Policy Manager 6.1 / 7.x: Workflow Developers Reference

SOA | software™

PM_Workflow Developers Reference

PM_Workflow Developers Reference

Copyright

Trademarks

SOA Software, Inc.

SOA Software, Inc.
12100 Wilshire Blvd, Suite 1800
Los Angeles, CA 90025
(866) SOA-9876
www.soa.com
info@soa.com

Disclaimer

Table of Contents

1 Introduction

This document serves as a technical reference for developers creating and maintaining Policy Manager workflow definitions. This document contains sections that present details about all of the build-in workflow variables, conditions, and functions that have been defined for Policy Manager workflow. This document also contains details on using command-line tools to enrich the governance automation that can be enabled with Policy Manager workflow.

1.1 Notational Conventions

This document contains syntax examples for each command-line tool described. The following conventions are used:

- Words and characters in **bold face** are entered exactly as shown.
- Command elements to be substituted with user-selected values are shown in *italic-face-font*.
- Optional parameters are shown in square brackets:

  [**-option1**].

- Curly braces are used to illustrate choices to be made between alternative values:

  {*password* | *encrypted* | **?**}

- Complex sets of choices are shown by a vertical list of choices that must be used together within large square brackets:

  [**–optionA**
–optionB]

  This means you can optionally select one of the alternatives; or all alternatives within large curly braces:

  { **–optionA** *value-of-option-a*
–optionB *value-of-option-b* }

  If the choices involve multiple, related parameters, then the choices are separated by a blank line in the vertical list of alternatives.

2 Workflow Conditions

The following tables list the built-in workflow condition types that can be used with Policy Manager workflow.

Note: Policy Workflow does not include conditions.

2.1 General Conditions

These workflow conditions can be used in any type of Policy Manager workflow

Condition Type	Usage
isNull isNotNull	Tests for the presence or absence of an object. This is most useful for testing variables or XML entities. Argument *object* The value to be tested.
isEqual isNotEqual	Tests the value of an object. This is most useful for testing variables or XML entities. Arguments *operand1* *operand2* The values to be compared.
organizationHasCategory	Checks to see if an organization's UDDI definition contains a particular category and optionally checks the category value. Arguments One of the following: *organizationKey* The UDDI key of the organization. *organizationName* The organization's name. *tmodel* The UDDI tModel key of the category. *keyValue* *(optional)* The category value to test. Note: The most useful form of this condition would be to use workflow variables in the *organizationKey* argument. For instance, `${service.scope}` would test the categories of the service's organization.
runScript	Run a script to check for a special condition. Please refer to a later section that gives details of running scripts from workflow.

2.2 Authorization Conditions

These conditions are used to perform authorization tests on the user performing a workflow action.

Condition Type	Usage
authorizeByPrivilege	Tests to see if the workflow user has privileges on the object being managed by the workflow. Arguments *scope* — *(optional)* The UDDI key of the organization where privileges are to be tested. This defaults to the organization owning the workflow object. *action* — The single object privilege to test *actions* — A comma-delimited list of privileges to test - any one of which must be present.
authorizeByRole	Tests to see if the workflow user has been assigned to a role in the Policy Manager Management Console. Arguments *scope* — *(optional)* The UDDI key of the organization where role assignments are to be tested. This defaults to the organization owning the workflow object. *role* — The single role name to look for *roles* — A comma-delimited list of roles to look for - any one of which must be assigned.
authorizeByOwner	This condition passes if the workflow user is the current "owner" of the workflow instance. The "owner" is set whenever an action's result is fired.

2.3 Service Specific Conditions

These workflow conditions can only be used in the Service Workflow definition.

Condition Type	Usage
isPhysicalService isVirtualService	Returns true if the service falls into the specified category.
isProxyService	Returns true if the service is acting as a proxy to another service (set via the "Act as proxy" admin action.).
hasProxyService	Returns true if another service is acting as a proxy to this service.
isManagedService	Returns true for Physical Services that are being managed by an Agent. (set via the "Manage" admin action.).
isDiscoveredService	Returns true if the Physical Service was created by the service discovery function of certain Agents.
isPublishedService	Returns true if the service has had the "publish" workflow function applied to it. The use of "publish" is discouraged because it disabled workflow-controlled access control and its setting cannot be undone.
isInOrganization	Returns true if the service is in a specified branch of the Policy Manager organization tree. This condition is useful to perform different workflow paths for different parts of the enterprise. Arguments One of the following: *organizationKey* The UDDI key of the organization. *organizationName* The organization's name.
hasCategory	Checks to see if the service's UDDI definition contains a particular category and optionally checks the category value. Arguments *tmodel* The UDDI tModel key of the category. *keyValue* *(optional)* The category value to test.

