The Manipulation of a Step's Status in z/OSMF Work-flows Through Variable Passing

Keith Miller

White Paper
June 21, 2015
Version 1.0

IBM Washington Systems Center

Introduction:

The manipulation of the complicated syntax of z/OSMF work-flows can be especially challenging depending on what needs to be accomplished. This has led to difficulty in trying to engineer new work-flows as well as modify existing ones especially with regard to variable passing. An exorbitant amount of time can be wasted trying to troubleshoot the plethora of errors that can come back when the XML of the work-flow definition file is not structured properly; a consequence of the stringent syntax of work-flows.

The aim of this document is to outline one of the processes related to z/OSMF work-flow definition file creation that is not immediately obvious to the z/OSMF beginner; variable passing. The information provided should ease some of the pain that comes along with the constant testing necessary to get a work-flow working properly.

This document assumes that the user is somewhat familiar with some of the general wording related to z/OSMF work-flow definition files. Any intricate ideas will be thoroughly described for the edification of the user. It is also noted that this is for z/OSMF V2.1 and higher.

Background:

For the user, convenience is of the utmost priority, especially in z/OSMF where larger work-flows can range upwards of two hundred plus steps. A way of automatically altering the state of steps outside the realm of what the user can do is a way of fulfilling this convenience factor.

Of these process two are the most important in accomplishing this:

- Passing variables from a REXX exec to a work-flow.
- Passing variables from a work-flow to a REXX exec.

Z/OSMF allows you to pass variables from outside a work-flow into it to fulfill certain conditions that allow you to change the state of a step. For an extreme example, the user can complete a step that submits a job that alters the state of all steps in the work-flow from Ready to Complete. This is not an ideal situation but it does illustrate the power of variable passing in work-flows, and can lead your mind to wander about the other applications of this.

A more realistic example of the uses of variable passing is given below:
If a customer is completing a work-flow it is quite possible that they will be completing steps that relate to features that they don't even have installed on their system. This is where the variable passing comes in. It is possible to run a command (D PROD,STATE) that comes back with what features are enabled on their system, and through a REXX exec the data returned from the command can be parsed, and the variable values can be passed back to the work-flow to set the unnecessary steps to skipped. Details on how to accomplish what is stated in this example is provided in the subsequent sections.

Again, the applications of variable passing can extend beyond what is presented here.

A Sample Step Template

Below is a sample picture of a top level step in a work-flow definition file. This image will be very important as the future sections refer to it quite heavily.

```
<step name="Step0">
    <title>Submit JCL job</title>
    <description>In this step, you execute a JCL job to pass a variable to a REXX exec as well
        as change a variable in the work-flow</description>
    <variableValue name="inputcmd"/>
    <instructions>Instructions place holder.</instructions>
    <weight>5</weight>
    <skills>System Programmer</skills>
    <template>
      <fileTemplate substitution="true">stat_exec.txt</fileTemplate>
      <submitAs maxRc="0">TSO-REXX-JCL</submitAs>
      <output needResolveConflicts="false">/tmp/var_inputfile.txt</output>
      <predefinedVariable name="inputcmd"> ${instance-inputcmd}</predefinedVariable>
    </template>
</step>
```

Figure 1

A quick description of each of the most important tags found in the picture is provided. Later sections will go more into detail regarding the actual use of each tag. The order of the tags in a work-flow's steps is assumed to be the same across all steps in the work-flow.

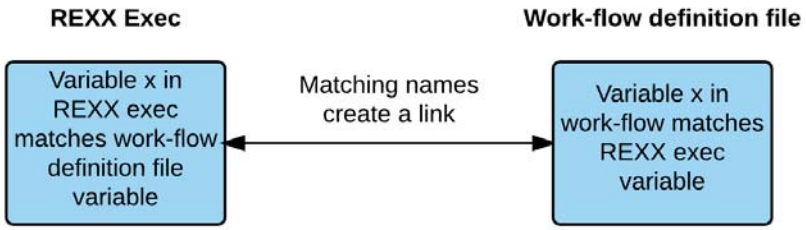
<variableValue>	Tag that creates a user input area in the perform tab of the step in the work-flow. The choice in the <choice> tags (figure 3) is passed to the variable name after the user selects what they want.
<template>	An anchor tag that holds all of the other tags as described below.
<fileTemplate>	Tag that loads the REXX exec that is in the same directory as the work-flow definition file. In this example the file is called stat_exec.txt.
<submitAs>	Tag that tells the work-flow to wrap the exec from <fileTemplate> in a certain JCL template so that the REXX code can run when the job is submitted.
<output>	Tag to specify a file directory to replace the $_output variable in a REXX exec. In this example $_output would be changed to /tmp/var_inputfile.txt upon loading the perform tab.
<predefinedVariable>	Tag that takes a variable and allows for its value to be input into a REXX exec.

