Chapter 15.

March 2008

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

> How do I use my legacy operating system utilities and scripts with IBM Information Server?
>
> *Excellent question! (Presumably we are talking about things like Perl, Awk, Sed, GZip, etcetera.) Functionally, IBM Information Server (IIS) offers you a much more productive and easy to maintain, graphical, drag and drop means to replace all of that procedural/third generation programming. But, until you fully make the leap to the 21st century (kidding), we can preserve those assets and run them inside the IIS parallel framework easily and without issue.*

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.

15.1 Terms and core concepts

Unix, the original extract, transform and load engine?

For those who don't already know, Unix, and now Linux, are littered with literally dozens of handy utilities for text and data processing. By means of example, this month's IBM Information Server Developer's Notebook (IISDN) author was working on a server migration back in 1990, moving a hotel company's central reservation system from an aged Sperry computer (the server had water running through it for cooling), to a modern Unix SMP computer and relational database.

This hotel company's IT staff was writing the data extraction and conversion routines in COBOL, the development tool they knew best at the time. Lunch, which is a pretty easy event to plan for (happening every day at pretty much the same time), was an issue every day. Folks were always running late creating various deliverables. More than a few times, lunch was saved in a few minutes by writing an ad-hoc Unix Awk, Sed, Grep, or related script; this script replacing hours of COBOL development.

Unix text processing, versus say, IBM Information Server (IIS)

Well, that's an obvious question for folks who still think of products like the IBM Information Server DataStage component as an extract transform and load (ETL) product. 8 years ago or so, products like DataStage were just ETL products, but they have evolved, providing greater value to the enterprise. Consider the following;

- Again, this month's IISDN author just completed a proof for a new IBM Information Server (IIS) customer. This customer is similar to a FedEx or DHL (it wasn't either of those two companies), providing global transportation and logistics. With information hubs in various corners of the globe, on disparate systems, with disparate message and data sharing protocols, this customer was having quite a bit of issue with data quality and related.
- By means of just one example, we painted (drag and drop) an IIS DataStage Job, and deployed it as an always on, real time, data repair utility (a DataStage Job). We inserted this Job into an existing data stream that was having issue, using this new Job to repair a critical data and logistics need. Neither the source or receiving systems needed to be aware/impacted by this correction.
- The above is an example of a global and real time problem. If you return to the simpler and previous example of just extract, transform and load, IIS has several advantages over (Unix Shell) programming;

- IIS is graphical, drag and drop, very little programming per se. Unix Shell is line editor, third generation, procedural programming, harder to create and maintain.
- IIS is inherently parallel. The same single IIS Job you paint via drag and drop, can run serially, parallel on one computer, or parallel across several computers simultaneously, and it does this without any (Job) design changes.

  Unix Shell is not parallel aware; you'd have to write a Unix Shell script to work in parallel on one computer, which would be very hard. Running a Unix Shell script in parallel across multiple computers- Not something we'd recommend.
- All of the pre-built and pre-optimized performance awareness IIS has; for example, awareness of each modern database server's high performance parallel load and unload utilities, etcetera.
- IIS works across multiple platforms, versions and operating systems.
- And dozens of other benefits.

Common connector architecture; inputs, outputs, mid-streams

Generally within an IBM Information Server (IIS) DataStage component Job, there are two types of operators (stages). These are;

– Operators that act as end points; meaning, an operator that is either the initial input, or terminating output of a given Job. (They either input or output, not both.)

  Flat files, SQL connectors, and other operators commonly act in this manner.

– Operators that sit in the middle of a data stream.

  Sort, Aggregate, Transformer, and other operators commonly act in this manner. (These operators *input and output*.)

As stated, the topic of this document is incorporating operating system utilities and scripts (utilities) for use with IBM Information Server (IIS). The following comments are offered;

– Utilities can variably just receive data, just output data, or do both.

– If the utility is to act as the input operator to a DataStage Job (act as that Job's data source), then that utility must send output to *standard out*. (This utility should not exclusively output/write to a file.)

  This utility may receive DataStage driven variable parameters on the command line. (Example to follow.)

- If the utility is to act as the output operator to a DataStage Job (act as that Job's data destination), then that utility should receive *standard in*. (This utility should not read exclusively from a file.)

  This utility may receive DataStage driven variable parameters on the command line. (Example to follow.)