Condition Type	Usage
proxiedServiceHasCategory	Checks to see if the proxied parent service's UDDI definition contains a particular category and optionally checks the category value. Arguments *tmodel* The UDDI tModel key of the category. *keyValue* *(optional)* The category value to test.
isCompliant	Returns true is all compliance tests run on a service completed with a status below a specified level. Also will return true if no compliance tests have ever been run for the service. Argument *complianceThreshold* The test result level considered as test failure: `warning` or `failed`.
isRunningComplianceTests	Returns true if compliance policy tests are currently executing for this service.

2.4 Contract Specific Conditions

These are the workflow conditions that are specific to Contract Workflow.

Condition Type	Usage
isAnonymousContract	Returns true if workflow is operating on an anonymous contract.

3 Workflow Variables

All conditions and functions can have a set of arguments specified by the **<arg>** elements. The contents of the **<arg>** elements are parsed for variable references in the form of ***${variable}*** which will be converted to the underlying value. The status, old-status, and owner elements in workflow **<result>** and **<unconditional-result>** are also parsed for variables to be dynamically converted. An empty string is substituted for any variable reference that is not recognized.

Variable substitution for **<arg>** elements will produce different results depending on how it is used. If the variable reference is the only content of the **<arg>** element, the actual object referenced by the variable will be substituted. If the **<arg>** elements is a mix of characters and variable references, the variable reference is replaced with the string equivalent of the variable's content by calling the `toString()` method on the variable's object.

Note: Policy Workflow does not include variables.

3.1 General Variables

These built-in workflow variables can be used anywhere in Policy Manager workflow definitions.

Variable	Content
${subject}	The *Subject* for the current caller of the workflow instance.
${caller}	Qualified name (domain\user) of the user that is calling workflow.
${owner}	Qualified name (domain\user) of the current "owner" of the workflow step. The "owner" is set whenever an action's result is fired.
${root}	The *OrganizationKey* object for the top-level (Registry) node of the organization tree.
${organization.email.contact}	Comma-delimited list of organization primary email contacts for the organization tree path from the root to the organization containing the service or contract.
${document}	The *WorkflowDocument* object for the active workflow instance. *(Internal SOA use only).*

3.2 Service Specific Variables

These built-in workflow variables are only meaningful in the Service Workflow definition.

Variable	Content
${service} ${service.key}	The *ServiceKey* object for the service the workflow is managing. This becomes the service's UDDI key when converted to a string.
${service.name}	The name of the service the workflow is managing.
${service.scope}	The *OrganizationKey* object of the organization the service belongs to. This becomes the organization's UDDI key when converted to a string.
${service.consumer} ${service.consumers}	A *Collection* of *OrganizationKey* objects for the organizations with contracts to this service.
${service.manager}	The internal ServiceManager published OSGi service. *(Internal SOA use only).*
${category.manager}	The internal *CategoryManager* published OSGi service. *(Internal SOA use only).*

3.3 Contract Specific Variables

The following built-in workflow variables can be used in Contract Workflow definitions.

Variable	Content
${contract} ${contract.key}	The *ContractKey* object for the contract the workflow is managing.
${contract.name}	The name of the contract the workflow is managing.
${contract.version}	The version number of the contract.
${contract.description}	The contract's description.

Variable	Content
${contract.summary}	A text block containing the definition of the contract.
${contract.scope}	A text block containing the providing organization along with the services and operations that are within the scope of the contract.
${contract.startdate}	The effective date/time of the contract.
${contract.enddate}	The expiration date/time of the contract.
${is.anonymous}	“true” if this is an anonymous contract. “false” otherwise.
${contract.provider}	The *OrganizationKey* object of the contract’s provider organization. This becomes the organization’s UDDI key when converted to a string.
${provider.name}	The name of the contract’s provider organization.
${provider.contact}	The name of the primary contact for the contract’s provider organization.
${provider.email.contact}	The email address of the primary contact for the contract’s provider organization.
${contract.consumer}	The *OrganizationKey* object of the contract’s consumer organization. This becomes the organization’s UDDI key when converted to a string.
${consumer.name}	The name of the contract’s consumer organization.
${consumer.contact}	The name of the primary contact for the contract’s consumer organization.
${consumer.email.contact}	The email address of the primary contact for the contract’s consumer organization.
${contract.manager}	The internal *ContractsManager* published OSGi service. *(Internal SOA use only).*

4 Workflow Functions

Policy Manager 6.1 / 7.x includes four command-line tools that are used to manipulate PKI keys, X.509 certificates, and interface with Certificate Authorities. Each section provides an overview of each command-line tool and a list of supported command-line parameters for managing the keys and certificates associated with the following:

Note: Policy Workflow does not include functions.