Passing Variables from REXX Exec to a Work-flow

The usefulness of being able to pass a value between a REXX exec and a work-flow can prove extremely helpful in cases where you want to use an exec to modify the state of a step in a work-flow. The passed variables can be used to fulfill the <expression> and <condition> tags that are described in a later section.

Before a description of the process to pass a variable is given the most important part of the process is noted. The variable names defined in the exec **must match** the variable names defined in the work-flow definition file or it will not work. The matching variable names allows z/OSMF to "see" the updated variables in the exec as well as in its own work-flow. **Figure 2** provides a visual representation of this.

Figure 2



Below is a picture outlining the command in REXX to use to update the variables in a work-flow. The ADDRESS SYSCALL with a WRITEFILE modifier is a system command that allows the exec to create a file on your system for z/OSMF to read.
More info on the syntax of this command can be found here:
http://www-01.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.bpxb600/bpx1rx52.htm

The most important part of this command is the $_output. The $_output corresponds to the directory specified by the user in the <output> tags in the work-flow (in this case it would be /tmp/var_inputfile.txt). It is displayed and filled in by z/OSMF when the user running the work-flow from the perform tab is about to submit the job.

```
/* Parse data like you normally would using REXX    */
/* and load each variable into a stem. For example  */
/* a name could be 'variable1_status = Disabled'    */
wrc = ADDRESS SYSCALL "WRITEFILE $_output 777 data."
```

The simple act of writing the variables in the stem data. using $_output causes the all of the variables with corresponding names inside of the work-flow to be updated.

This is true as long as:

- The specified directory exists.
- You have permission to write to the directory specified.
- The variable names in the stem match the names defined in the work-flow.

The way it works is that Z/OSMF reads the file created after the variables in the stem are written to it. It then compares the variable values in the file to what the current variable values are in the work-flow and updates them to reflect what was found in the file.

This automatic update by writing to a file is a built in feature of z/OSMF and can be taken for granted from here on out. It also is noted that you **must use $_output** in order for this to work. Using a hard-coded directory name inside of the exec (even if it is the same directory name specified in the work-flow) instead of $_output will not work.

Passing Variables from Work-flow to a REXX Exec

Now that we have covered getting a variable from an exec to a work-flow the next order of business is to pass a variable from a work-flow to an exec. This can be done rather succinctly, and is slightly less involved than than the last step.

Figure 3 relates to a sample work-flow variable created to illustrate how it relates to the REXX code.

Figure 4 relates to the REXX code and how it takes the value of the variable from the work-flow and uses it in the program.

Figure 3

```
<variable name="inputcmd">
    <label>Question</label>
    <abstract>Place holder text</abstract>
    <description>Do you want to run D PROD,
        STATE or D PROD,REG? </description>
    <category>started</category>
    <string>
        <choice>Use D PROD,STATE.</choice>
        <choice>Use D PROD,REG.</choice>
    </string>
</variable>
```

```
runcmd = '${inputcmd}'
if runcmd = 'Use D PROD,STATE' then do
  command = 'D PROD,STATE'
end
if runcmd = 'Use D PROD,REG' then do
  command = 'D PROD,REG'
end
```

Figure 4

If you return to **Figure 1** you will notice a tag called <predefinedVariable>. This is the tag that allows us to pass the variable value defined in **Figure 3** in the work-flow to the REXX exec in **Figure 4**. Notice how the name of the variable in the exec and the name of the variable in the work-flow XML are the same exact name. The most important part of **Figure 4** is "${inputcmd}", this is where the instance of the variable is substituted into the exec. The rest of the code displayed is just to show how the variable can be used.

The variable passed in can then be modified based off how the exec is programmed and then passed back to the work-flow using the SYSCALL command as discussed in the previous section.

Expression and Condition Structure

Now that we have learned to pass variables both ways, we need a way for the work-flow to actually "see" what was passed into it. This is where the condition and expression tags come into play.

Below is a sample picture of the syntax of the <condition> and <expression> tags in z/OSMF. This block of XML and any variations of it will go under the </description> tag of any step that you want altered. So for example in **Figure 1** one above, this code would would go between the </description> and the <variableValue> tags.

This example specifically relates to using a specific variable to change the state of a step in a work-flow.

```
</description>
<condition>
    <expression><![CDATA[(${Step0.stepState}=="Complete")?true:false]]></expression>
    <description>If Step0 is complete set check what is in the stateExpression tag.</description>
    <targetStateSet>
    <description>Set this step to Skipped if the variable1's value is disabled.</description>
        <stateExpression state="Skipped"><![CDATA[${instance_variable1_status} == 'Disabled']]></stateExpression>
        <!--more then one stateExpression are supported here.-->
    </targetStateSet>
</condition>
```

Figure 5

The first thing you may notice is the confusing shorthand ternary operator "?true:false" at the end of the expression tag. To elucidate the meaning behind the syntax, it is as follows: If the step state of Step0 is "Complete" (true) then continue to check what is the <stateExpression> tags, else if Step0 is not complete (false) don't check what is in the <stateExpression> tags. For the programming savvy user this format can be hearkened back to some of the older programming languages, and can even be seen in some of the more recent programming languages such a Java.

