Chapter 17.

May 2008

Welcome to the May 2008 edition of IBM Information Server Developer’s Notebook. This month we answer the question;

> How do I read and load possibly corrupt input data files into my IBM Information Server DataStage component? And, How do I do this in a repeatable/reusable manner?
>
> *Excellent question! We see this often in electronic data interchange (EDI) installations where maybe the source system is on older technologies; data files get truncated, or non-visible control characters get inserted into the file.*
>
> Basically our strategy in this case is to accept the input data file in the least restrictive format allowed, so we may then perform analysis on the input data stream using the drag and drop capabilities of DataStage.

In short, we are going discuss all of the relevant terms and technologies above, and provide examples that detail how to deliver this functionality using IBM Information Server.

Software versions

All of these solutions were *developed and tested* on IBM Information Server (IIS) version 8.01, FixPak 1A, using the Microsoft Windows XP/SP2 platform to support IIS client programs, and a RedHat Linux Advanced Server 4 (RHEL 4) FixPak U6 32 bit SMP server (Linux kernel version 2.6.9-67.EL-smp) to support the IIS server components.

IBM Information Server allows for a single, consistent, and accurate view of data across the full width of the corporate enterprise, be it relational or non-relational, staged or live data. As a reminder, the IBM Information Server product contains the following major components;

> WebSphere Business Glossary Anywhere™, WebSphere Information Analyzer™, WebSphere Information Services Director™, WebSphere DataStage™, WebSphere QualityStage™, WebSphere Metadata Server and Metabridges™, WebSphere Metadata Workbench™, WebSphere Federation Server™, Classic Federation™, Event Publisher™, Replication Server™, Rational Data Architect™, DataMirror Transformation Server™, and others.

Obviously, IBM Information Server is a large and capable product, addressing many strategic needs across the enterprise, and supporting different roles and responsibilities.

17.1 Terms and core concepts

Illegal, or non-visible characters, and a design pattern

With the numerous operating system platforms, and relational and non-relational data stores that IBM Information Server (IIS) supports, there really could not be one built in Operator that could filter out or repair 'illegal' or non-visible characters. Who is to say what is an illegal character in every case? What is an illegal character on one platform, or for one application, may be a perfectly valid character for another platform or application. Also, IIS can easily process binary data, which is non-visible (non-printable) by its definition.

However, detecting illegal or non-printable characters is a valid use case for IIS, and we demonstrate a solution to this need below. Further, after demonstrating a solution, we then demonstrate the more general case of creating a reusable service component within IIS.

Figure 17-1 displays the sample IIS DataStage component Job that forms the bulk of our focus in this document.

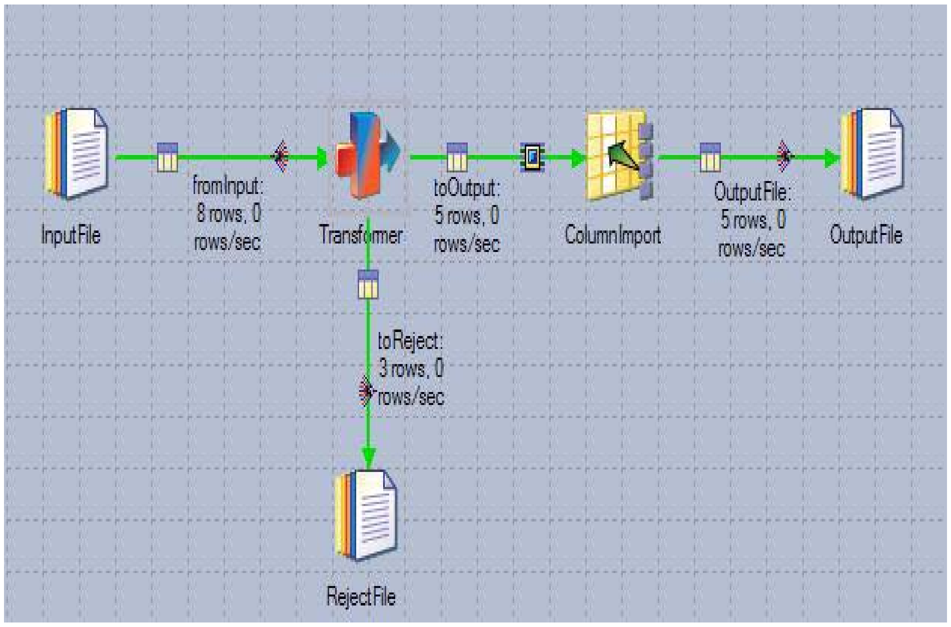

InputFile
fromInput: 8 rows, 0 rows/sec
Transformer
toOutput: 5 rows, 0 rows/sec
ColumnImport
OutputFile: 5 rows, 0 rows/sec
OutputFile
toReject: 3 rows, 0 rows/sec
RejectFile

Figure 17-1 Sample DataStage Job to detect and report illegal input characters.

A code review of the Job displayed in Figure 17-1 is offered below;

- InputFile represents an IIS DataStage component Sequential File operator. While any input type would have sufficed here, Sequential Files are the least restrictive; the input data source that most often allows illegal characters.

  Whereas the initial input operator for a DataStage Job is normally used to parse the input columns, assign data types and related, we wont do that here. In this job, we use one input column of the least restrictive data type (VarChar or similar). The design approach we are using is to allow ourselves to read every input piece of data, then using an algorithm under our exact and direct control, examine what we have received as input.

  Here the Sequential File operator reads everything, and then passes it to the next operator, a Transformer.

- Transformer is an IIS DataStage component Transformer operator, the workhorse of DataStage. This operator will be promoted to a reusable Job component later in this document, so the Links to this operator were named with that idea in mind; very generic, very obvious as to each Link's purpose in life.

  The basic algorithm of this operator is to substitute every illegal character (there will be a list of every character we wish to target), to a single type of character that we can easily count. After this (global) substitution, we count this known character. If we have more of this character after the substitution than we did before, we reject the input data row.

  The net of this design pattern is that the Transformer can be programmed to do anything; a Transformer could reject a row for any reason we choose, Transformers being fully programmable.

  Lastly, we call one of the output Links to this operator a Reject Link. While this Link does output bad rows, this Link is technically not a Link of type Reject; it is a Standard Output Link.

- ColumnImport is an IIS DataStage component Column Import operator; this operator is normally found in the Restructure drawer of the Palette view of the DataStage Designer program.

