

# The Power of Variables

## The Current Scope

Many people only think of variables as input data changes or parameters that could change the execution of the task. They are missing the ability to exploit variables as a way to minimize changes from one processing environment to another.

They are also missing an opportunity to simply extract a naming standard that ultimately results in dozens or hundreds of replicated tasks, and simplifying it to the source of the change, i.e. using a portion of the file name.

## The Opportunity

Using variables based on lifecycle stage, or business entities have been proven to decrease the number of workload task definitions, as well as reduce the effort or the 'energy' the staff spends doing repetitive updates based on geographical location, data input or business entity.

Let's look at examples in each of these cases. The initial power of variables came when scheduling products were gaining widespread use in data centers across the world.

Then, second generation scheduling products were introduced that expanded the use and value of variable substitution and allowed the simple substitution of characters in job names and in the executables themselves, allowing 'cookie cutter' definitions to be reduced to a

standard, variable-ized, name in a single workload definition. This struck a chord for some customers, especially those processing regional retail stores, or other large numbers of similar business entities. For example, a large nationwide retailer processed regional workload for 14 different credit centers which all processed the same exact workflow tasks, in the same exact sequence, only the names changed across the regions.

The executables themselves used variables, but up until this point, job scheduling products were not able to apply the same efficiencies. With this new capability, the retailer was able to reduce the number of job definitions from 832 to 74, in this one workflow alone.

### Taking it to the Next Level

In the newer job scheduling solutions, variable substitution could not only be applied at the task definition level, but could also be propagated into the executables in file names, or other parameters within the script or JCL.

This expanded the value that could be achieved and ultimately contributed to reducing overall workload definitions.

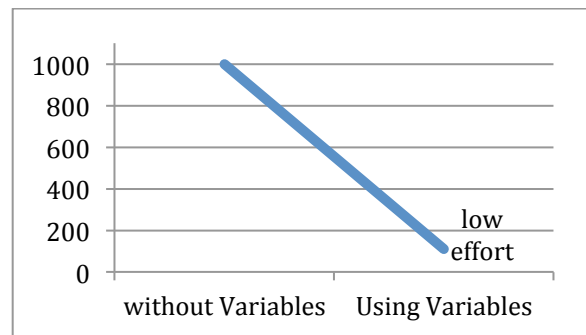
We have seen companies exploit variables to drastically reduce the amount of workload definitions and could effectively reduce and redeploy staff to other, more important, work assignments!

### Unlimited Possibilities

Many company's process similar workload tasks for different business entities they support. This could be as simple as 10 tasks executed on 100 servers, or as massive as 1000 tasks executed for 50 different business entities. If you do the math, using variable substitution, the first example would reduce 1000 definitions to 10.

$$10 \times 100 = 1000 \text{ definitions or}$$
$$10 \times 100 = 110 \text{ definitions}$$

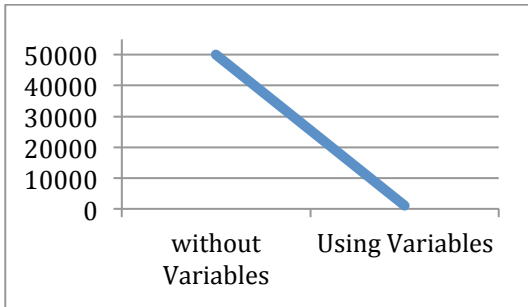
Which formula would bring better efficiency?



When you think about the amount of time and effort spent defining and maintaining these workload tasks, there could be significant savings that could allow you to reduce or redeploy staff to do more important functions, making better use of their time, and providing better, more consistent service to the business.

The second example is even more dramatic:

$$50 \times 1000 = 50000 \text{ definitions or}$$
$$50 \times 1000 = 1050 \text{ definitions}$$



This method of exploiting variables has allowed companies to expand their services to new business entities with very little impact, as the base definitions are already there, they simply add a new trigger to supply the new variable names.

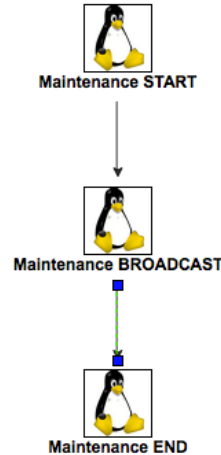
### Simplifying Maintenance

It also makes additions to the base workload much simpler, as you only add new definitions in one place, not in many places. This simplifies and reduces the implementation effort.

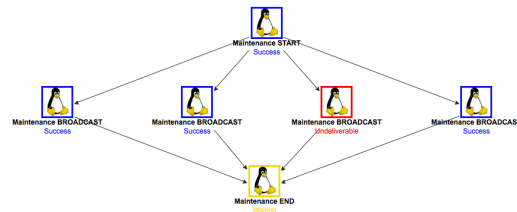
Let's take another look at the example with 10 tasks to run on 100 servers. Using variables would simplify the number of executables, but how do I disperse this work to the servers? You would still need workload definitions pointing to each server.

But what if you could define the 10 tasks, one time, then the automation software could disperse the definitions and execute the workload on each the 100 servers? This would eliminate the need to create executables on each local server and reduce the setup time and manual effort.

Even if the disbursement of work was in the middle of the workflow, as seen below, you should be able to define one task, and have it be broadcast to run on every server in the specified group.



When this workflow executes this middle job expands out and executes on every server in the defined cluster.