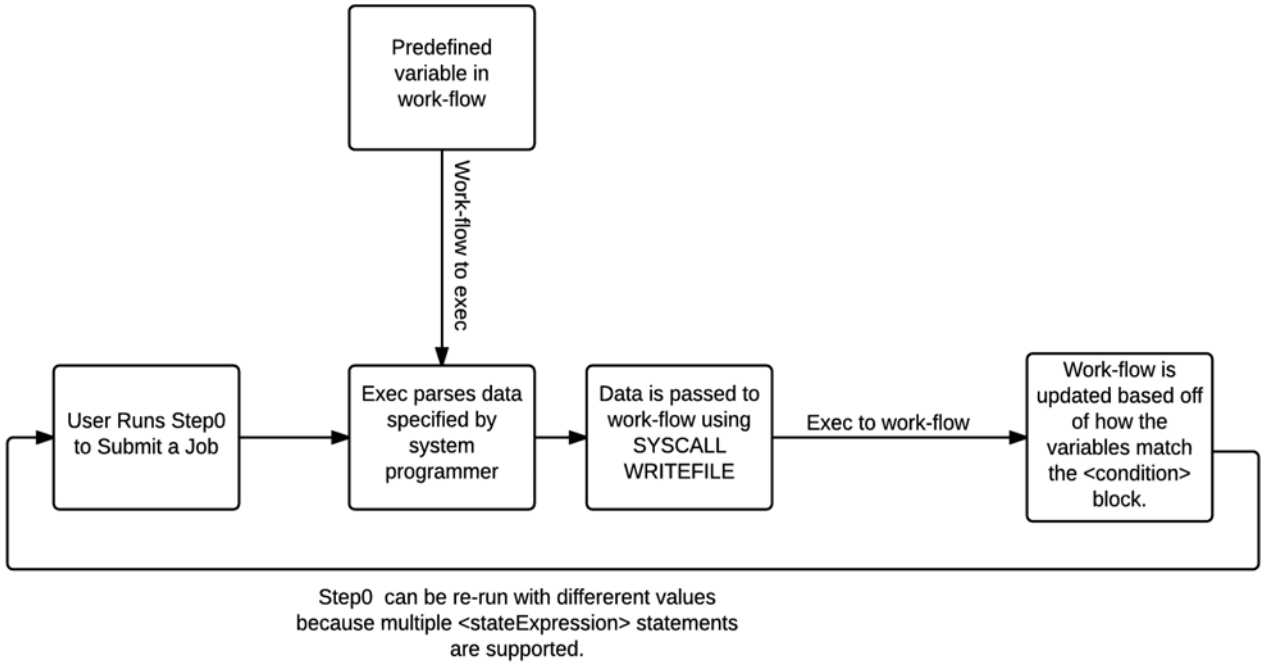
Below that in the <stateExpression> tags is what we are most interested in. After the Step0 of the work-flow is seen as completed. The work-flow will take the most recent variable value of "variable1_status" to see if its value is "Disabled", if so set state = 'Skipped' of whatever step this condition block was added to. The state can be set to Ready, Skipped, or Complete(Override).

Note: "variable1_status" can be called whatever you want, as well as its value of "Disabled" can be called whatever you want, as long as the value parsed in the REXX exec when passed back to the work-flow matches both. The "instance_" modifier is necessary at all times though.

Every time a step in a work-flow is completed z/OSMF will run through its the <condition> tags that were defined to see if there are any changes. This is why more than one <stateExpression> is supported as a step's perform tab can be run multiple times, so different data can change the state of a step to something different.

Overview

For the final part of this document an outline of what we have learned is given in simple flow chart form. This just shows how all of the sections presented are inter-related.



Conclusion

What was discussed in this document should give you a good footing on how to start manipulating a work-flow through variable passing. Again, this is just a general overview of the process and is just a baseline meant to get you past the troubleshooting stage of trying to figure out the correct syntax. What is presented here can be expanded to meet whatever requirements that you need for your project.

Some Useful Links:

I have included some other links that I believe will help answer any questions you may have regarding some other technical details of z/OSMF unrelated to variable passing.

z/OS Management Facility Programming
http://www-01.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.izua700/toc.htm

z/OS Management Facility Configuration Guide
http://www-01.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.izua300/toc.htm

Acknowledgments:

Several people were instrumental in the creation of this white paper, without them I would not have had anything to write about in the first place. They also helped provided feedback on the content presented in this paper.

- Marna Walle, STSM z/OS System Install
- Jeff Bland, Dept. Tools/SRVLIB/AUTOLINK