  Column Import is the operator that now parses our input data stream and produces named columns and data types. Column Import receives a pre-cleansed/pre-tested input data stream free of defects, as provided by the previous Transformer operator.

- The remaining operators within the Job displayed in Figure 17-1 serve as output devices only.

Code reusability and DataStage

There are a dozen or more technologies within the IBM Information Server (IIS) DataStage component to promote code reuse. The technology within DataStage to reuse single or sets of built in operators (like the Transformer used in Figure 17-1), is the Shared Container.

A Shared Container allows for reference to a subset of a previously created DataStage Job. In Figure 17-1, we created a Transformer that is pretty useful and reusable, so we will promote it to a Shared Container. This work is performed far below. For now, Figure 17-2 displays the Job from Figure 17-1, with the Shared Container in place.

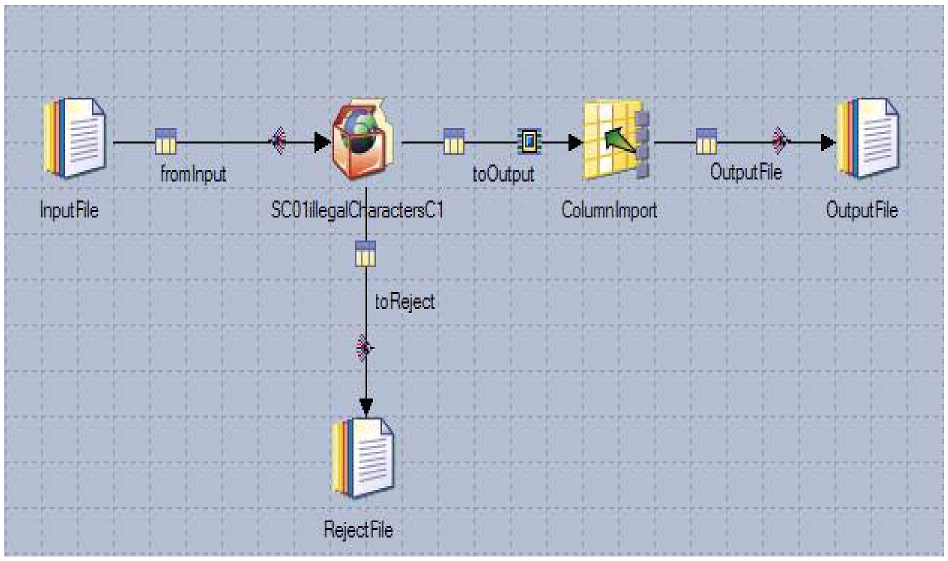


Figure 17-2   Same job as above, with reusable Job components in place.

Figure 17-3 displays the contents of the Shared Container displayed in Figure 17-2. A Shared Container is a reference to another piece of DataStage Job code; meaning, if any number of DataStage Jobs reference a given Shared Container, and the code in that Shared Container changes, all Jobs referencing that Shared Container benefit from the change.

When you import a Shared Container, you import a reference to that code. If you wish optionally, you can break that reference and make an independent, stand alone copy of the code that the Shared Container held.

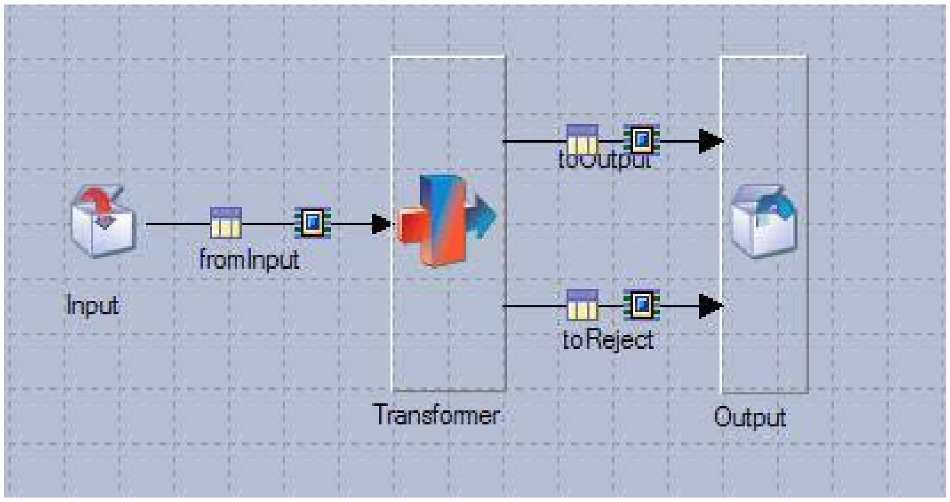


Figure 17-3 Contents of Shared Container first displayed above.

Figure 17-3 completes our discussion of Shared Containers, as per what is detailed in this document. Figure 17-4 and Figure 17-5 detail that one can even have Shared Container which itself makes reference to (other) Shared Containers.

Previously we promoted the Transformer we were using to a Shared container, because the function provided by this operator was very reusable. However, the Column Importer may also present functionality that could be reused. You wouldn't automatically bind the Transformer and that given Column Importer together, because the Transformer has use for other output record types.

Figure 17-4 displays a Shared Container that itself contains a Shared Container. We know Figure 17-4 displays a Shared Container because of its input and output operators, which are specific to Shared Containers.

.

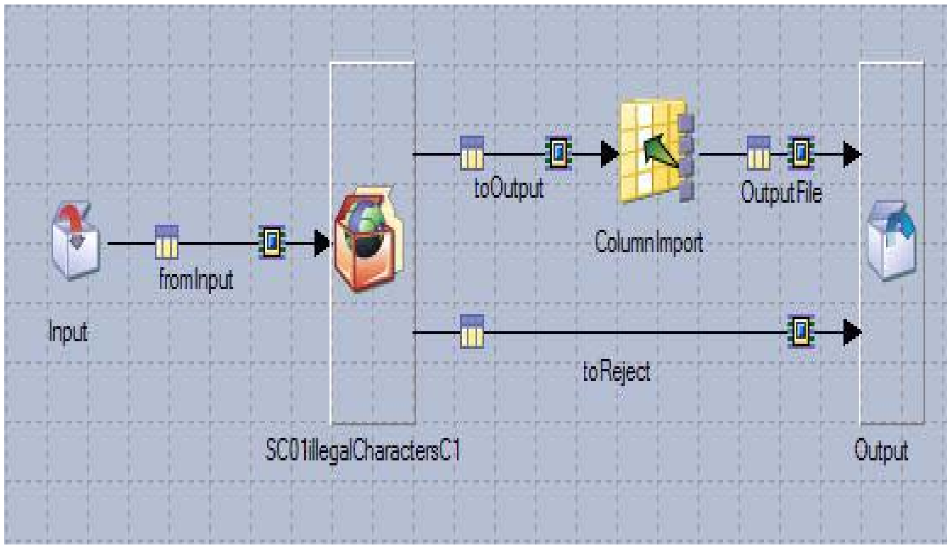


Figure 17-4   A Shared Container which itself makes use of another Shared Container.