4.1 General Functions

These workflow functions can be used in any type of Policy Manager workflow.

Function Type	Usage
print	Prints out the name/value pairs contained in the **<arg>** elements to stdout along with a timestamp.
log	Sends a message to the Policy Manager log. Arguments *message* — The message to be logged along with the object the workflow is managing. *severity* — *(optional)* The logging severity for the message: **fatal**, **error**, **warning**, **info** (default), **debug**, or **trace**.
runScript	Run a script to perform a special function. Please refer to a later section that gives details of running scripts from workflow.
getOrganizationemailContact	Produces a comma delimited list of organization contact email addresses. Arguments *organization* — *(optional)* The UDDI key of the organization whose email contacts are to be retrieved. Defaults to the parent organization of the service or contract. *contactType* — A comma-delimited list of organization contact "use types" that are to be selected. *emailType* — *(optional)* A comma-delimited list of organization contact email "use types" that are to be selected. *variable* — *(optional)* The name of the workflow variable

Function Type	Usage
	that is to hold the resulting list of email addresses. This name must not contain a period. The default is to use the first value in the *contactType* argument.
email	Generates and sends an email message. Arguments *smtpHost* *(optional)* The host name of the SMTP server. Defaults to the global mail server configured in Policy Manager. *to* A comma-delimited list of recipients. *from* The value to be used as the email sender. *cc* *(optional)* A comma-delimited list of copy recipients. *subject* The subject line of the message. *message* The body of the message. *priority* *(optional)* The priority of the message: **`high`**, **`medium`**, **`low`**. Default is **`medium`**. *parseVariables* Should ${…} variables be resolved in the following arguments: *to*, *cc*, *subject*, *message.*

4.2 Service Specific Functions

These workflow functions are only useful in the Service Workflow definition

Function Type	Usage
addCategory	Adds a new “keyed reference” entry in the service’s UDDI category bag. Arguments *tmodel* The UDDI tModel key of the new keyed reference. *keyName* *(optional)* The name of the new keyed reference. *keyValue* The value to be placed in the new keyed reference.

Function Type	Usage
deleteCategory	Removes a particular keyed reference from the service's UDDI category bag. Arguments *tmodel* The UDDI tModel key of the keyed reference to be removed. *keyValue* The key value of the keyed reference to be removed.
setUniqueCategory	Adds a new "keyed reference" entry in the service's UDDI category bag after deleting any entries with the same tModel key. Arguments *tmodel* The UDDI tModel key of the new keyed reference. *keyName* *(optional)* The name of the new keyed reference. *keyValue* The value to be placed in the new keyed reference.
setLifecycleStage	Sets or updates the *lifecycle stage* value in the service's category bag. This value appears in the Service Overview section of the Service Details page. Arguments *stage* The value of the new service lifecycle stage.
publish	Sets the "publish" flag for the service. ***Note:*** SOA does not recommend ever using this function because the only thing it does is disable the workflow-based service access control. Also, once this flag is set, there is no way to turn it back off to restore workflow access control.
getCategoryValue	Placed the key value of a keyed reference selected from the service's category bag into a workflow variable. Arguments *tmodel* The UDDI tModel key of the keyed reference. *keyName* *(optional)* The name of the keyed reference. *var* The name of the workflow variable to receive the key value of the selected keyed reference. Do not include any periods in this name.