- If the utility is to operate in the middle of a data stream (sit in the middle of the DataStage Job data stream), then this utility should receive standard in and write standard out.

While the above may seem obvious, not every Unix/Linux Bourne Shell script (for example), is written in such a manner; some scripts expect to read from a file, and output to same. You could run those style of utilities inside a DataStage Sequence, but you could not place them inside a DataStage Job data stream; by design, those utilities aren't stream capable.

Note: Why use standard in and standard out for mid-stream operators, and standard in or standard out for end point operators?

These operating system utilities and scripts (utilities) are going to run in parallel, on multiple and concurrent CPUs. If these utilities can only write to serial and fixed files, that kind of defeats the purpose of a highly performant parallel framework like IBM Information Server.

15.2 Examples using Unix/Linux utilities

This section contains 4 example IBM Information Server (IIS) DataStage component Jobs. Comments related to these 4 examples follow;

- PJ01_UsingPerl demonstrates how to run a native Perl routine inside an IBM Information Server (IIS) DataStage component job.

  This example puts the Perl routine in the middle of a DataStage Job data stream; Perl routines could also act as data stream end points; input or output.

- PJ02_UsingAwk is similar to Job PJ01 above; Awk differs from Perl, in that Awk is generally hosted inside a Unix/Linux shell, and so the set up of standard in and standard out differs. (Perl doesn't need a shell; Perl supplies its own runtime environment.)

  This example also sits in the middle of a DataStage Job data stream.

- PJ03_UsingGZip demonstrates how to run an operating system utility (in this case GZip), as a data source.

  Similar to the manner in which most data sources can start multiple, concurrent input scan agents, this Job displays how to get higher throughput through use of the IIS parallel framework.

- PJ04_SpecialOutputCase displays use of a Unix/Linux wrapper, and data outputs.

  GZip (for example), can not receive its input from standard in, as is the need with data outputs. As a simple work around to this GZip limitation, we place a simple and generic Unix/Linux wrapper around this utility to solve this limitation.

Note: All of the examples that follow, read from, or write to, a Sequential File operator. For simplicity, we used a 2 column format; col1, col2, Varchar data type.

In the case where the example read or writes to GZip, that command syntax is documented.

Example 1: PJ01_UsingPerl

Figure 15-1 displays the basic IBM Information Server (IIS) DataStage component Job layout we use in this Job, and the next Job, PJ02_UsingAwk. The following is offered:

- A Sequential File operator starts and ends the Job.

- We set the file name, record and field delimiters, and created 2 columns of a generic data type (Varchar).
- You could use any complexity of column counts and data types.

– A Generic Operator from the Processing drawer of the Palette view sits in the middle of the DataStage Job.