And Figure 17-5 displays our original Job from Figure 17-1, with the nested Shared Containers in place.

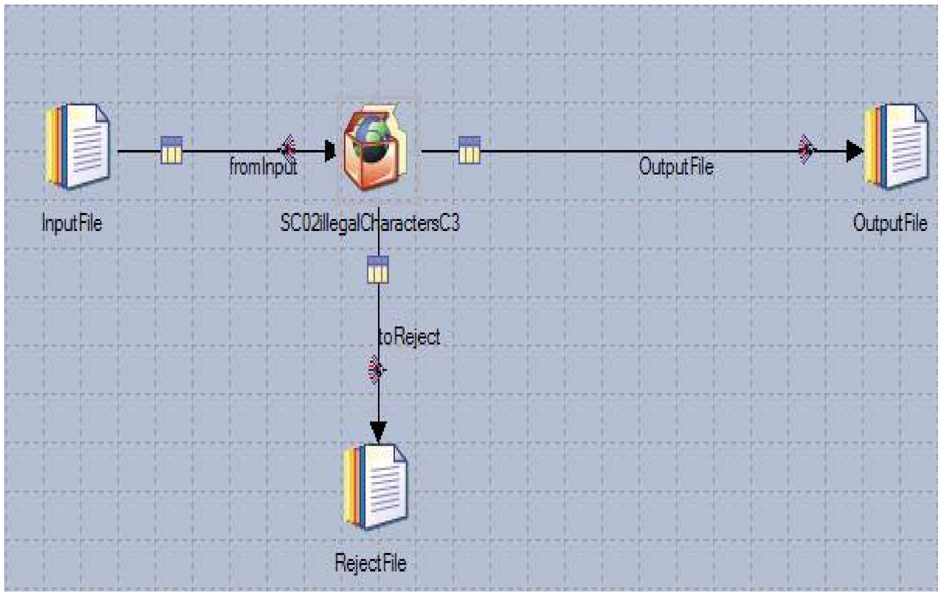


Figure 17-5   The original Job from above, with nested Shared Containers.

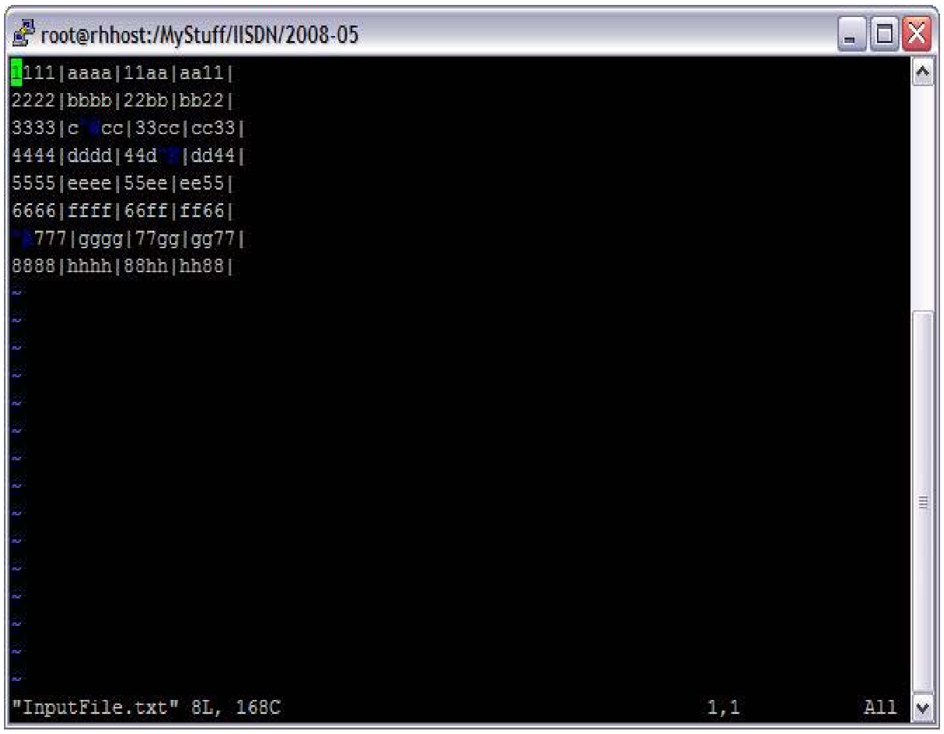
17.2 Creating the example(s) outlined above

In the section above, we discussed an IBM Information Server (IIS) DataStage component Job that demonstrates how to process illegal characters in an input data stream. First we will create that Job, then we will modify that Job to demonstrate Shared Containers and DataStage Job code reuse. These two topics can be treated independently, we just happen to use the first Job to demonstrate how to create and use Shared Containers.

To create the IIS DataStage Job example from Figure 17-1, perform the following;

1. Create an ASCII text file with a random collection of illegal or non-printable characters.

   Figure 17-6 displays a 4 column, 8 row sample input file. In 3 of the input data rows, we used our program editor to insert illegal (non-printable) characters. If you are using Vi(C), in Input Mode, press Control-V (for verbose), and then enter a control sequence; for example, Control-A.



