should look like the output below.

```
E:\osh>cat output/mhw.txt
"1", "HELLO WORLD!"
"2", "HELLO WORLD!HELLO WORLD!"
"3", "HELLO WORLD!HELLO WORLD!HELLO WORLD!"
"4", "HELLO WORLD!HELLO WORLD!HELLO WORLD!HELLO WORLD!"

E:\osh
```

Section 5. Load the operator into DataStage

Set up the environment

As DataStage does not have access to the environment you set up for development, you have to make a few changes so that DataStage can find your operator. First, your operator's dll file is not inside Windows' or DataStage's PATH environment variable, the easiest way to put your operator into the PATH is to move the dll into the PXEngine's bin directory. Alternatively, you can update your windows PATH variable to include your project's directory.

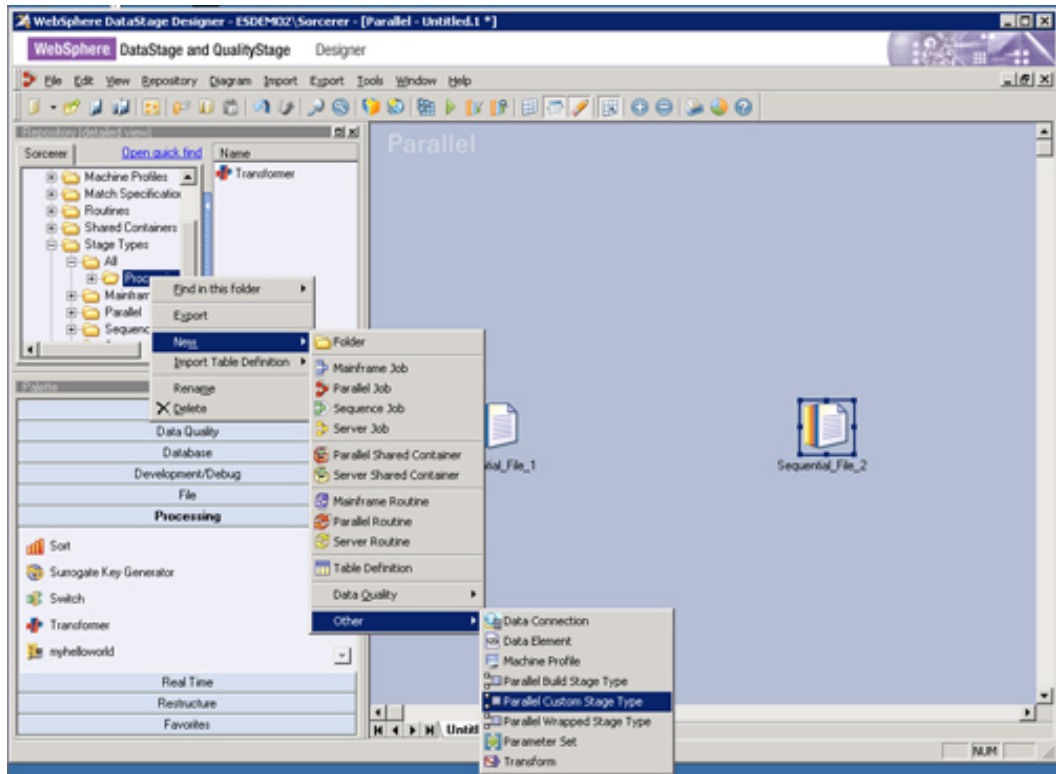
However, the operator is still not found by DataStage, as there is no mapping from the dll to the OSH name. To fix this, open up the PXEngine's main operator.apt file, located in e:\IBM\InformationServer\Server\PXEngine\etc\, then add **myhelloworld myhelloworld 1** to this file.

Add the operator

Before starting this section, you need to know some of the following information about your DataStage environment:

- DataStage username
 - DataStage password
 - DataStage server name
 - DataStage server port
 - DataStage project
1. Start DataStage Designer and log into a project that you are able to use.
 2. Once you are inside, right-click any folder in the repository view.
 3. Select **New > Other > Parallel Custom Stage Type**.

Figure 2. Path to setting the stage type

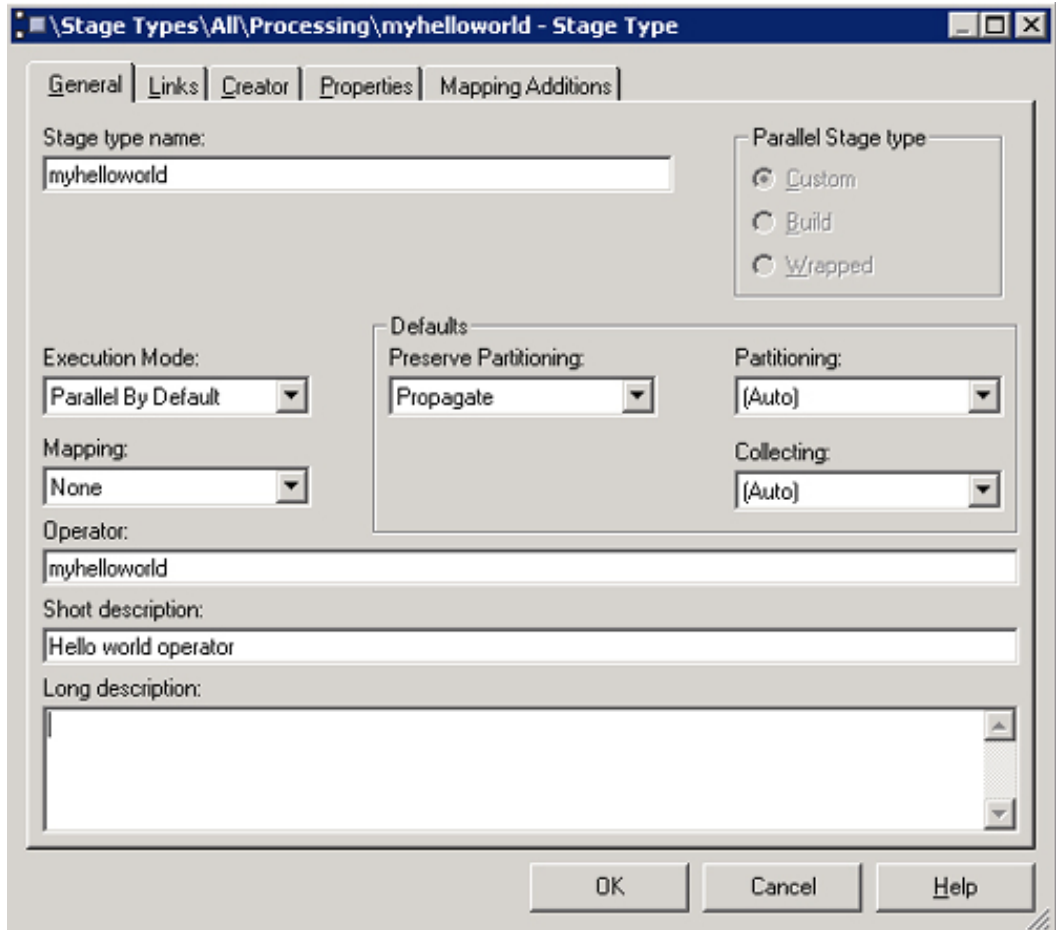


4. Fill in the tabs as show in the following figures. You do not need to modify the mapping tab.

Fill in the following under the General tab:

- a. **Stage type name** -- With the name of your operator (myhelloworld)
- b. **Operator** -- With the name of your operator (myhelloworld)
- c. **Short description** -- A short description of what the operator does (hello world operator)

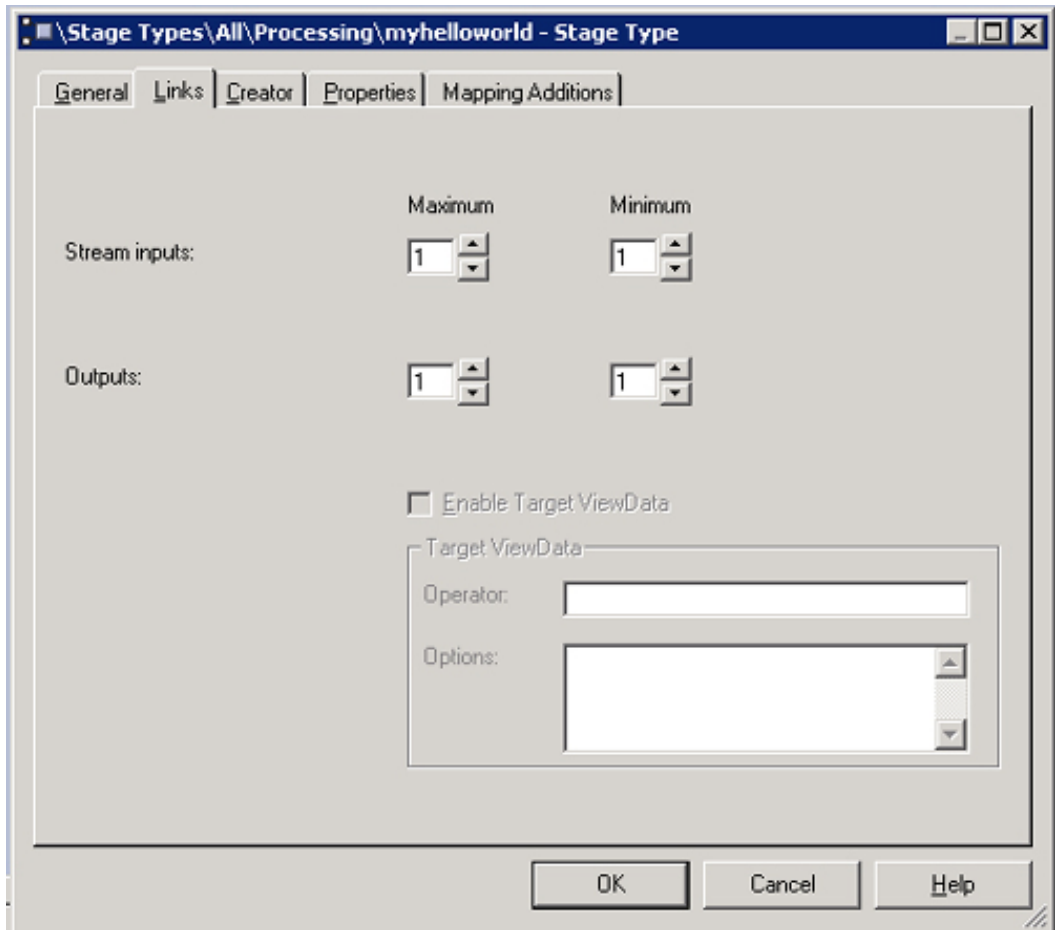
Figure 3. General tab



Fill in the following under the Links tab:

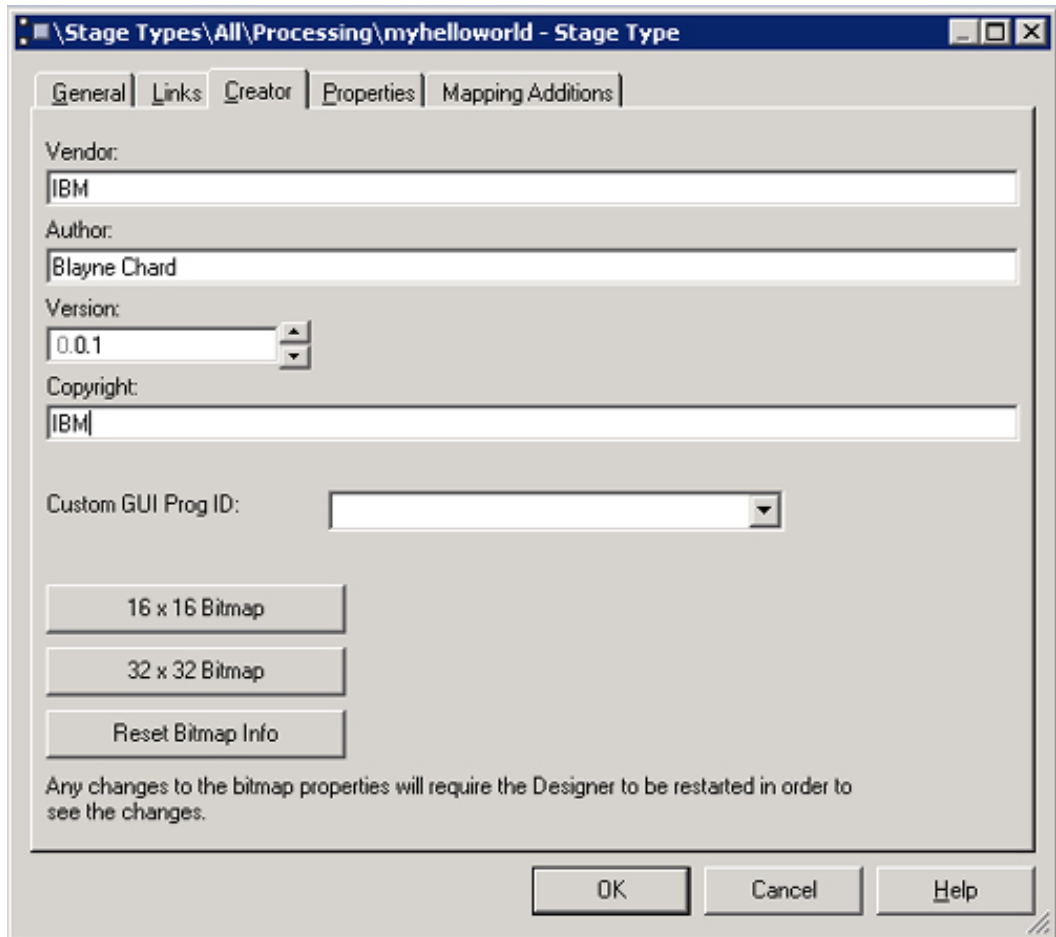
- a. **Stream inputs Maximum** -- With the maximum number of input links (1)
- b. **Stream inputs Minimum** -- With the minimum number of input links (1)
- c. **Outputs Maximum** -- With the maximum number of output links (1)
- d. **Outputs Minimum** -- With the minimum number of output links (1)