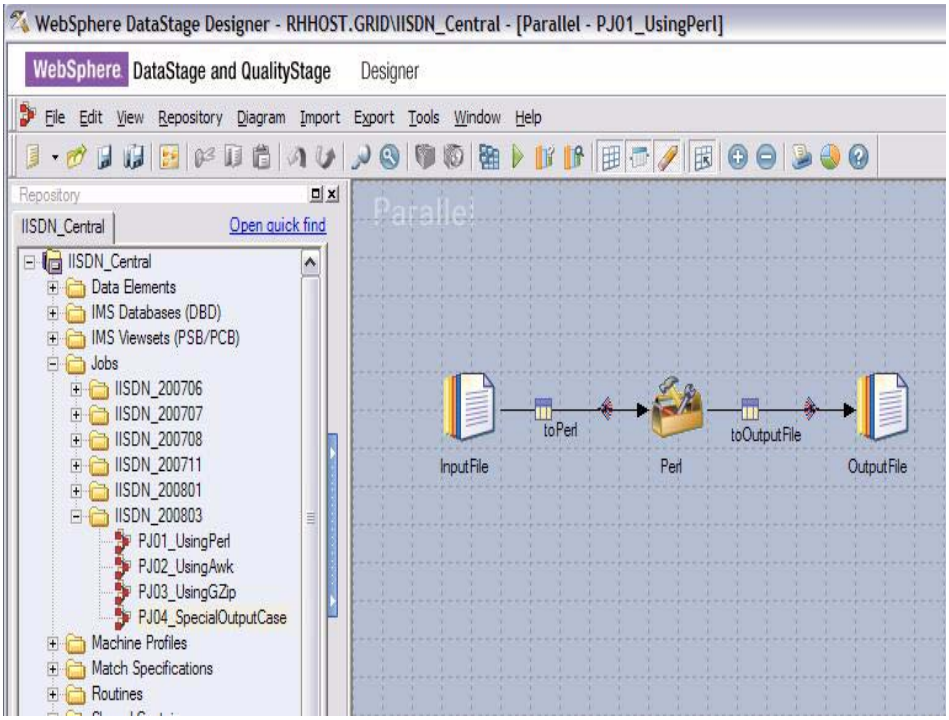

Figure 15-1 Basic job layout for both PJ01_UsingPerl, and PJ01_UsingAwk.

Figure 15-2 displays configuration of the Generic Operator, used in this example to invoke Perl. the following is offered;

– You have to modify the Output TAB -> Columns, just like every other DataStage Job Operator.

  This task is considered a very entry level skill, and is not detailed here.

– The only real modification is on the Stage TAB -> Properties TAB, inside the Options -> Operator text entry field.

  Here we enter the value,

  perl {*absolute pathname to Perl language script*}

  Example as shown in Figure 15-2.

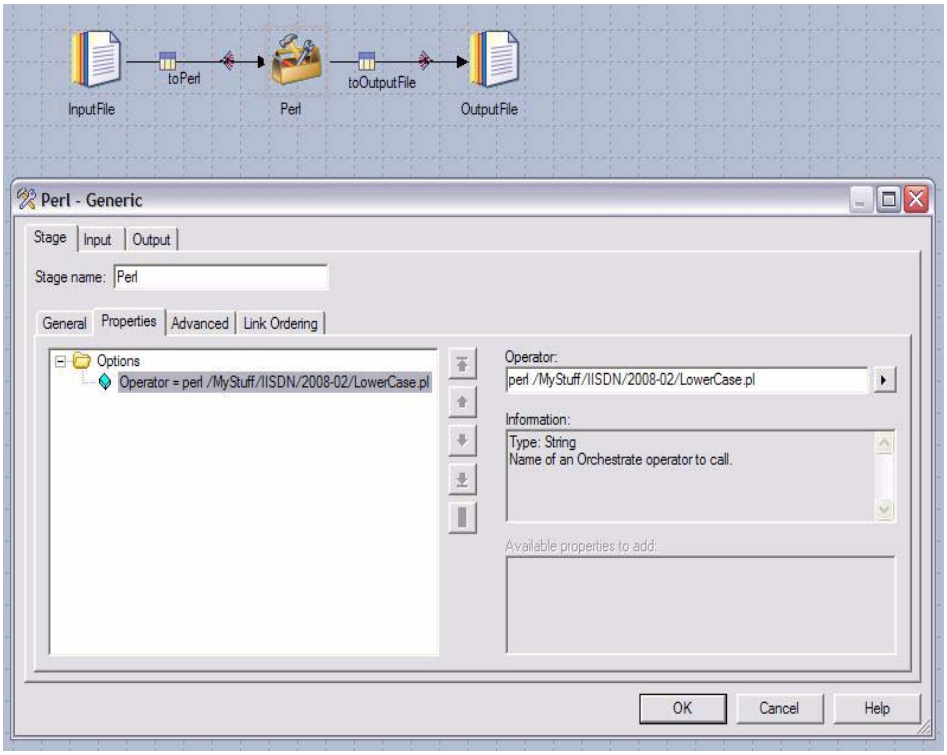

Figure 15-2 PJ01_UsingPerl, Generic Operator.

Example 15-1 displays the Perl script created for this example. A code review follows this example.

Example 15-1 Sample Perl program, converts all input to lowercase text.

```
#!/usr/bin/perl


@inputArray=(<>);

  #

my $cntr = 0;


while ( $cntr < @inputArray )

  {


  my $lc_inputRecord = lc $inputArray[ $cntr ];
```



```
print $lc_inputRecord ;

$cntr++;

}
```

A code review of the sample Perl program in Example 15-1 is listed below;

- Basically this is the minimum Perl program syntax to accept standard in (reference to @inputArray), and then write to standard out.
- The only real processing this Perl program does is invoke "lc", which in this context will fold all input to lower case text.
- If one needed to advance this Perl example, you'd generally add new Perl syntax program code inside the while loop.

Example 2: PJ02_UsingAwk

Figure 15-3 displays the PJ02_UsingAwk example, and the means in which we need to configure this Generic Operator. Comments related to this example;

- So again the basic flow is, Sequential File input, then Generic Operator, then Sequential File output. Only configuration of the Generic Operator differs Job over Job (PJ01_UsingPerl versus PJ02_UsingAwk).
- In Figure 15-2, the Options -> Operator we invoked was "perl {file name}", because perl supplies its own run time environment. But Awk doesn't do that; Awk needs to run inside a shell.

  In Figure 15-3, we see the Options -> Operator is "sh {filename}", because Awk expects to run inside a shell.

Note: Obviously we could make each of these samples Jobs more complex. Its just that we chose not to.

These operating system utilities and scripts run in parallel and can be made part of the most complex and performant DataStage Jobs you've ever previously needed.

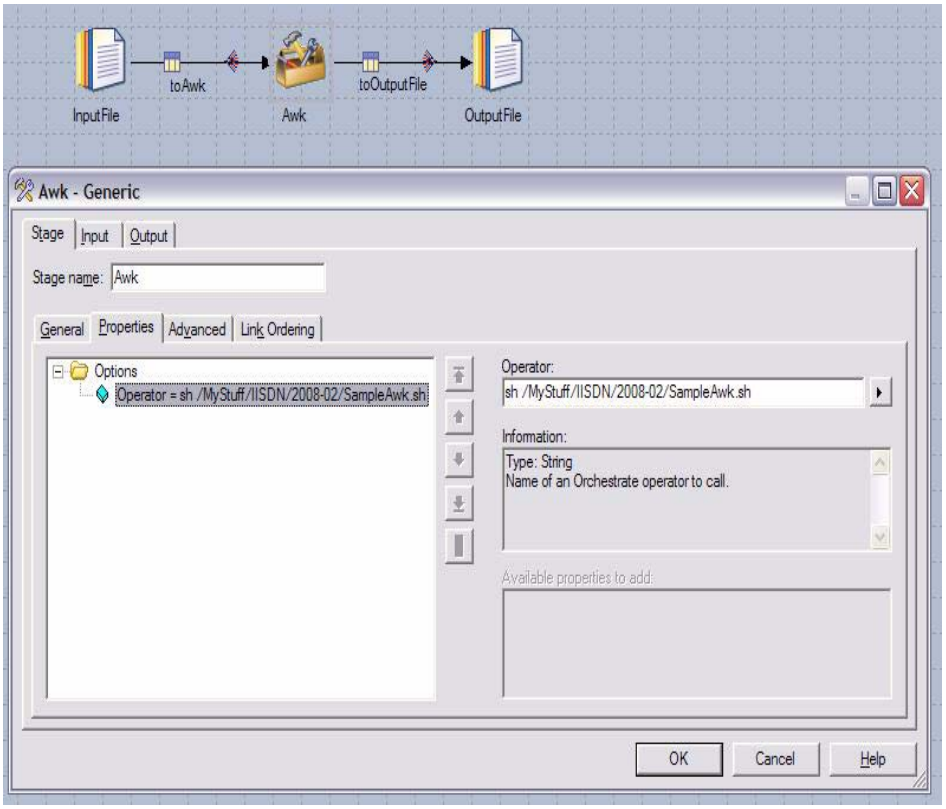

Figure 15-3 PJ02_UsingAwk, Generic Operator.

Example 15-2 displays the Bourne Shell and Awk script created for this example. A code review follows this example.

Example 15-2 Sample Shell script, with Awk, converts all text to lowercase.

```
#!/bin/sh

awk '

  {
  print tolower( $0 );
  }

  '
```


A code review of the sample Bourne Shell and Awk program in Example 15-2 is listed below;

- Per the construct in Example 15-2, we read from standard in and write to standard out.
- Again, this example folds all input text to lower case.
- Per Awk required syntax, those are left single quotes in that file.

Example 3: PJ03_UsingGZip

Figure 15-4 displays sample DataStage Job, PJ03_UsingGZip. To use GZip as a data source, we could once again use the Generic operator and configure its properties as we did before. If we use the External Source Operator, we can achieve greater performance through parallelism. The External Source Operator allows one to start numerous concurrent (parallel) read operators.

In Figure 15-4, we set the Output TAB -> Properties TAB, Source -> Source Program. Further comments include;

- Here we set the Source Program value to,

  gzip -c -d {absolute pathname to file.gz}

  - The "-d" flag tells Gunzip to read target file in place (uncompress it in place).
  - The "-c" flag tells GZip to output results to standard out.
- We may configure as many parallel and concurrent output data sources as make sense. To accomplish this, Right-Click Source, and select, Add sub-property -> Source program, to add another source program.

  Configure this program as you did before. this program need not be another GZip; it could be any utility that outputs to standard out.

Note: GZip files suffer a variable length (internal) record format. And there are no GZip utilities that allow multiple invocations. Huh? While you can achieve parallelism by invoking numerous and concurrent GZip output streams (source programs), each will have to operate on a different target file.

This is a current limitation of GZip. If someone writes a GZip utility that can accept an offset parameter on the command line, that limitation of GZip would go away.

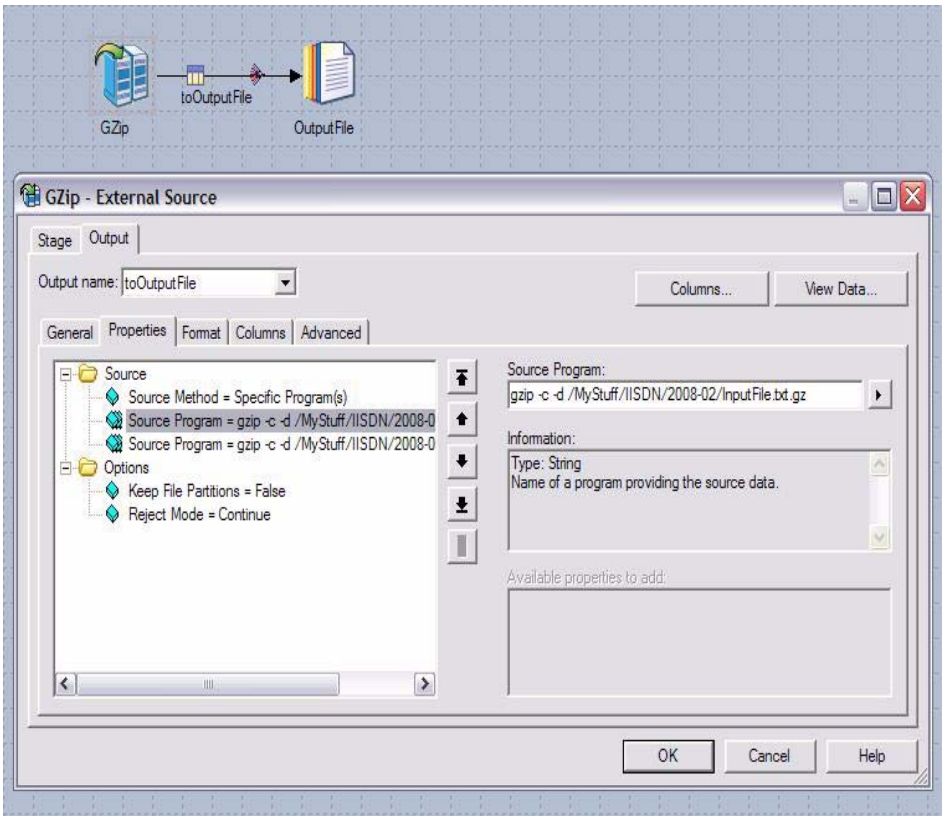

Figure 15-4 PJ03_UsingGZip, External Source Operator.

Example 4: PJ04_SpecialOutputCase

Sample DataStage Job, PJ04_SpecialOutputCase, is displayed in Figure 15-5. In this example, we wish to output to GZip, and have GZip compress our results, the output from our DataStage Job data stream.

> *But GZip does not accept data from standard in, as is required for this to work. As a solution, we place a wrapper around GZip to accept standard in. This solution works for GZip, and any other utility suffering this limitation.*

Lastly, we pass a command line parameter to this wrapper; a command line parameter which we know from our other work could be a Job Parameter (a variable, or Sequence driven value).

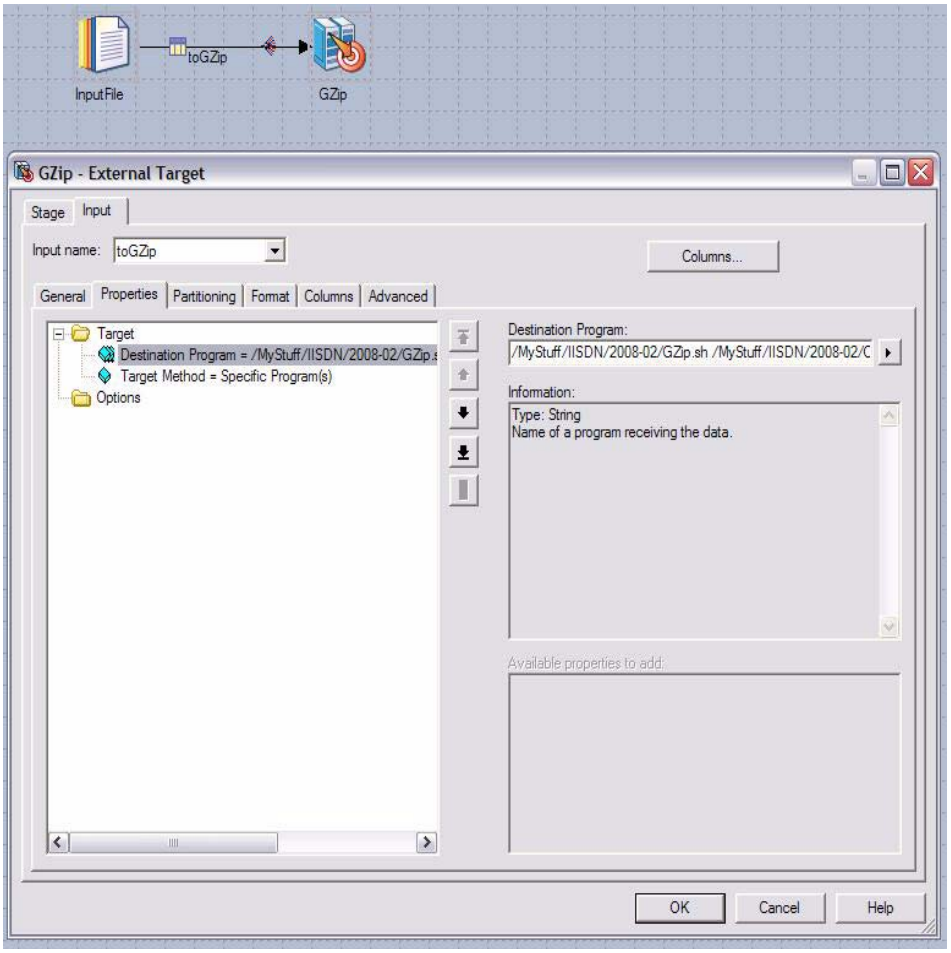

Figure 15-5 PJ04_SpecialUseCase, External Target Operator.

A code review of the Bourne Shell wrapper we used for GZip follows in Example 15-3.

Example 15-3 Sample Shell script wrapper to support standard in to GZip.

```
#!/bin/sh

cat > $1 <&0

gzip $1 2> /dev/null
```


From Example 15-3, the following is offered;

- The (final) output file name to the script in Example 15-3 is passed on the command line, and could represent a DataStage Sequence Parameter, or other variable.

  Bourne Shell refers to the first argument passed on the command line as, $1.

- The "cat > $1" portion of this line says output to the variable value of the first argument to this program.
- The "<&0" says read from standard in.
- The gzip argument on the last line could be replaced by any useful operating system utility.

15.3 Summation

In this document, we reviewed or created:

We detailed using (re-using) legacy operating system utilities and scripts inside the IBM Information Server parallel framework, for performance, and also the benefit of preserving legacy assets (older program scripts and utilities).

Persons who help this month.

Adam Fecadu, Ken Wilson, Aliky Kouroupis.

Additional resources:

None.

Legal statements:

Special attributions:

The listed trademarks of the following companies require marking and attribution:

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Microsoft trademark guidelines

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Intel trademark information

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

Other company, product, or service names may be trademarks or service marks of others.