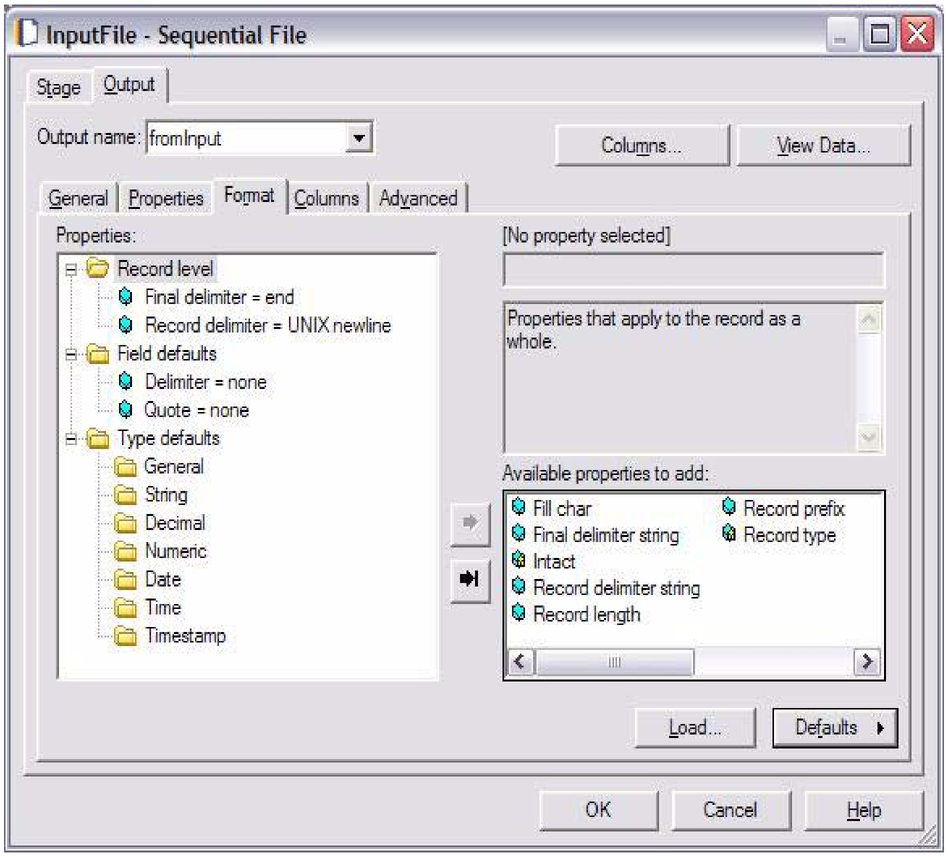
root@rhhost:/MyStuff/IISDN/2008-05

```
1111|aaaa|11aa|aa11|
2222|bbbb|22bb|bb22|
3333|c[illegible]cc|33cc|cc33|
4444|dddd|44d[illegible]|dd44|
5555|eeee|55ee|ee55|
6666|ffff|66ff|ff66|
[illegible]777|gggg|77gg|gg77|
8888|hhhh|88hh|hh88|
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"InputFile.txt" 8L, 168C                                 1,1           All
```

Figure 17-6 Sample input data file with illegal characters.

2. Launch the IBM Information Server (IIS) DataStage component Designer Program, create a new Parallel Job, and save this Job with a name of your choosing.
3. From the Palette view of the Designer Program, drag and drop the 5 operators displayed in Figure 17-1;
   - You need 3 Sequential File operators, a Transformer operator, and a Column Import operator.

     If you’ve never used Column Import, its located in the Restructure drawer of the Palette view. Transformer is under Processing.
   - Relocate (move) and rename all of the operators and Links as displayed in Figure 17-1.
4. For the InputFile Sequential File operator, complete the following;
   a. Set the Output TAB -> Properties TAB -> Source -> File (name) value to equal the sample input file you created above.
   b. Set the Output TAB -> Format TAB, various properties so that you may read the file you created above. From the image in Figure 17-7, we made settings that would read the single input line as one input column.



Output name: fromInput

Properties:

- Record level
  - Final delimiter = end
  - Record delimiter = UNIX newline
- Field defaults
  - Delimiter = none
  - Quote = none
- Type defaults
  - General
  - String
  - Decimal
  - Numeric
  - Date
  - Time
  - Timestamp

[No property selected]

Properties that apply to the record as a whole.

Available properties to add:

- Fill char
- Final delimiter string
- Intact
- Record delimiter string
- Record length
- Record prefix
- Record type

Figure 17-7 Input File, Output TAB -> Format TAB.

c. Set the Output TAB -> Columns TAB to contain only one column.

We named our single input column "Record", of type VarChar.

5. For the Transformer operator, complete the following;

a. Open the Transformer operator with a Double-Click, then open the Transformer Stage Properties dialog box.

Figure 17-8 displays the icon in the Toolbar you need to click to access this dialog box. (Look for the crudely drawn Red Arrow.)

b. In Figure 17-8, and under the Stage TAB -> Variables TAB, define 5 Transformer Stage Variables, example as shown;

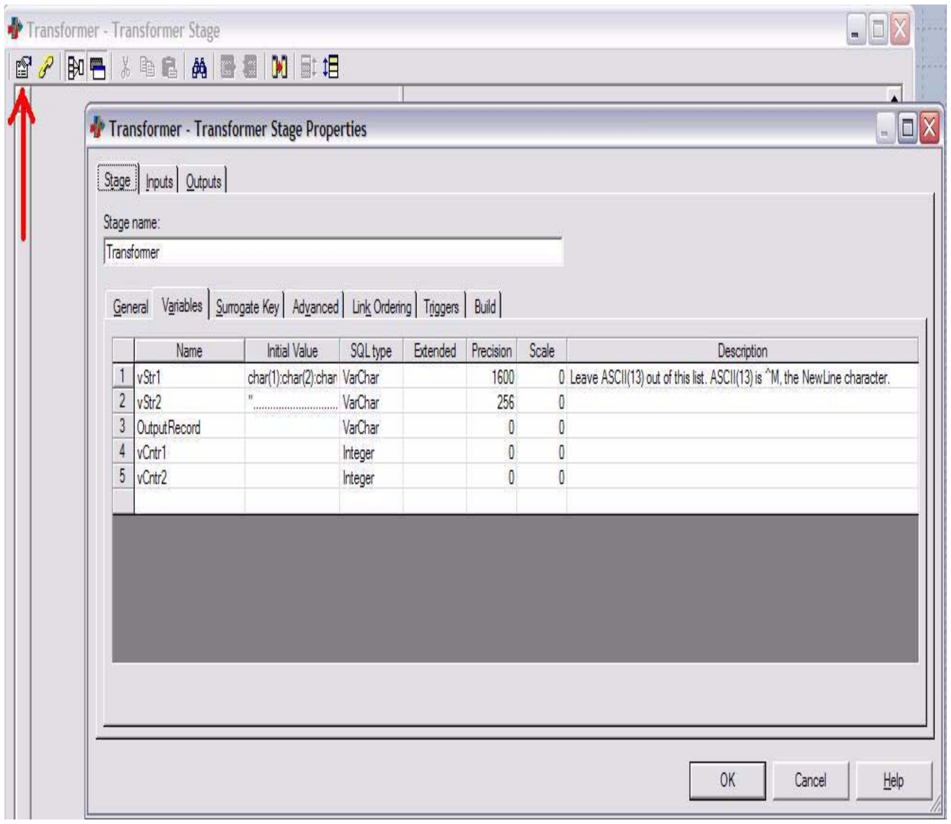
- vStr1 is defined to have an initial value equal to every ASCII character we wish to recognize as an illegal, non-printable character.