Figure 4. Links tab



Fill in your information in the appropriate text boxes under the Creator tab.

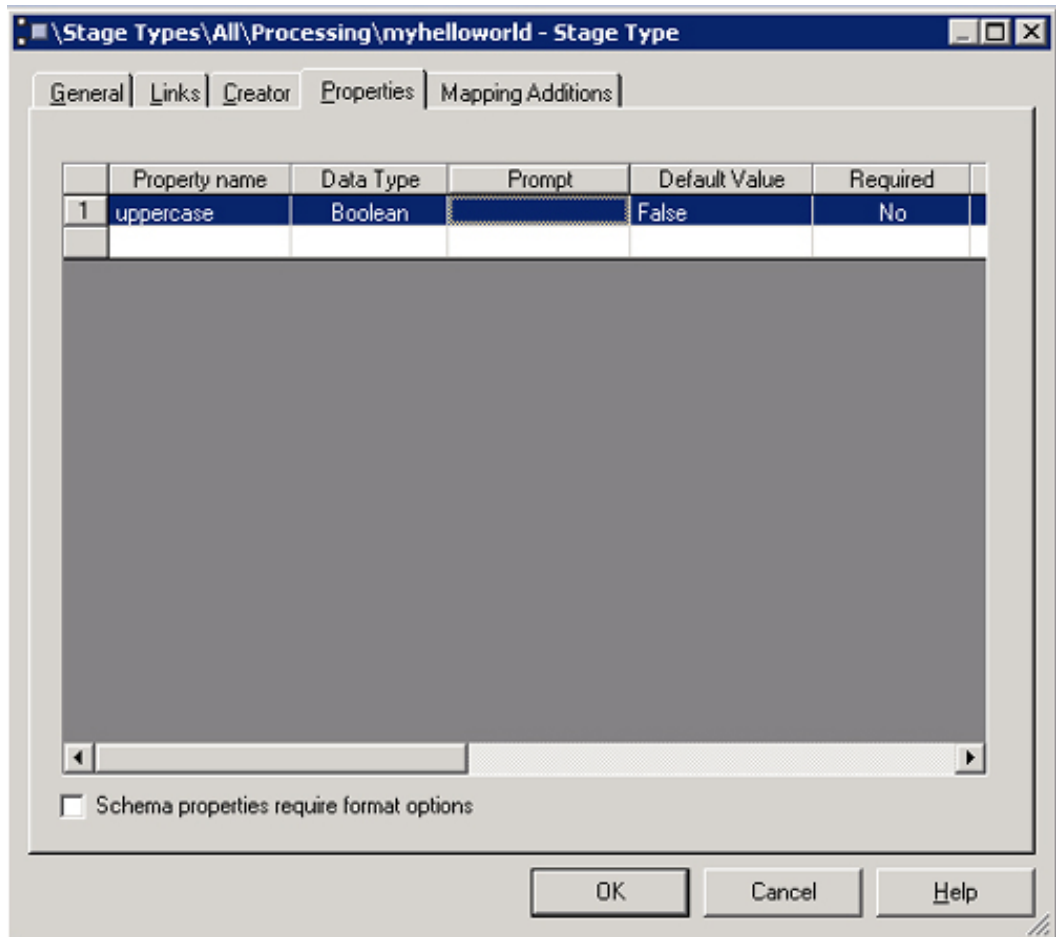
Figure 5. Creator tab



Add a new property under the Properties tab.

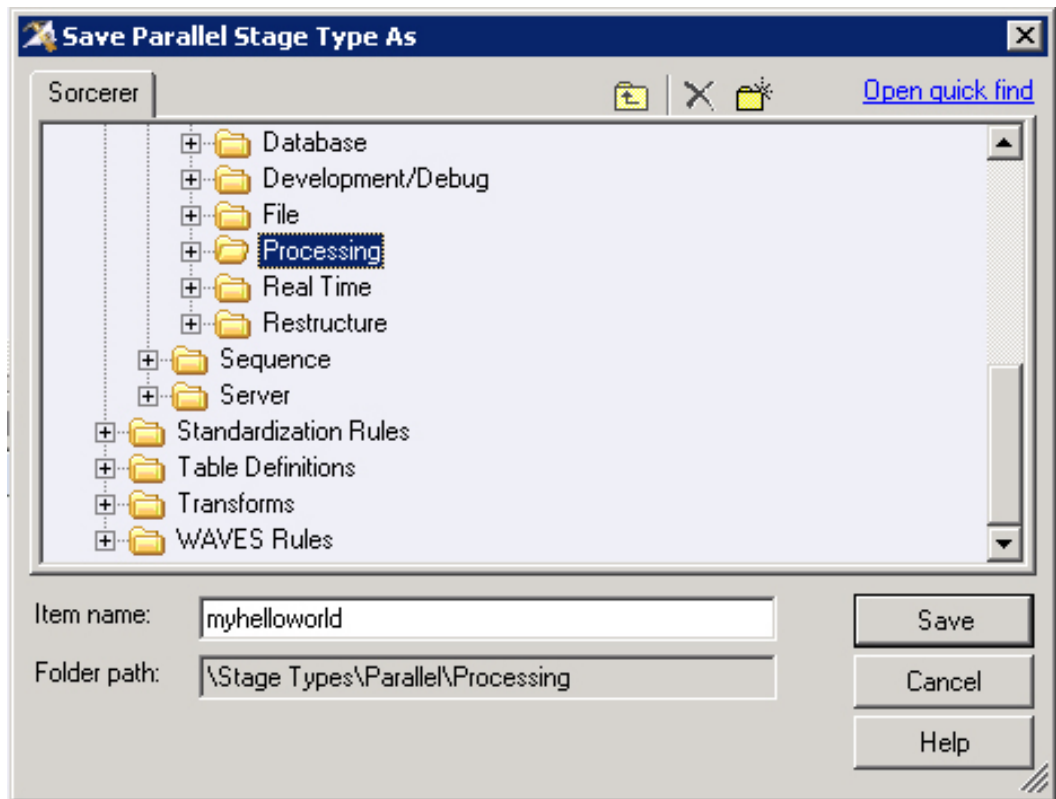
- a. **Property name** -- The name of the parameter as defined in the operator(`uppercase`)
- b. **Data Type** -- The data type (`boolean`)
- c. **Default Value** -- Default value to use (`False`)
- d. **Required** -- If the property is required (`No`)