Function Type	Usage
getCategoryValueFromProxiedService	Placed the key value of a keyed reference selected from the proxy parent service's category bag into a workflow variable. Arguments *tmodel* The UDDI tModel key of the keyed reference. *keyName* *(optional)* The name of the keyed reference. *var* The name of the workflow variable to receive the key value of the selected keyed reference. Do not include any periods in this name. *handle.errors* [`lax` | `strict`] Generate warnings or errors when used on a non-proxy service.
copyCategoriesToProxyServices copyCategoriesFromProxiedService	Copy selected keyed reference from the service's category bag to the category bag of its proxy virtual service. Arguments *include* A list of tModel keys of the categories to be included. The default is to include all non-reserved categories. *include.groups* A list of tModel keys of the category groups to be included. *exclude* A list of tModel keys of the categories to be excluded. *exclude.groups* A list of tModel keys of the category groups to be excluded. *replace* A list of tModel keys of the categories to be completely replaced in the proxy service. *replace.groups* A list of tModel keys of the category groups to be replaced in the proxy service. *recursive* *(copyToProxyServices only)* Should the selected categories be copied to the entire chain of services (`true`) or to just the direct proxies (`false`=default). These arguments are whitespace-delimited lists of UDDI keys. The entries can contain a wildcard (*) at either the beginning, the end, or both.

Function Type	Usage
exportService	Exports the service as a ZIP archive in a designated location on the Policy Manager server. Argument One of the following: *export.file* The fully qualified name of the export ZIP. *export.folder* The directory where the export ZIP will be placed. The default is *[SOA-HOME]*`/sm60/export`. The file will be named: `service-`*[service-uddi-key]*`-export.zip` *include.artifacts* *include.operational.policies* *include.qos.policies* *include.compliance.policies* *include.pki.keys* specifies what is to be included in the service export. The default is to include everything If the export file already exists, it will be kept but renamed to include a date/time stamp.
exportProxyServiceChain	Exports a Proxy Virtual Service along with its entire chain of proxied parent services. This function has the same set of arguments as the *exportService* function. The default name of the export ZIP archive will be: `servicechain-`*[service-uddi-key]*`-export.zip`
performActionOnProxyServices performActionOnProxiedService	Perform a workflow action on the related proxy or proxied parent service Arguments One of the following: *action.id* The **`id`** of the workflow **`<action>`** to be performed. *action.name* The **`name`** of the workflow **`<action>`** to be performed. *action.message* An optional comment that will be logged with the workflow history of the action event. *handle.errors* [lax | strict] Generate warnings or errors when the specified action is not part of the target service's current workflow **`<step>`**. *recursive* *(performActionOnProxyServices only)* Should the action be performed on the entire chain of services (`true`) or to just the direct proxies (`false`=default). Additional arguments can be passed and will be available as workflow variables for use in the target action. The names of these additional arguments should not contain any periods (.).

Function Type	Usage
performAction	Perform a workflow action on the service with the same key but in a remote Policy Manager instance. Arguments One of the following: *action.id* The **id** of the workflow **<action>** to be performed. *action.name* The **name** of the workflow **<action>** to be performed. *action.message* An optional comment that will be logged with the workflow history of the action event. *handle.errors* [lax | strict] Generate warnings or errors when the specified action is not part of the target service's current workflow <step>. One of the following: *workflow.service.key* Specify the instance of the *workflow.service.qname* Workflow Service for the remote *workflow.service.binding.identifier* Policy Manager *workflow.service.username* The user on the remote Policy Manager to use to perform the action. This is either a qualified user name (*domain\user*) or a user in the *Local Domain.* *workflow.service.password* The password for that user. *remote.transport.method* The strategy to be used to transmit the perform-action request to the remote Policy Manager: ***sync*** *(default)*Function does not complete until the action has been completed on the remote Policy Manager. ***async*** Function completes immediately and the perform-action request is placed on a persistent queue to be processed by a background thread. Additional arguments can be passed and will be available as workflow variables for use in the target action. The names of these additional arguments should not contain any periods (.)

4.3 Contract Specific Functions

The following workflow functions can be used in Contract Workflow definitions.

Function Type	Usage
version	Moves the contract from "*Draft"* to "*Active and Activated"* state.
deactivate	Moves the "*Active and Activated"* contract to the "*Active but Deactivated"* state.
activate	Moves the "*Active but Deactivated"* contract to the "*Active and Activated"* state.
exportContract	Exports the contract as a ZIP archive in a designated location on the Policy Manager server. Argument One of the following: *export.file* The fully qualified name of the export ZIP. *export.folder* The directory where the export ZIP will be placed. The default is *[SOA-HOME]*`/sm60/export`. The file will be named: `contract-`*[contract-key]*`-export.zip` *include.artifacts* Export attached metadata (default=true). *include.qos.policies* Export any attached QoS policies (default=true). If the export file already exists, it will be kept but renamed to include a date/time stamp.
performAction	Perform a workflow action on the contract with the same key but in a remote Policy Manager instance. Arguments One of the following: *action.id* The **`id`** of the workflow **`<action>`** to be performed. *action.name* The **`name`** of the workflow **`<action>`** to be performed. *action.message* An optional comment that will be logged with the workflow history of the action event. *handle.errors* [`lax` | `strict`] Generate warnings or errors when the specified action is not part of the target contract's current workflow <step>. One of the following:

Function Type	Usage
	workflow.service.key Specify the instance of the *workflow.service.qname* Workflow Service for the remote *workflow.service.binding.identifier* Policy Manager *workflow.service.username* The user on the remote Policy Manager to use to perform the action. This is either a qualified user name (*domain\user*) or a user in the *Local Domain.* *workflow.service.password* The password for that user. *remote.transport.method* The strategy to be used to transmit the perform-action request to the remote Policy Manager: ***sync*** *(default)*Function does not complete until the action has been completed on the remote Policy Manager. ***async*** Function completes immediately and the perform-action request is placed on a persistent queue to be processed by a background thread. Additional arguments can be passed and will be available as workflow variables for use in the target action. The names of these additional arguments should not contain any periods (.)

5 Using Scripts in Workflows

Scripting can be used to provide customer-specific processing for workflow <condition> and <function> elements of service and contract workflow. Scripts can be embedded in the workflow XML definition or they can be referenced from the workflow to a file system location. Workflow scripting currently supports Beanshell and Jython scripts for PM61 and JavaScript for PM7x.

The following Workflow function can be used to invoke a script to perform any special processing that is required. This same structure is used to invoke a script in a `<condition>` workflow element. In this case, the script must return a boolean value.

```
<function  type="runScript">
      <arg  name="lang"> beanshell | jython </arg>

      { <arg name="file">script-file-path</arg>

        <arg name="script">
            body of the script
        </arg> }

      [<arg name="var-name">var-value</arg>] ...
</function>
```

Supported Arguments:

<arg name="lang"> beanshell | jython </arg>

Specify one of these values to select the language of the script. Beanshell (http://www.beanshell.org/) or Jython (http://www.jython.org/) are supported for PM61 and JavaScript for PM7x (https://developer.mozilla.org/en-US/docs/Rhino).

<arg name="file">*script-file-path***</arg>**
<arg name="script"> *body of the script* **</arg>**

The script to be executed can be either read from a file or included directly within the workflow definition XML file. Use one of these two arguments to specify the script body or its file system location accessible to the Policy Manager containers.

<arg name="*var-name***">***var-value***</arg> ...**

You can specify any number of additional arguments that will be passed into the script as native script variables.

5.1 Build-In Script Variables

The following native script variables related to the workflow context are assigned before the script begins execution.

Variable	Content
callerSubject	The *Subject* for the current caller of the workflow action. This contains the actual Java Subject object while the **${caller}** workflow variable will only contain the user name of the current caller.
transientVars	A set of name/value pairs that can be referenced as workflow variables. This is a **Map<String,Object>** Java object. The information contained in this variable only exists for the current action execution (and any triggered automatic actions.)
propertySet	This is a collection of named data elements that can be accessed as workflow variables. The information contained in this variable is persisted throughout the lifetime of the workflow instance. This is an *Open Symphony* PropertySet Java object.
serviceKey	The unique key of the service. *(Defined for service workflows only.).*
contractKey	The unique key of the contract. *(Defined for contract workflows only.).*
others	Any additional **<arg>** elements present in the scripted function or condition definition will be assigned to script variables. For instance, if the script definition contains an element: `<arg name="myPhone">555-1212</arg>` then a script variable named “myPhone” will be defined and assigned the script value of “555-1212”.

6 Command-Line Tools for Workflow

Policy Manager provides a command-line tool that allows integrating Policy Manager workflow with external processing. This tool can be used to perform workflow actions on services and contracts. The tool also provides a facility to update the workflow definition XML without using the Policy Manager Management Console.

6.1 Performing Workflow Actions

The `callWorkflow` command-line tool can be used in external processing to cause a Workflow action to be performed on a particular service or contract.

callWorkflow{**.sh**|**.bat**}

(**--performAction**

(**--performRemoteAction**
{ **--remoteServiceKey** *uddi-service-key*
--remoteServiceQName *service-qname*
--remoteServiceBindingId *binding-identifier* }
--remoteAdminUser [*domain*\]*username*
--remotePassword *password*
[**--remoteMethod** **sync** | **async**]))

{ **--service** *uddi-service-key*
--serviceKey *uddi-service-key*
--serviceQName *service-qname*
--serviceBindingId *binding-identifier*
--contract *contract-key* }

--action *workflow-action-name*
[**--comment** *"comment for workflow history log"*]

--user [*domain*\]*username*
--password *password*

[**--host** *PM-host-name*]
[**--port** *PM-port*]

Supported Command-Line Parameters:

––performAction
––performRemoteAction

Use one of these required parameters to specify whether the workflow action is to be performed on a service or contract in the local Policy Manager or on a remote Policy Manager instance.

––remoteServiceKey *uddi-service-key*
––remoteServiceQName *service-qname*
––remoteServiceBindingId *binding-identifier*

Use one of these when performing an action on a remote Policy Manager to specify the instance of the Policy Manager Workflow Service that has been created in the local Policy Manager connected to the desired remote Policy Manager. You can specify either the service's UDDI key, QName (*{xml-namespace}xml-localpart*), or a binding identifier that has been assigned to the service.

––remoteAdminUser [*domain*\]*username*
––remotePassword *password*

When performing an action on a remote Policy Manager, these two parameters specify the user credentials on the remote Policy Manager instance when performing the specified action.

––remoteMethod sync | async

This optional parameter specifies the strategy to use when performing a workflow action on a remote Policy Manager instance.

sync The command will not complete until the workflow action has been completed on the remote Policy Manager instance. *This is the default.*

async The command completes immediately after the request to perform the remote workflow action has been placed on a persistent queue. Policy Manager includes a background process that will perform the workflow action on the remote Policy Manager instance as soon as possible.

--service *uddi-service-key*
--serviceKey *uddi-service-key*
--serviceQName *service-qname*
--serviceBindingId *binding-identifier*
--contract *contract-key*

Use one of these required parameters to specify the target service or contract for the Workflow action to be performed.

--action *workflow-action-name*

Use this required parameter to specify the Workflow action that is to be performed for the selected service or contract. This parameter specifies the name attribute on the Workflow <action> element. For instance:

```
<action id="1000" name="WF-Import Complete">
```

This action could be selected using the following parameter:

```
--action "WF-Import Complete"
```

Note that if the value for this parameter includes embedded spaces, it must be enclosed in quotation marks.

--comment "*comment for workflow history log*"

This optional parameter is used to specify a comment that will be added to the Workflow history log entry for the action performed. If the value for this parameter includes embedded spaces, it must be enclosed in quotation marks.

--user [*domain*\]*username*
--password *password*

These required parameters specify the user name and password for a user on the local Policy Manager that will process this request. This user must have permission to use the Workflow Service on the local Policy Manager instance. In addition, if the workflow action is to be performed on a service or contract in the local Policy Manager, this user must also have the permissions and/or roles needed to perform that action.

--host *PM-host-name*
--port *PM-port*

These optional parameters specify hostname and port for the local Policy Manager container. These default to `localhost` and `9900`.

6.2 Loading Updated Workflow XML Definitions

The `callWorkflow` command-line tool can also be used to load updated workflow XML definitions in the local Policy Manager instance using a scripted process. This process must be performed from the command-line on the server a Policy Manager container is executing.

callWorkflow{.sh|.bat}

––updateDefinition

{ **––serviceWorkflowFile** *file-system-path*
––contractWorkflowFile *file-system-path* }

[**––reset** | **––noreset**]

––user [*domain*\]*username*
––password *password*

[**––host** *PM-host-name*]
[**––port** *PM-port*]

Supported Command-Line Parameters:

–– updateDefinition

This required parameters is needed to select the Update Workflow Definition processing.

––serviceWorkflowFile *file-system-path*
––contractWorkflowFile *file-system-path*

Use one of these to specify the file system location of the updated workflow definition XML file. This XML file must be specified as a file system location that is accessible by the Policy manager container.

––reset | **––noreset**

When **––reset** is specified, all of the current service or contract workflow instances are reinitialized by performing the **@reset** initial action. This is

needed when there is the possibility that one of the workflow instances would have an active **<step>** that is not present in the new workflow definition.

--user [*domain*\]*username*

--password *password*

These required parameters specify the user name and password for a user on the local Policy Manager that will process this request. This user must have permission to use the Workflow Service on the local Policy Manager instance. In addition, if the workflow action is to be performed on a service or contract in the local Policy Manager, this user must also have the permissions and/or roles needed to perform that action.

--host *PM-host-name*

--port *PM-port*

These optional parameters specify hostname and port for the local Policy Manager container. These default to `localhost` and `9900`.