If we thought the letter "D" was an illegal character, we would put its ASCII value in this list. As it turns out, the ASCII characters in the ranges 1 through 27, and then 127 through 255 are the characters we wish to test for, and capture in this use case.

The full (Initial Value) for vStr1 is, "char(1):char(2):", and so on; ASCII 1 through 17, and then 127 through 255.

- vStr2 is defined to have an initial value of some constant character; we used the period symbol, but any single character works.

  vStr2 must contains as many single characters as are defined for vStr1; meaning, if vStr1 defines 22 characters, then vStr2 should have 22 periods, per our algorithm and design.



Stage name:
Transformer

	Name	Initial Value	SQL type	Extended	Precision	Scale	Description
1	vStr1	char(1):char(2):char	VarChar		1600	0	Leave ASCII(13) out of this list. ASCII(13) is ^M, the NewLine character.
2	vStr2	".............................	VarChar		256	0	
3	OutputRecord		VarChar		0	0	
4	vCntr1		Integer		0	0	
5	vCntr2		Integer		0	0	

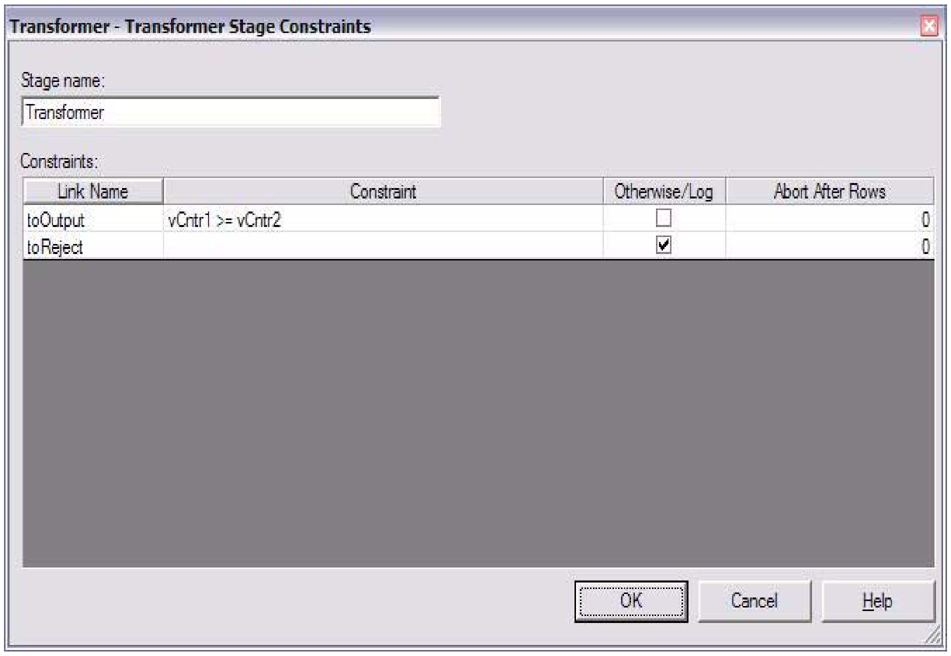
Figure 17-8 Transformer configuration, Screen 1 of 3.

c. Figure 17-9 displays the Links dialog for this Transformer operator; this dialog is opened from the second icon on the tool bar referenced in Step-5.a above.

- Enter a Constraint for the toOutput Link, as displayed in Figure 17-9.
- Check the toReject Link to enable it as an 'Otherwise" output.

vCntr1 and vCntr2 will be set equal to the number of periods in both the original input record (vCntr1, fromInput.Record), and then the substituted/massaged output record (vCntr2, toOutput.Record).

If we have more periods in the massaged record, then it had illegal/target characters, and we reject the row; this is the Otherwise condition and toReject Link. If not, the input record flows to our standard output Link, toOutput.



Link Name	Constraint	Otherwise/Log	Abort After Rows
toOutput	vCntr1 >= vCntr2	☐	0
toReject		☑	0

Figure 17-9 Transformer configuration, Screen 2 of 3.

d. Figure 17-10 displays the Transformer operator Properties dialog proper. Here we need to edit the Derivation Expressions as displayed;

- vStr1 and vStr2 do not need to be set; their values never change.
- OutputRecord's Derivation Expressions is set equal to,

  Convert(vStr1, vStr2, fromInput.Record)

Convert is a built in DataStage, Transformer operator, String manipulation function; in effect, convert every character found in vStr1 to its counterpart in vStr2.

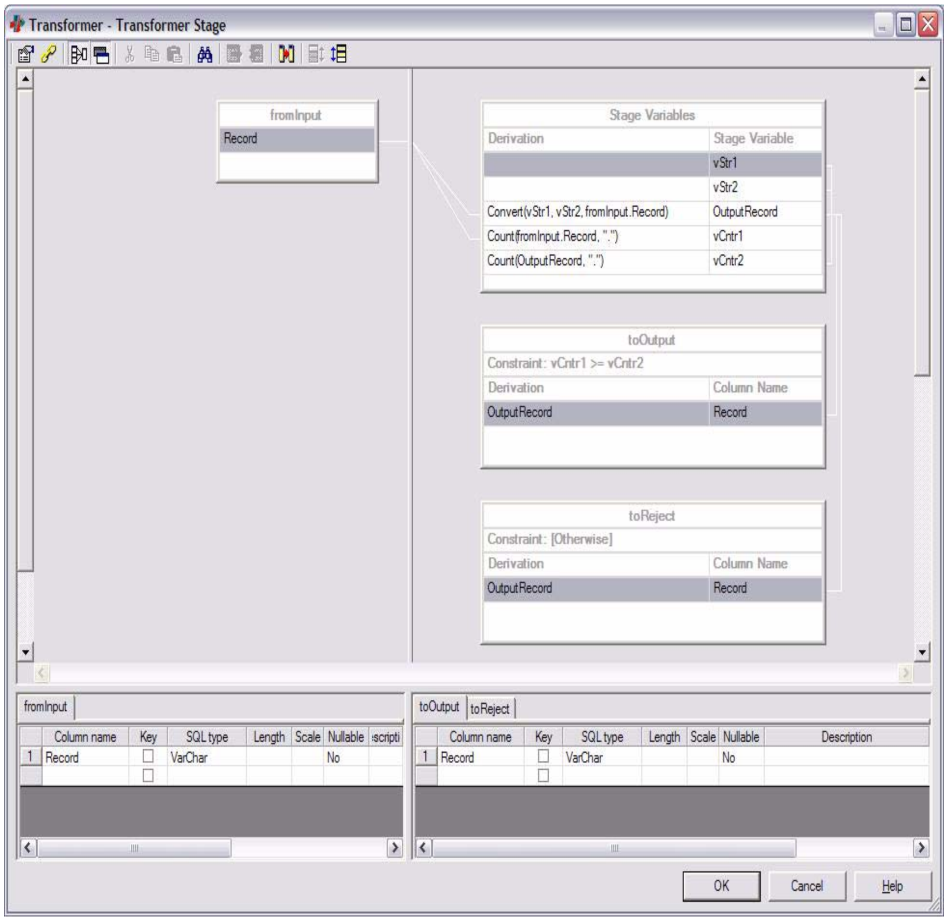
- vCntr1’s Derivation Expression is set equal to,