Figure 6. Properties tab



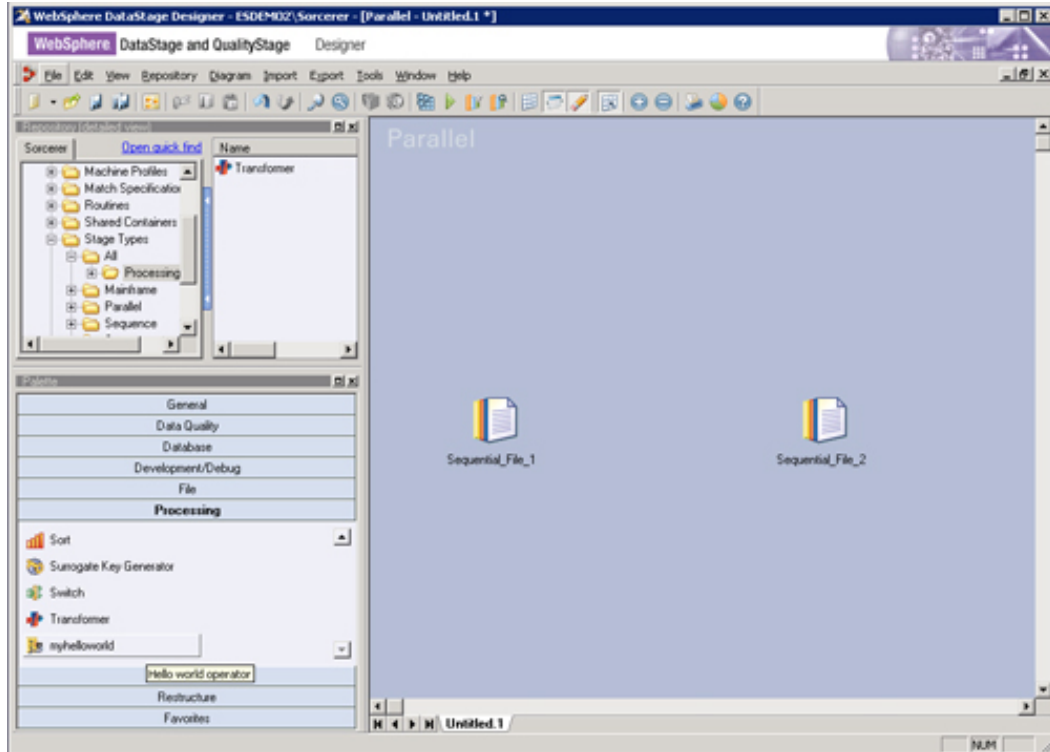
5. Click **Ok**. This brings up the Save as dialog box. Select an appropriate place to save, (the example uses the processing operator type folder) then click **Save**.

Figure 7. Save as



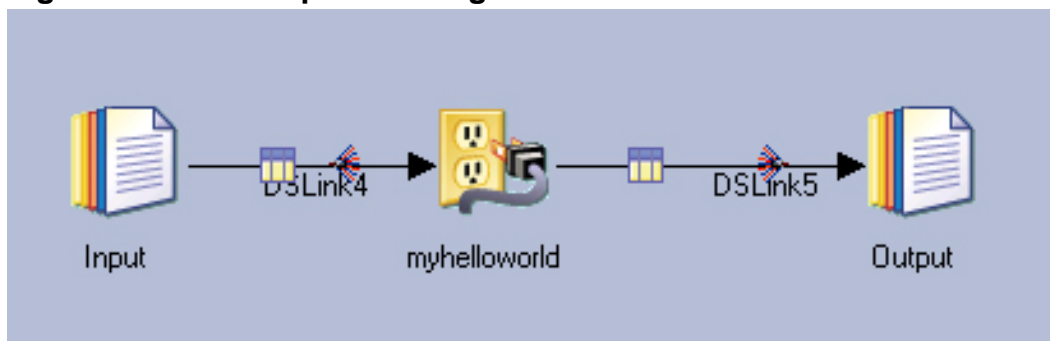
6. You should now see a myhelloworld operator inside the Processing tab of the palette.

Figure 8. Processing tab



7. To check that everything has completed successfully, build a simple parallel job to test the new custom operator. Create a job by dragging two sequential file operators and the myhelloworld operator onto the canvas, then link them together as shown in Figure 9.

Figure 9. Link the operators together



8. Open the two sequential file operators and set the details of the files they are reading from or writing to.

Figure 10. Add the location of the file to read from

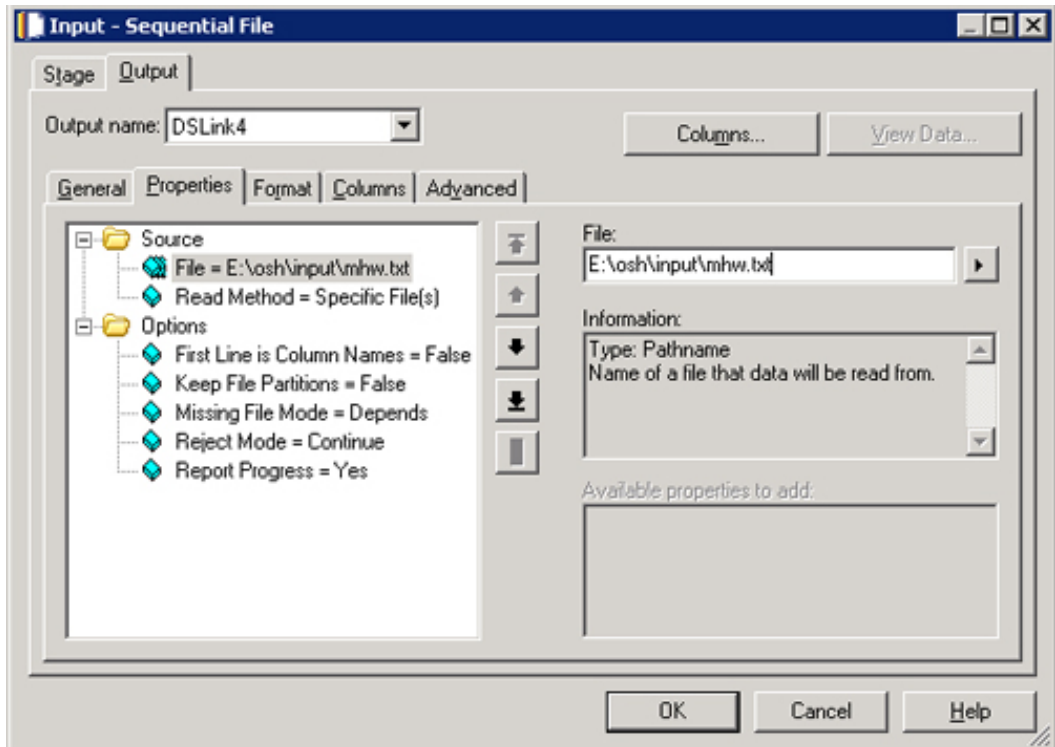
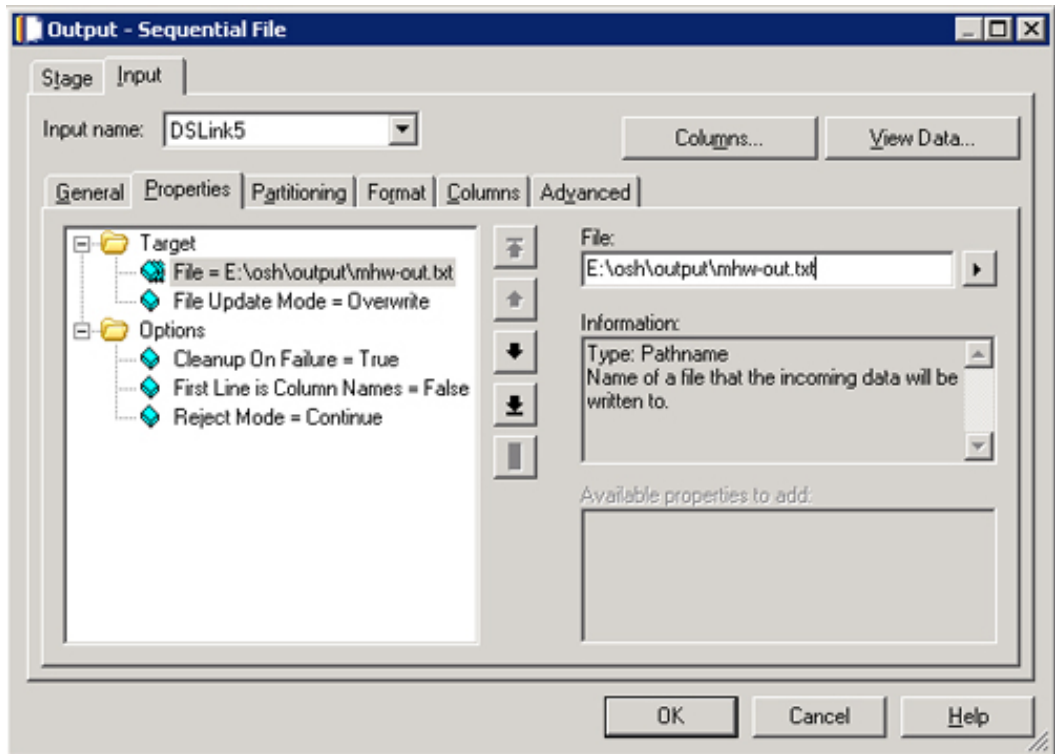


Figure 11. Add the location of the file to write to



9. Open the myhelloworld operator and set the uppercase parameter to

True.

Figure 12. Add "uppercase" to the list of parameters

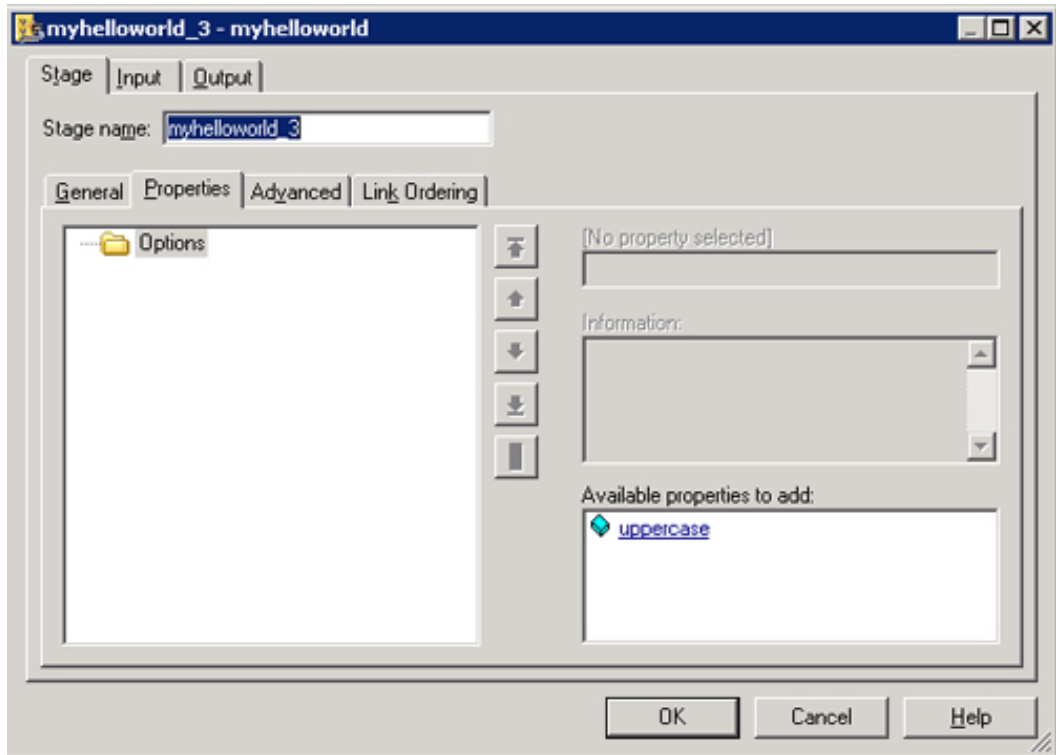
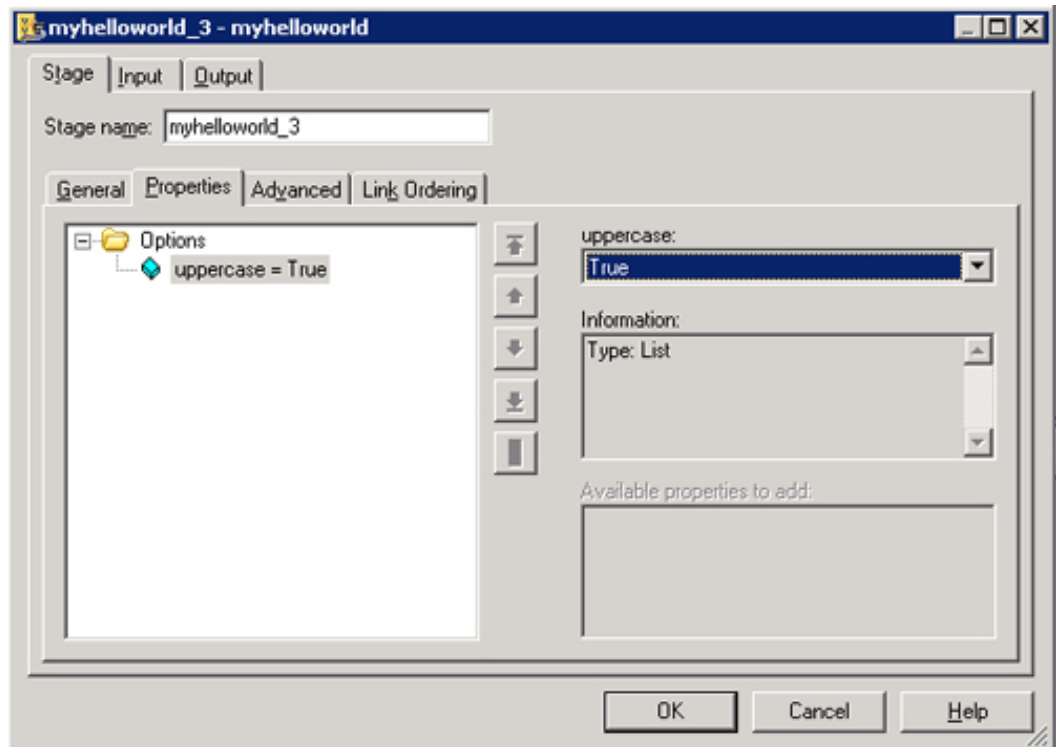
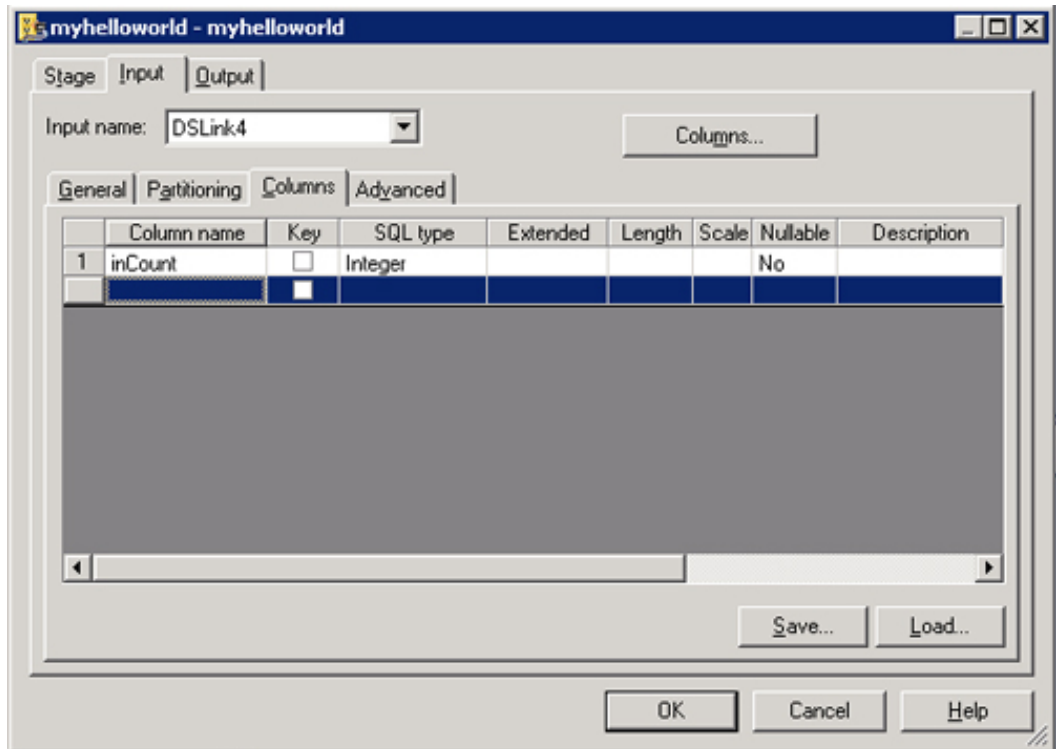


Figure 13. Set uppercase to be true



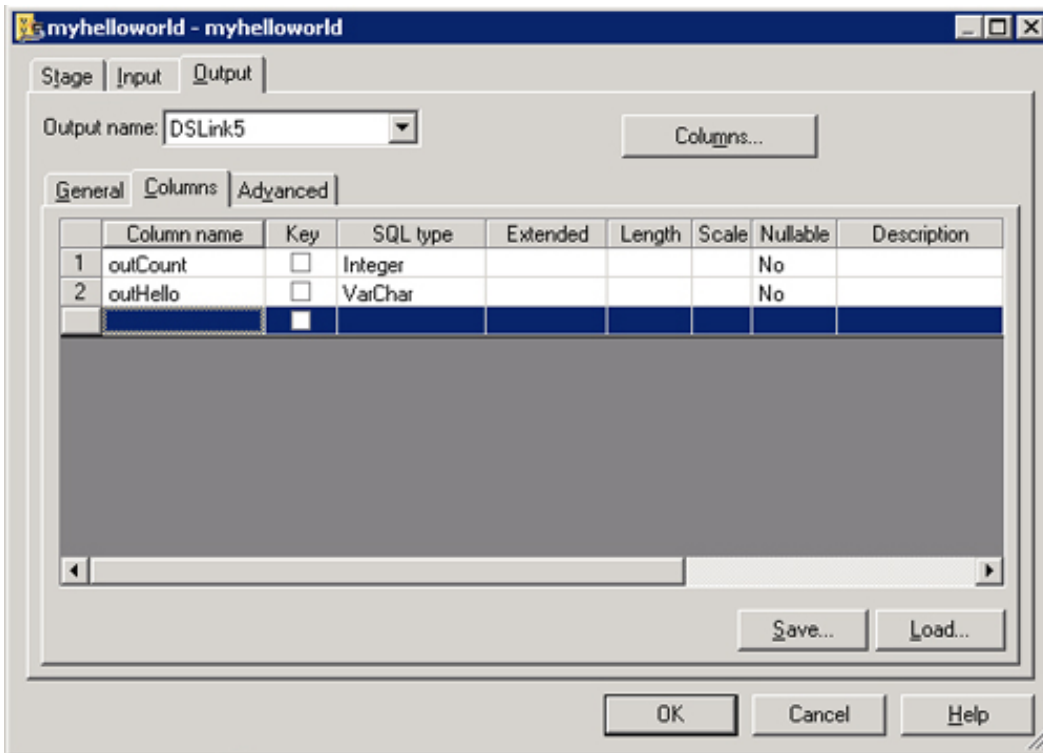
10. Inside the myhelloworld operator, go to **Input > Columns**, and add the input column. Add a new column with the column name **inCount** and the SQL type **Integer**.

Figure 14. Add input column information



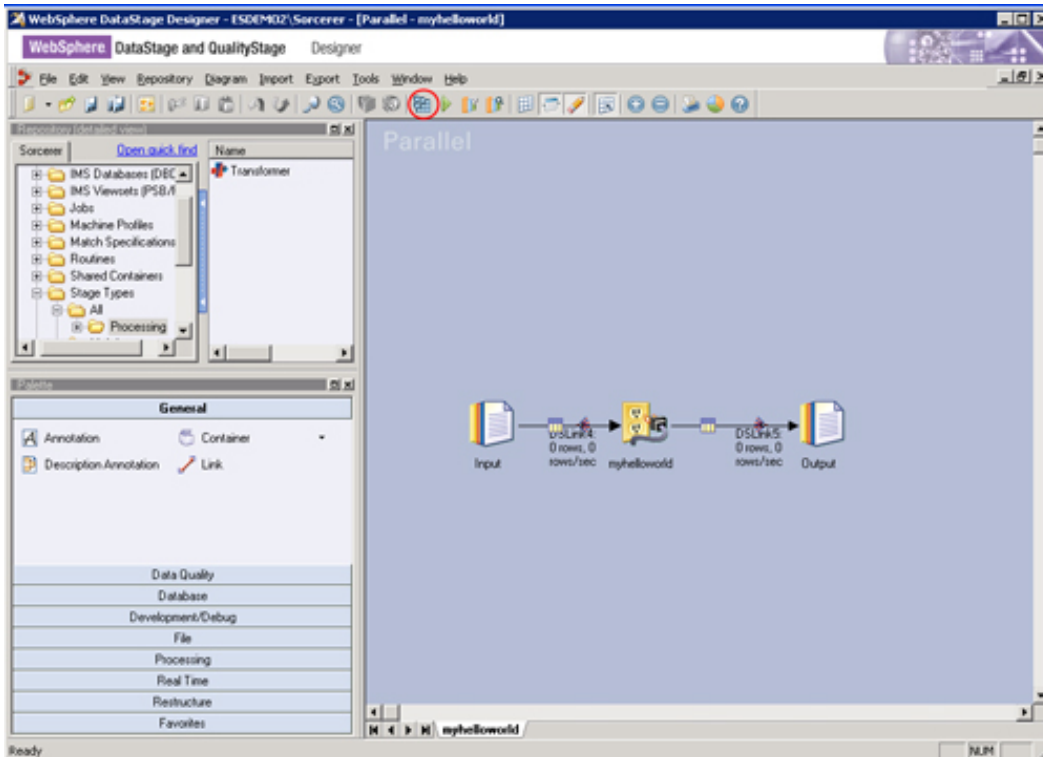
11. Go to **Output > Columns**, and add the output columns. Add two new columns one with the column name **outCount** and the SQL type **Integer**, and another with the column name **outHello** and the SQL type **Varchar**.

Figure 15. Add output column information



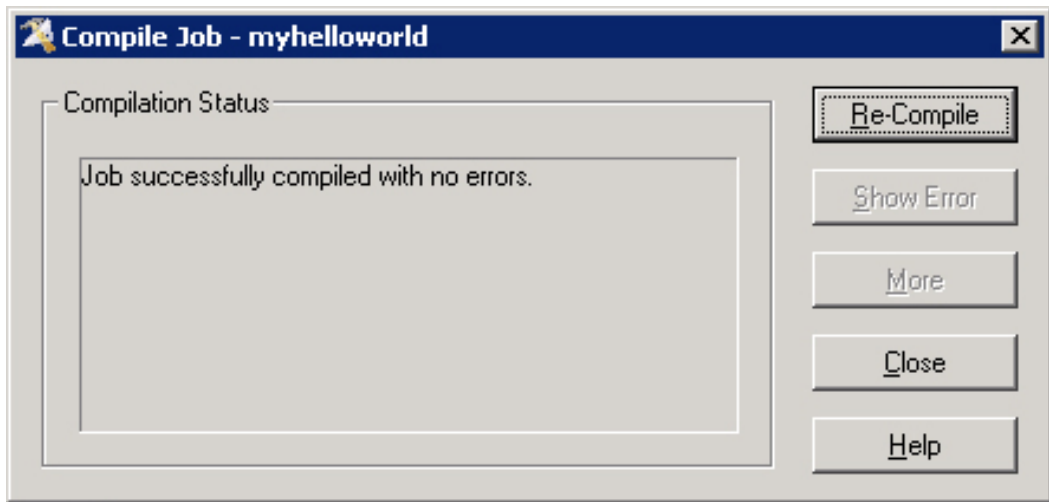
12. Save your job, then click the **Compile** icon.

Figure 16. Click the Compile icon



This should result in the compile success dialog box.

Figure 17. Compile successful dialog box



- 13. Click the **Run** icon, which spawns the run dialog box, where you click **Run**.

Figure 18. Click the Run icon

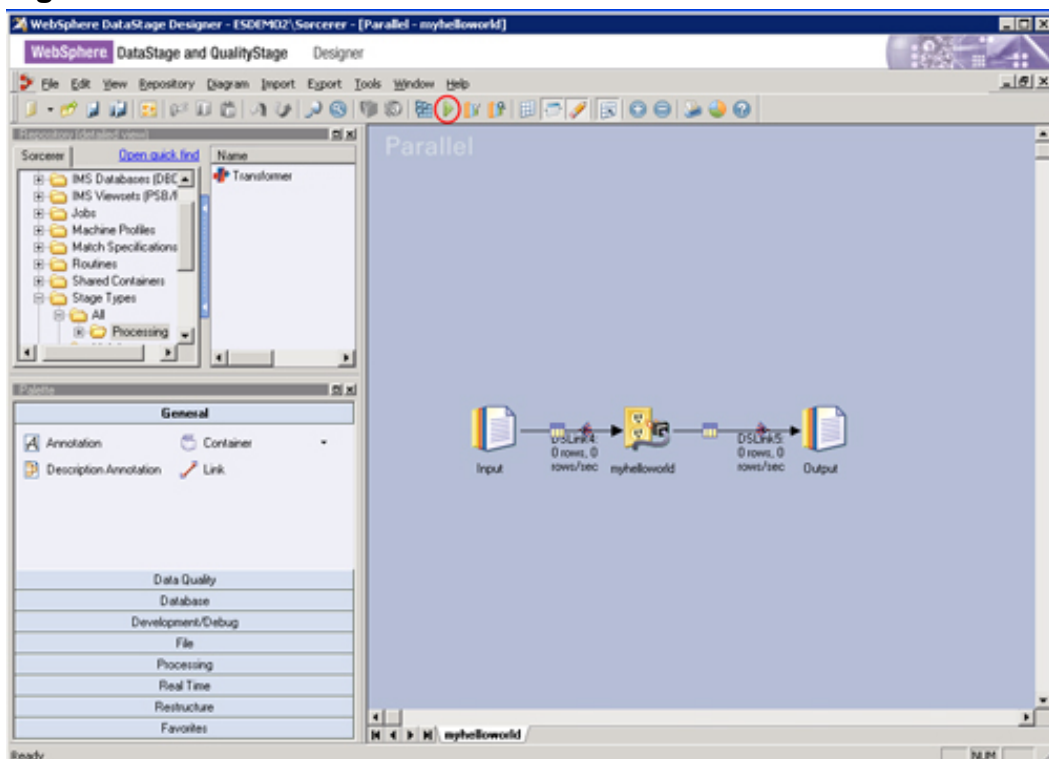
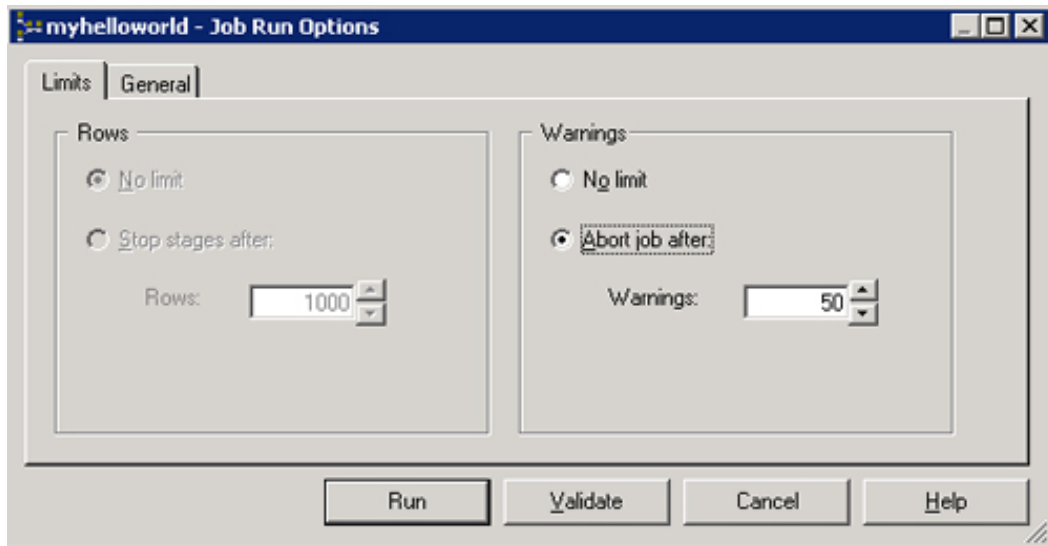
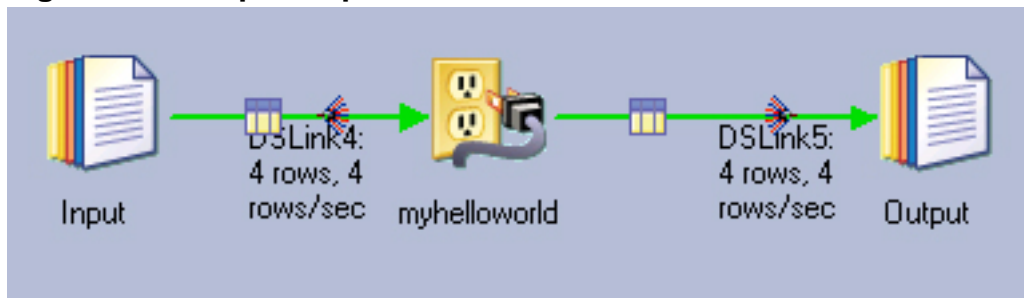


Figure 19. Click Run



14. If all goes well, in a couple of seconds the links should turn green in your canvas and display the number of records they have processed.

Figure 20. Completed process



Congratulations! You now have a working MyHelloWorld operator inside DataStage.

Downloads

Description	Name	Size	Download method
Code for all the examples	code.zip	4KB	HTTP

[Information about download methods](#)

Resources

Learn

- [WebSphere DataStage zone](#): Get more details, and access resources and support for DataStage.
- [Information Integration zone](#): Read articles and tutorials and access documentation, support resources, and more, for the IBM Information Integration suite of products.
- [developerWorks Information Management zone](#): Learn more about DB2. Find technical documentation, how-to articles, education, downloads, product information, and more.
- Stay current with [developerWorks technical events and webcasts](#).

Get products and technologies

- Build your next development project with [IBM trial software](#), available for download directly from developerWorks.
- Learn more about the [MKS Toolkit](#).

Discuss

- [Participate in the discussion forum for this content](#).
- Participate in [developerWorks blogs](#) and get involved in the developerWorks community.

About the author

Blayne Chard

Blayne Chard is a intern at the IBM Silicon Valley Lab in San Jose, Calif. He received is bachelor's degree with honors in Computer Science from Victoria University of Wellington, New Zealand. Blayne currently works for the WebSphere Information Server team.