Another powerful capability provided by built-in variables is transitioning workload from Development to Test to Production. One of the best ways to minimize errors occurring as you progress thru the application lifecycle is to omit any **changes** from being introduced.

By utilizing simple variables, you can define one end-to-end business data workflow, substitute generic variables with environment specific values, then simple have a trigger for each environment.

In a workflow definition, the task names can be relative to the stage of the lifecycle and are automatically changed when workflows are promoted to various stages. For example, a generic task named...

**RGN\${stage}\_process\_input 1000**

Could resolve to...

**RGNDEV\_process\_input 1000**

...in the development environment.

Then, when promoted to TEST environment, it would be automatically changed to resolve as...

**RGNTST\_process\_input 1000**

Using built-in default variables allows you to easily create variables based on naming standards.

<u>PayDEV run</u>	<u>REGNTH Billing Process</u>
<u>PayPRD run</u>	<u>REGSOUT Billing Process</u>
<u>PayTST run</u>	<u>REGWEST Billing Process</u>

Using substring notation, only a portion of the trigger name is used to assign a value to the variable.

Exception criteria can be used to stage tasks to only run in the environment they are designed for, allowing new tasks to be executed only in development environments, but not run in TEST or PRODUCTION until they are promoted.

Evaluate At:	Trigger Time
Name:	stg
Operator:	=
Value:	dev

Deploying this same workflow for multiple business entities is now as simple as defining one new trigger, NOT an entirely new definition of workflows and tasks.

<u>REGCNDNDA Billing Process</u>
<u>REGGEAST Resource Testing</u>
<u>REGGRMN Billing Process</u>
<u>REGNRTH Billing Process</u>
<u>REGSOUT Billing Process</u>
<u>REGWEST Billing Process</u>

This minimizes the effort and thereby decreases the lead-time for new workload definitions to take effect. Tasks that used to require a 3-5 day advance notice can now be done in minutes.

## Variables Based on File Names

Perhaps the most beneficial use of variable substitution comes when a company receives an large amount of the same file type, where only the file names are slightly different, indicating the business entity for which they contain data.

As input files began to trigger the processing of workload data, the need came to be able to extract a portion of the filename and assign that as the value of the variable.

This eliminated manual manipulation of data that often resulted in typos and failures in processing. Some organizations found that this

capability would drastically reduce the manual efforts currently required to enter this information, and would result in reducing months of manual effort.

Eliminating the human factor reduced the production errors and ensured that SLA commitments were met, preventing service level breaches.

In the example below, the only important value was the last level of the file name, distinguishing which store the file was from.

Specific Business File Names
MY.CRITICAL.BUSINESS.FILE.FROM.STORE1234
MY.CRITICAL.BUSINESS.FILE.FROM.STORE7349
MY.CRITICAL.BUSINESS.FILE.FROM.STORE4532
MY.CRITICAL.BUSINESS.FILE.FROM.STORE6321
Generic File Trigger
MY.CRITICAL.BUSINESS.FILE.FROM.STORE\${STORE#}

File-based built-in variables allow customer to easily assign a value to the variable and minimize their definitions of workload tasks.

### Using Built-in Variables

Today's workload automation solution must have pre-defined variables, based on common practices to supply information to the workload processes themselves. We've talked about using the trigger names and portions of a file name.

The most common use of variables to replace manual effort today is the date/time variables. Many times the date/time is part of the filename itself, which can be extracted as previously described in this document. But

maybe you need to extract the file date of the file that initiated the workload.

Being able to use a variable can eliminate manual effort. In businesses with fluctuating volumes, it might be important to extract the file size information and use that to adjust SLA times. Or, you might just need to scan the content and use the results of the scan as the variable assignment.

<b>\${ops_trigger_file_date}</b>	the file date of the file that fired the trigger
<b>\${ops_trigger_file_size}</b>	the file size of the file that fired the trigger
<b>\${ops_trigger_file_date}</b>	the file date of the file that fired the trigger
<b>\${ops_trigger_file_scan}</b>	the scan results

When workload is launched into the system, there are a number of built-in functions that can be used to resolve changing values within the processing.

These can be used to extract current date information and either rollback the current date values, or advance them forward. This functionality can eliminate manual effort being used today.

Built-in Functions	
<b>\$_date([format, day_offset, hour_offset, minute_offset])</b>	default format is yyyy-MM-dd HH:mm:ss Z Standard JAVA

## Exploiting the Value

Companies are looking at ways to process data more immediately in response to business activities. Some are even looking at file arrival alone being the significant event to trigger workload processing to initiate.

It used to be that all the files were stored in a location and at a specified time the workload processing would begin with whatever files had been received. Now, with business models changing to a more real-time, instantaneous processing of information, we are seeing new requirements to more immediately initiate workload processing as soon as a file arrives.

Customer expectations have evolved with the introduction of the Internet, Wi-Fi access, smart phones, iPads, etc. IT has to change the way they respond to these business challenges, and variables can play a key role in keeping the overhead to a minimum and providing the agility the business needs.

Variables have been used since the beginning of JCL on the mainframes and have expanded across the different operating systems and environments. They can allow you to simplify definitions, reduce maintenance and accelerate production implementations, bringing valuable capabilities to the business much faster.

The value related to exploiting variables within your environment can grow exponentially based on the processing you perform. Let's take a different look at some of the numbers we talked about earlier.

Customers have reportedly been able to apply variable capabilities to more quickly react to the business growth and enable faster implementations.