  Count(fromInput.Record, ".")

- And vCntr2’s Derivation Expression is set equal to,

  Count(OutputRecord, ".")

- Click OK when your Transformer equals what is displayed in Figure 17-8, Figure 17-9, and Figure 17-10.



fromInput

Record

Stage Variables

Derivation	Stage Variable
	vStr1
	vStr2
Convert(vStr1, vStr2, fromInput.Record)	OutputRecord
Count(fromInput.Record, ".")	vCntr1
Count(OutputRecord, ".")	vCntr2

toOutput

Constraint: vCntr1 >= vCntr2

Derivation	Column Name
OutputRecord	Record

toReject

Constraint: [Otherwise]

Derivation	Column Name
OutputRecord	Record

fromInput

	Column name	Key	SQL type	Length	Scale	Nullable	:scripti
1	Record	☐	VarChar			No	
		☐					

toOutput | toReject

	Column name	Key	SQL type	Length	Scale	Nullable	Description
1	Record	☐	VarChar			No	
		☐					

Figure 17-10 Transformer configuration, Screen 3 of 3.

Note: Transformer Stage Variables are set equal to their Derivation Expression upon the receipt of every row from the Input Link. In our example, 2 variables are set once, and then never changed; this is because those 2 variables have no Derivation Expression. The 3 remaining variables are set on every row as stated.

Transformer Stage Variables are set in order of their appearance in the Transformer Properties dialog box, one after another. While this is an important point to make in most use cases (this capability gives us a history of input data records, and we are able to compare the last record to the current), we do not see or benefit from this behavior in our current example.

The (output) Link Properties dialog for the Transformer can evaluate Transformer Variable values, to determine where output records from this operator are directed; example as displayed in Figure 17-9.

6. For the Column Import operator, complete the following;
   a. Under the Stage TAB -> Properties TAB, set the Input -> Import Input Column value to equal, Record, the name of the column the prior operator passes to Column Import.

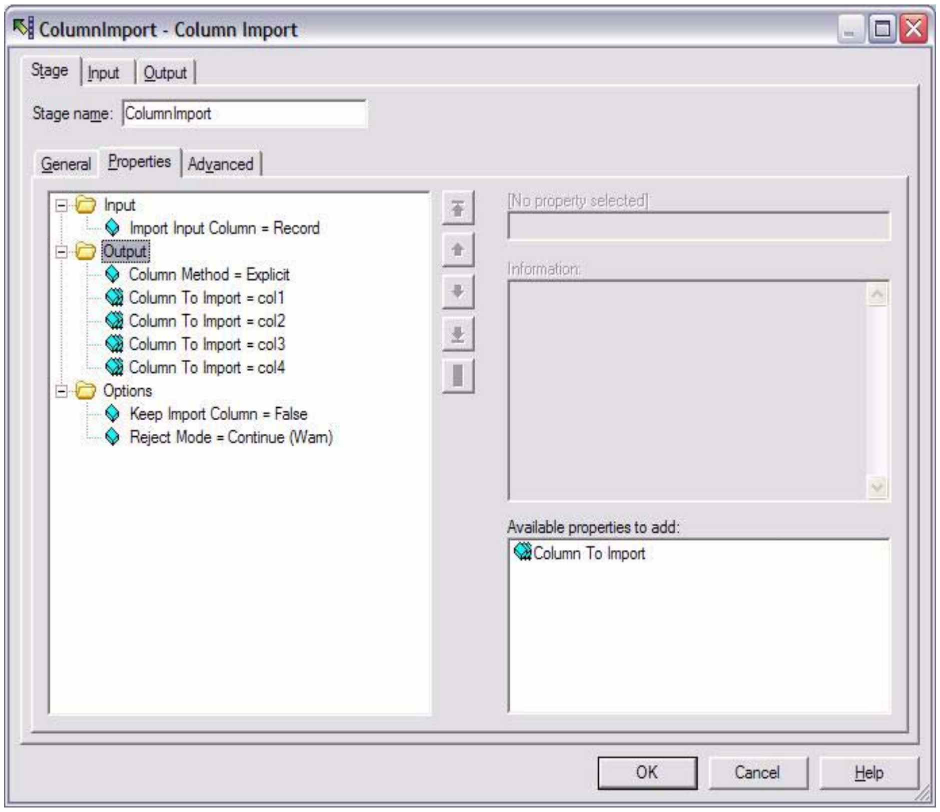
      Example as shown in Figure 17-11.
   b. Under the Stage TAB -> Properties TAB, Output, add as many columns as you have in the formatted input data file; our example displayed four columns.

      This is done by highlighting Output, Right-Click, and selecting Add sub-property -> Column to import.

      After adding the (sub-property), set the value to this control by highlighting this new entry, and adding a column name from the drop down list box displayed on the right hand side. Because no control or meta-data object is passing us these columns names, you will have to enter them manually.

      From our specific example, you are done when you have added 4 columns, named col1, col2, etcetera.
   c. Under the Output TAB -> Mapping, drag and drop all 4 input columns to the output side of the display.
   d. Under the Output TAB -> Columns TAB, you may optionally set output column data types and related properties.

      Click OK when you are done.



Stage name: ColumnImport

- Input
  - Import Input Column = Record
- Output
  - Column Method = Explicit
  - Column To Import = col1
  - Column To Import = col2
  - Column To Import = col3
  - Column To Import = col4
- Options
  - Keep Import Column = False
  - Reject Mode = Continue (Warn)

[No property selected]

Information:

Available properties to add:
Column To Import

Figure 17-11 Column Import operator.

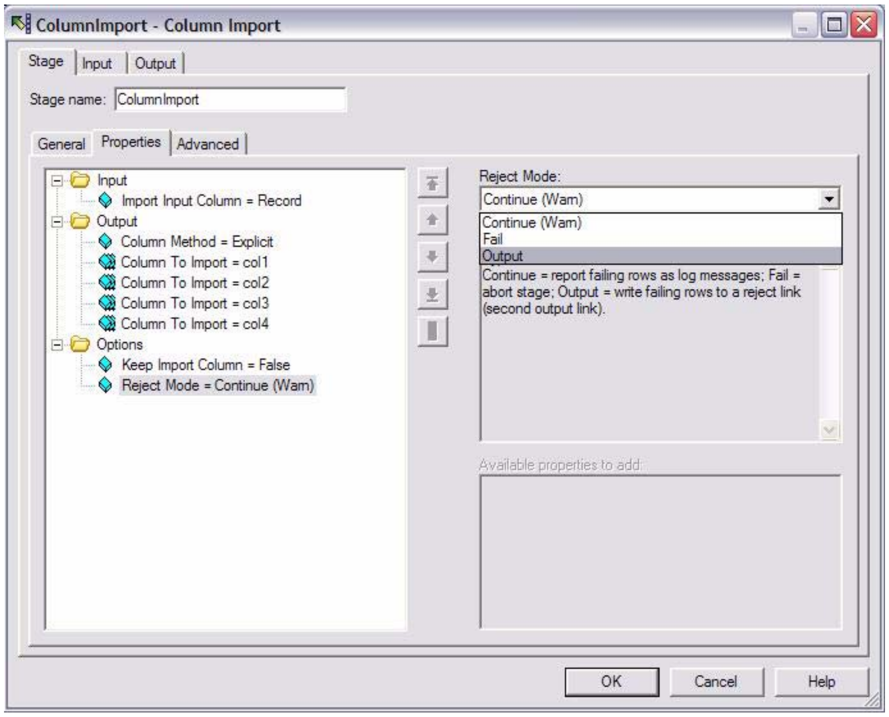
Note: The Column Import operator could itself generate reject records, the most common cause being a semantic integrity violation; trying to cast what was a VarChar input data type, to say an Integer, etcetera.

While we will not fully document this capability here, in general we state;

– Add a new, second Output Link to the Column Import operator. This second Link will default to type, Reject Link.

– In the Column Import operator, Stage TAB -> Properties TAB, set the Options -> Reject Mode property to Reject.

Example as shown in Figure 17-12.



Stage name: ColumnImport

- Input
  - Import Input Column = Record
- Output
  - Column Method = Explicit
  - Column To Import = col1
  - Column To Import = col2
  - Column To Import = col3
  - Column To Import = col4
- Options
  - Keep Import Column = False
  - Reject Mode = Continue (Warn)

Reject Mode: Continue (Warn)

Continue (Warn)
Fail
Output

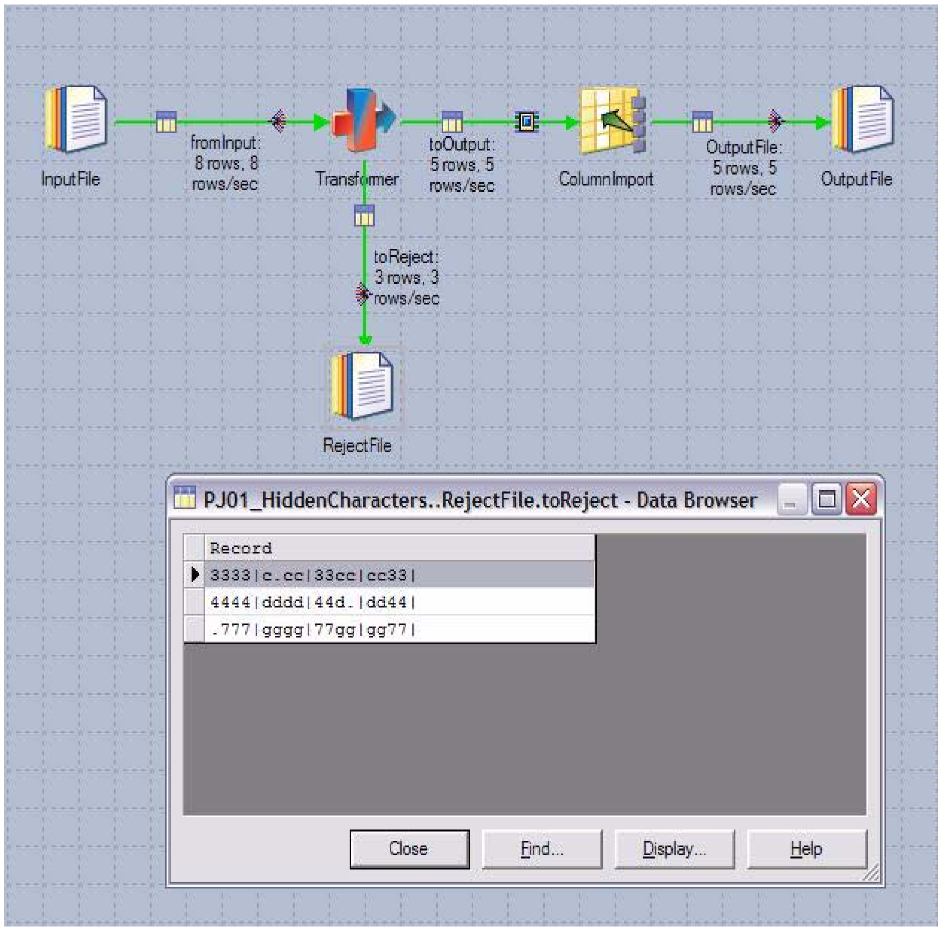
Continue = report failing rows as log messages; Fail = abort stage; Output = write failing rows to a reject link (second output link).

Figure 17-12 Optional configuration: Column Import with Reject Link.

7. Remainder of this example-
   - To complete the remainder of this example, fill in the required properties for the two Output Links, the Sequential File operators.
   - The RejectFile will only receive one column, because its input data stream was not processed via Column Import; RejectFile only receives the original input data column, Record.
   - OutputFile will receive the full 4 column input data stream.
8. Save, Compile and Run this sample Job-

   Per our specific example, 5 data records should pass to the OutputFile, and 3 should flow to the RejectFile.

   Sample output from the RejectFile is displayed in Figure 17-13. Notice that periods now replace the illegal/non-printable characters.


InputFile
fromInput: 8 rows, 8 rows/sec
Transformer
toOutput: 5 rows, 5 rows/sec
ColumnImport
OutputFile: 5 rows, 5 rows/sec
OutputFile
toReject: 3 rows, 3 rows/sec
RejectFile
PJ01_HiddenCharacters..RejectFile.toReject - Data Browser
Close
Find...
Display...
Help

```
Record
3333|c.cc|33cc|cc33|
4444|dddd|44d.|dd44|
.777|gggg|77gg|gg77|
```

Figure 17-13 Sample of RejectFile output.

17.3 Promoting code reuse from the above example

The example above creates an IBM Information Server (IIS) DataStage component Transformer operator that we might use in dozens of applications; the capability to reject loosely formatted records from an input data stream. The technology within DataStage to promote reuse of subsets of DataStage Jobs is the Shared Container.

To create and demonstrate a Shared Container, and from the completed DataStage Job above, perform the following;

1. Inside the IBM Information Server (IIS) DataStage component, Designer Program, open the Job we created in the example above.

   From the Menu Bar, select Save As, and complete the dialog box steps to save this existing DataStage Job with a new name.

   (This step gives us a back up of the existing Job, and offers some amount of traceability; you will be able to compare and contrast what you had, from what you are about to create.)

2. On the Parallel Canvas, highlight with a Single-Click the Transformer operator.

Note: A Single-Click can highlight one operator on the Parallel Canvas. You may also Click, drag and hold, to select multiple operators into a Shared Container.

3. From the Menu Bar, select, Edit -> Construct Container -> Shared.

   This action will produce a dialog box where you can name your Shared Container, and place it in the Project Repository in a specific location.

   Name your shared Container, and for ease, place it in the same folder as the DataStage Job you are currently editing. (Best practice may have you place this new asset in a parent folder to the Project Repository that specifically possesses Shared Containers.)

4. The above action will produce a display equal to that as shown in Figure 17-2.

   Save, Compile, and Run this new Job.

   This new Job should work exactly like the first.

Testing and developing farther

– If you are feeling adventure filled, open the Shared Container and change its properties; you can open the Shared Container by finding its entry in the Project Repository, and opening it, working with it like any other DataStage (full) Job.

– To fully ensure you understand how to use a Shared Container, create an entirely new IIS DataStage Job. To incorporate a Shared Container into this new Job, drag and drop the Shared container from the Project Repository onto the Parallel Canvas, example as shown in Figure 17-2.

  There is only one new technique you may need-

  - The Shared container Object we created had 3 Links; 1 for input, and 2 output. *And these Links had specific names.*

- Your newly create Job will also have 3 Links, but these Link names may differ. How is DataStage to know which Link from the Shared Container maps to those in the consuming Job?

  Figure 17-14 displays the solution.

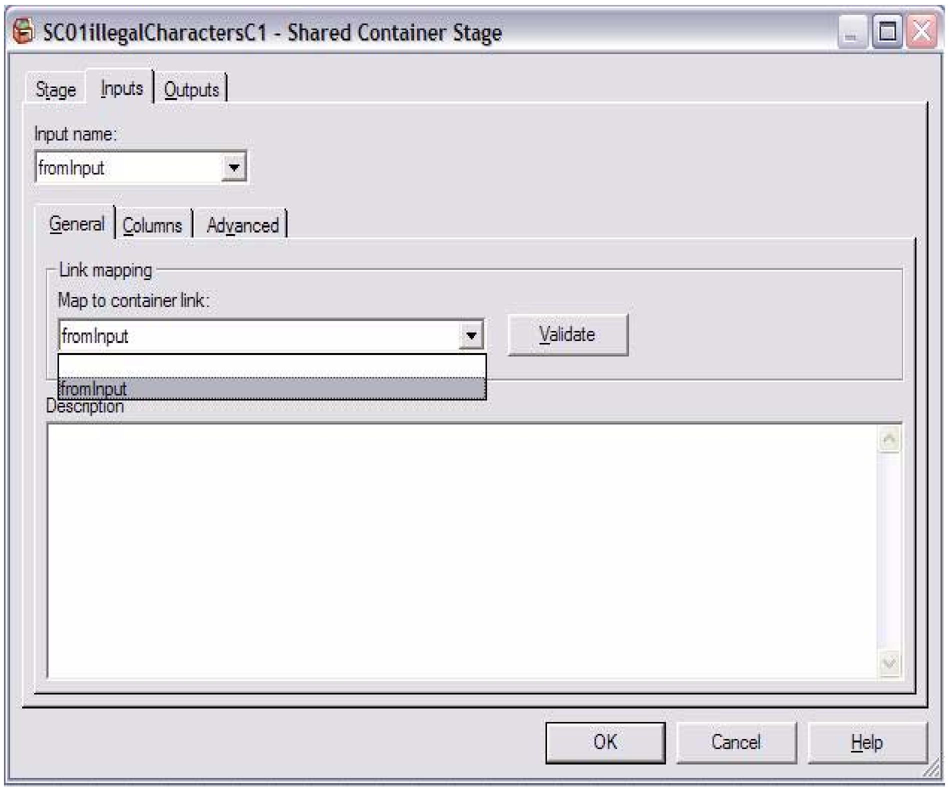


Figure 17-14   Properties dialog from the consuming Container object.

- In Figure 17-14, we have opened the Shared Container object in the consuming DataStage Job-

  Under the Input TAB -> General TAB, we specify which Shared container Link this Job Link is associated with.

  The Validate button is also useful to check the Link, and its column counts and data types.

  This specification and validation needs to be done for each Link; our example has 3, with the remaining 2 Links being located under the Output TAB.

17.4 Summation

In this document, we reviewed or created:

We detailed a technique for eliminating data records from our input data source which have illegal or non-printable characters; using a Transformer and string manipulation, we outlined a framework that could capture and reject nearly anything.

The above technique would make for a useful and reusable IBM Information Server (IIS) DataStage component asset. Using Shared Containers, we detailed how to promote the above example into a fully dynamic (imported by reference) reusable program asset.

Persons who help this month.

Danny ‘Lieutenant Dan’ Owens, Allen ‘Server’ Spayth.

Additional resources:

None.

Legal statements:

Special attributions: