Community Manager®: Custom Workflows

SOA | software™

Community Manager

Custom Workflows
Version 7.2
October, 2014

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

Contents

Chapter 1 | Community Manager Custom Workflows: Overview

Workflow defines the sequence of steps that are followed in a business process, including such related data as conditions (for example, a ticket must be resolved before it can be closed), state (for example, a ticket can have states of Open, Resolved, and Closed), or role (for example, a certain step can only be completed by an Administrator).

Defining the workflow for a business process gives you control over the process and allows you to monitor and customize as needed to streamline the business process.

Community Manager includes default out-of-the-box workflows for certain resources, such as API contracts, and allows you to customize the workflow for several key resources. For example, you could build a workflow that customizes the platform's Export functionality so that at the click of a button an authorized user could export platform data to a designated folder, to be picked up by a corresponding Import function in a custom workflow in another build. You could then use this custom workflow feature to automate the transfer of data from a development environment to a QA environment.

This document provides information about the resources that follow default workflows, the resources for which you can define custom workflows, and the process you'll need to follow to design, create, test, and implement a custom workflow.

It includes the following main sections:

- **Custom Workflows**—the workflows you can customize, how to get started, and how to implement a custom workflow on the platform.
- **Workflow Reference: Functions, Conditions, and Variables**—the technical details you'll need to develop a custom workflow for the platform.

Note: Community Manager uses OSWorkflow v2.8.0 from OpenSymphony. OSWorkflow is an open-source workflow engine written in Java.

Custom Workflows

The ability to customize the platform default workflow for resources gives you great flexibility. For example, you could customize the ticket workflow so that each new ticket for your API in the platform automatically triggers an email to create a ticket in your own internal trouble ticketing system.

This section provides a high-level overview of custom workflow design, development, and implementation, including:

- Workflows You Can Customize
- Steps for Implementing a Custom Workflow
- Uploading a New Custom Workflow
- Testing a New Custom Workflow

- Implementing a New Custom Workflow
- Updating Existing Resources
- Updating User Documentation

Workflows You Can Customize

The platform supports implementation of custom workflows for the following types of resources:

- API
- App
- API Contract
- Group Membership
- Ticket

Steps for Implementing a Custom Workflow

At a high level, the steps to implement a custom workflow are listed below. The following sections provide more information on each of these steps. In some cases, certain actions must be performed by users with a specific role. Roles are shown for each step.

1 Create the custom workflow outside the platform. For more information and technical details, refer to *Developing a Custom Workflow* on page 9. (**Admin or delegate**)

2 Upload the custom workflow to the platform. See *Uploading a New Custom* Workflow below. (**Site Admin only**)

3 Assign the custom workflow to one or more resources. (**Site Admin**)

4 Test the new custom workflow on one or more individual resources, and resolve any issues as needed until you are sure the new workflow is fully functional and bug-free. For more information, see *Testing a New Custom Workflow* on page 8. (**Site Admin, Business Admin, or resource admin**)

5 Implement the new custom workflow as the default for new resources of that type. See *Implementing a New Custom Workflow* on page 8. (**Site Admin**)

6 If needed, update existing resources to use the new workflow, on a case-by-case basis. See *Updating Existing Resources* on page 9. (**Site Admin or Business Admin**)

7 Update user documentation. The platform's user documentation reflects the default workflow. If you change the workflow, it might render the platform documentation incomplete, incorrect, or both. You must make sure you update the documentation to reflect any changes from the default workflow. (**API Admin or delegate**)

Uploading a New Custom Workflow

Uploading or managing workflow is a Site Admin activity.

To upload a custom workflow, log in as the Site Admin and go to Administration > Workflows. Here you can easily manage custom workflows, including adding, editing, viewing, or deleting.

For additional instruction, if needed, refer to the platform online help.

Testing a New Custom Workflow

Changing the custom workflow for resources on the platform can significantly impact the user experience for all users on the platform. It's very important to test a new workflow thoroughly, making sure it works as planned, before implementing it as the default.

One strategy is to write out various use cases that will be affected by the workflow change, apply the new workflow to one or two custom resources, and then test those resources, using the use cases as a guide. Make sure all state changes and results are as expected.

If you encounter any issues, restore the default workflow and then delete the test workflow from the platform. When corrections are made, upload the corrected version and test again.

Make sure you are completely satisfied that the new workflow runs as expected before making it the default for new resources of the applicable type.

Below is an example of a testing strategy for a new API version workflow.

Example: to validate and test a new API version workflow

1 Load the workflow into Community Manager but do not set it as the default for APIs.
2 Create a test API.
3 Specify the new workflow as the default for the test API.
4 Use the API and verify:
   - The sequence of state transitions is correct.
   - The actions displayed are correct for the active state.
   - The workflow history is correct (including status).
5 Depending on the test results:
   - If the workflow needs more work, remove it so that it isn't available for selection in the platform.
   - When you're completely satisfied that the workflow is functioning as intended, set it as the default, if applicable. For more information, refer to *Implementing a New Custom Workflow* below.

Implementing a New Custom Workflow

Once a new custom workflow has been fully tested and you're sure it's working as expected, you can upload it to the platform and set it as the default for the resource type. Once you do this, all new resources of that type will use the new workflow.

This action can only be performed by the Site Admin.

Note that existing resources are not affected. For example, if you change the API version workflow, and have existing APIs, they will still use the workflow that was in use when the API was created. If needed, you can change the workflow for an existing API or app on a case by case basis.

- **To upload a custom workflow:** Log in as the Site Admin, go to Administration > Workflows, and then click **Add Workflow**.

- **To implement a custom workflow for Apps, APIs, API contracts, group membership, or tickets:** Log in as the Site Admin, go to Administration > Config, and then choose the resource. Update the default workflow field, choosing the new workflow from the drop-down list, and then save your changes.

  For additional instruction, if needed, refer to the platform online help.

Updating Existing Resources

You can update an existing app or API to use a new custom workflow if you are an Administrator for the resource.

The platform provides REST APIs to update an existing API to use a new workflow. At this point the user interface doesn't include a feature to update an existing API to use a new workflow; however, you can accomplish this using the platform API. There are operations to accomplish the following:

- Return information about the workflow governing a specified app version.
- Update the workflow governing a specified app version.
- Return information about the workflow governing a specified API version.
- Update the workflow governing a specified API version.

Updating User Documentation

It is the site administrator's responsibility to update the developer documentation to reflect any custom workflow changes.

When developers are using the platform, they are not aware that the sequence of actions in specific processes is governed by workflow; they only know that they must follow those processes to get the desired results.

The online help for the platform guides developers through step-by-step instructions to complete various activities associated with managing apps, API contracts, tickets, and so on.

If you change the process, you have the responsibility to also update the applicable online help content to give developers the additional information they need to comfortably use the platform. App developers are your target audience, and their user experience is very important. For example, if you implement a custom workflow so that an app cannot have an active contract in the production environment with one API until it's ready to run in the production environment with *all* the APIs it's using, you must let developers know that.

Developing a Custom Workflow

The workflow process is defined in an XML file that contains:

- Initial actions (actions that start the workflow)
- Workflow steps and actions
- Other specialized elements

The basic structure of required elements in a workflow XML document is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE workflow PUBLIC
   "-//OpenSymphony Group//DTD OSWorkflow 2.9//EN"
   "http://www.opensymphony.com/osworkflow/workflow_2_9.dtd">
<workflow>
   <initial-actions>
      One or more <action> elements
   </initial-actions>
   <steps>
      One or more <step> elements
   </steps>
</workflow>
```

Workflow Initial Actions

The first part of the workflow includes definitions of one or more initial actions that trigger it.

Initial actions are predefined reserved words, such as @Create. Initial actions bring the resource into the workflow at the beginning, whether the workflow in use is a default or custom workflow.

The specific initial actions for each type of workflow are given in the workflow reference section of this document.

The example below shows the initial action @Create used to start the App Version workflow.

```
  <initial-actions>
    <action id="1" name="@Create">
      <results>
        <unconditional-result old-status="Received" status="Setup" step="100" owner="${caller}" />
      </results>
    </action>
  </initial-actions>
```

Workflow Reserved Actions

There are specific actions for each workflow type that are used internally as part of the workflow. These are called reserved actions. Reserved actions are predefined for certain actions that might commonly be done to objects when something else happens in the system, as distinct from actions that occur as a result of a user's action such as clicking a button. The reserved action is triggered automatically under certain conditions as defined in the workflow.

The specific initial actions for each type of workflow are given in the workflow reference section of this document.

Within the context of workflow, there are two main purposes for reserved actions:

1 To specify conditions.

   The reserved action is used to verify whether an action is allowed or not, in the current context, before executing. For example, you could specify that an app can have connections in the Production environment only, or in the Sandbox environment only. Reserved actions allow you to define configurable behavior.

2 The reserved action is invoked internally when something else happens.

For example, if a group is deleted, a reserved action is invoked, and then executes one or more steps to determine what happens with the group members (for example, each is sent a notification). In this scenario, the user doesn't see any buttons or make any choices; when the reserved action is triggered, it executes whatever steps are coded into the workflow. You could use a reserved action to determine that when an API is deleted, all existing contracts relating to that API are deleted and a notification is sent out to all users affected by the change. Similarly, if an app is deleted, you could use a reserved action to determine what happens to API contracts associated with the app.

There are specific reserved actions for each type of workflow.

This section includes:

- Reserved Actions: Naming Conventions
- Reserved Actions: Examples

Reserved Actions: Naming Conventions

The platform uses two main formats for naming of reserved actions:

- Action name begins with "**reserved-**"; for example, under certain conditions the "reserved-connect" reserved action is sent to the user interface so that the user interface will display a Connect button. If we use this prefix on a reserved action, the UI uses it to customize the behavior, but doesn't show it as an action the user can invoke. It's internally invoked, or the purpose of it is to check whether an action is valid or not. Nobody invokes the reserved action. It indicates, in the current state, whether the action can be performed or not. It is used to customize the behavior, but not to customize the list of actions that the user can invoke. Example:
  - reserved-connect-from-app.Sandbox
- Action name begins with **@** sign. Internal actions that the user doesn't need to know about, or initialization actions that the product uses for adding an object; for example, when adding a contract or an app, the initial state that the resource is in. Actions that start with an @ sign are never part of the return list of actions when the workflow API is used to get a list of workflow actions that are valid for a resource. They are only used internally. Examples:
  - @app_switch_to_production
  - @Invite
  - @Revise (to revise a contract that is in Activated or Suspended state)
  - com.soa.apicontract.revise (valid when contract status is **com.soa.apicontract.inforce** and there is no corresponding contract with status **com.soa.apicontract.draft**.

Reserved Actions: Examples

Below are some examples of reserved actions from actual workflow documents.

```
<action id="59" name="@modify">

<action id="58" name="@read">

<action id="1000" name="Activate Contract">
```

```
<action id="1500" name="WF-Import Complete">
<action id="2000" name="Export to Clone" auto="TRUE">
<action id="3000" name="WF-Contract Revoked">
<action id="3999" name="Workflow has ended">
<action id="9000" name="Revoke in Clone" auto="TRUE">
```

Workflow Steps and Actions

The workflow definition file contains one or more steps. Each step has a unique ID and a name.

Each step can contain one or more Action elements, but they are not required.

Actions can be automatic or manual.

The basic structure of an Action element is shown below.

```
<action id="###" name="Display name of action">
   <results>
      Optional conditional <result> elements
      </unconditional-result old-status="value" status="value" step="###" />
   </results>
</action>
```

Note: A result step value of -1 causes no workflow state transition.

Developing a Custom Workflow: Steps to Consider

In developing your custom workflow, here are some planning steps to consider:

- Choose the tool you will use to create and modify the XML file.
- Decide what strategy you will follow to adopt unique numbers for each Step ID and Action ID.
- Determine how the workflow will terminate.
- Decide how you will validate and test the workflow.

Chapter 2 | Workflow Reference: Functions, Conditions, and Variables

This section serves as a technical reference for developers creating and maintaining workflow definitions for use with the platform. It includes details about the built-in workflow variables, conditions, and functions:

- For general use with any platform workflow
- For each customizable workflow

Functions, Conditions, and Variables: General Information

The functions, conditions, and variables work together and are the building blocks of your custom workflow.

Each workflow type has specific functions, conditions, and variables that only work within that type of workflow. In addition, there are general conditions that can be used in any CM custom workflow.

Note: The general use functions, conditions, and variables described in this section are only for use with Community Manager custom workflows. There is also a set of general use functions, conditions and variables for use with SOA Software Policy Manager workflows. Those are also valid for Community Manager workflows. You will see some of them in use in the examples in this document.

General Use: Functions

Functions are used to add behavior and automation to a workflow.

Below is a generic example of how a function is used in a workflow, and how its arguments are represented:

```
function type="functionName">
   <arg name="argName1">arg name 1</arg>
   <arg name="keyName">Key name</arg>
   <arg name="keyValue">Key value</arg>
</function>
```

There are no general use functions for workflow in the Community Manager platform.

General Use: Conditions

Conditions allow you to:

- Select the result of an action
- Restrict the availability of an action by:

- Restricting actions: to restrict when an action can be seen or performed.
- Restricting access: to control service and contract access.

Workflow uses conditions as logical expressions. Conditional functions can take arguments:

```
<arg>. . .</arg>
```

For example:

```
<conditions type="AND">
   <condition type="hasCategory">
      <arg name="tmodel">uddi:acmecorp.com:hasbeenpromoted</arg>
      <arg name="keyvalue">true</arg>
   </condition>
</conditions>
```

Some additional points to note about conditions and how you can use them in custom workflows:

- Conditions can also include nested conditions. You can use a structure of nested conditions to form a complex logical expression.
- You can code a logical NOT as a negative condition: <condition ... negate="TRUE">.

There is one general use condition available in developing custom workflows:

- authorizeByAtmosphereRole

authorizeByAtmosphereRole

Tests to see if the workflow user has been assigned one or more specified roles in the platform; returns Boolean true or false.

Arguments

Name	Description/Values
Role	One or more roles that are authorized to perform the function. Valid values: • ApiAdmin • ApiInvitedUser • AppAdmin • SiteAdmin • BusinessAdmin

Examples/Notes/Additional Information

In the example below, the function clones all API contracts from the Sandbox environment to the Production environment. The <condition> tag uses the **authorizeByAtmosphereRole** condition to specify that the user attempting to perform this action must be an App Admin (a member of the app team).

```
<action id="210" name="Submit For Review">
```

```
  <restrict-to>
    <conditions type="AND">
      <condition type="authorizeByAtmosphereRole">
        <arg name="role">AppAdmin</arg>
      </condition>
    </conditions>
  </restrict-to>
  <pre-functions>
    <function type="cloneAllAPIContracts">
      <arg name="EnvFrom">Sandbox</arg>
      <arg name="EnvTo">Production</arg>
    </function>
  </pre-functions>
  <results>
    <unconditional-result old-status="Sandbox" status="Review" step="300" owner="${caller}" />
  </results>
</action>
```

General Use: Variables

There are no general use variables for workflow in the Community Manager platform.

App Version Workflow

This section provides information about functions, conditions, and variables for the app version workflow, as well as initial actions and reserved actions.

App Version Workflow: Initial Actions

There is only one initial action valid for Community Manager workflows relating to apps/app versions, as shown below:

- @Create

App Version Workflow: Reserved Actions

The following reserved actions are defined for app version workflows:

- reserved-connect-to-api
- reserved-connect-to-api.Sandbox
- reserved-connect-to-api.Production
- reserved-approve-api-connection.Production
- reserved-cancel-api-connection.Sandbox
- reserved-cancel-api-connection

App Version Workflow: Functions

The following functions are available for the app version workflow:

cloneAllAPIContracts

Sets up connections in the target environment with all the API versions that the app version is connected to in the source environment.

Parameters

Name	Description/Values
EnvFrom	Source environment: The environment contracts are being cloned from (Sandbox or Production).
EnvTo	Target environment: The environment contracts are being cloned to (Sandbox or Production).

Examples/Notes/Additional Information

In the example below, this function is used to clone all API contracts from Sandbox to Production.

```
<pre-functions>
  <function type="cloneAllAPIContracts">
    <arg name="EnvFrom">Sandbox</arg>
    <arg name="EnvTo">Production</arg>
  </function>
</pre-functions>
```

activateAllAPIContractsInEnvironment

Activates all of an app's connections in the specified environment. For example, for an app with five contracts in the production environment, this function would activate all of them.

Parameters

Name	Description/Values
Environment	The environment in which all API contracts are being activated (Sandbox or Production).

Examples/Notes/Additional Information

In the example below, this function is used to activate all API contracts in the Production environment.

```
<pre-functions>
  <function type="activateAllAPIContractsInEnvironment">
```

```
    <arg name="Environment">Production</arg>
  </function>
  <function type="cancelAllAPIContractsInEnvironment">
    <arg name="Environment">Sandbox</arg>
  </function>
</pre-functions>
```

cancelAllAPIContractsInEnvironment

Cancels all of an app's connections in the specified environment.

Parameters

Name	Description/Values
Environment	The environment in which all API contracts are being cancelled (Sandbox or Production).

Examples/Notes/Additional Information

The example below shows the use of activateAllAPIContractsInEnvironment followed by cancelAllAPIContractsInEnvironment to activate all contracts in the Production environment and then cancel all contracts in the Sandbox environment.

```
<restrict-to>
  <conditions type="AND">
    <condition type="authorizeByAtmosphereRole">
      <arg name="role">AppAdmin</arg>
    </condition>
  </conditions>
</restrict-to>
<pre-functions>
  <function type="activateAllAPIContractsInEnvironment">
    <arg name="Environment">Production</arg>
  </function>
  <function type="cancelAllAPIContractsInEnvironment">
    <arg name="Environment">Sandbox</arg>
  </function>
</pre-functions>
```

App Version Workflow: Conditions

The following conditions apply to the app version workflow:

isAppTeamMemberUserLeaderOfAnyOtherGroup

Tests to see if the user is an app team member and is also a leader of at least another independent group. Returns true if the logged-in user is a leader of any independent group.

Parameters

None.

atleastOneValidAPIContractInEnvironment

Checks that there is at least one contract for the specified app version with the specified API in the specified environment. Returns true if the app is connected to at least one API version in the specified environment (states of Cancelled or ApiDeleted are not valid).

Parameters

Name	Description/Values
Environment	The environment being tested to see if there is a valid contract. For this function, the value will always be **Sandbox**.

Examples/Notes/Additional Information

In the example below, the action being performed is to cancel an API contract in the Sandbox environment. The workflow first checks that there is at least one valid API contract in the Sandbox environment.

```
<action id="202" name="reserved-cancel-api-connection.Sandbox">
  <results>
    <result old-status="Sandbox" status="Setup" step="100" owner="${caller}">
      <conditions type="AND">
<condition negate="true" type="atleastOneValidAPIContractInEnvironment">
  <arg name="Environment">Sandbox</arg>
</condition>
      </conditions>
    </result>
    <unconditional-result old-status="Sandbox" status="Sandbox" step="200" owner="${caller}" />
  </results>
</action>
```

allAPIContractsInEnvironmentApproved

Checks whether all API contracts in the specified environment are approved. Returns true if all connections in the specified environment are approved.

Arguments {mod}

Name	Description/Values
Environment	The environment for which all API contracts are being approved (Sandbox or Production).

Examples/Notes/Additional Information

In the example below, the action being performed is to approve all API contracts in the Production environment.

```
<action id="301" name="reserved-approve-api-connection.Production">
  <restrict-to>
    <conditions type="AND">
      <condition type="existAPIContractsForAllAPIsInEnvironments">
<arg name="EnvFrom">Sandbox</arg>
<arg name="EnvTo">Production</arg>
      </condition>
      <condition type="allAPIContractsInEnvironmentApproved">
<arg name="Environment">Production</arg>
      </condition>
      <condition type="authorizeByAtmosphereRole">
<arg name="role">BusinessAdmin</arg>
      </condition>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Review" status="Approved" step="400" owner="${caller}" />
  </results>
</action>
```

existAPIContractsForAllAPIsInEnvironments

Checks whether for each contract that exists in one API environment (sandbox or production) there is a corresponding contract for the other environment. Returns true if the app version is connected to the same API versions in the “EnvTo” environment as in the “EnvFrom” environment.

Arguments

Name	Description/Values
EnvFrom	The environment the API contract is exported from (Sandbox or Production).
EnvTo	The environment the API contract is exported to (Sandbox or Production).

Examples/Notes/Additional Information

In the example below, the action being performed is to approve all API contracts in the Production environment. The workflow first tests that there are contracts in existence, because a contract must exist before it can be approved.

```
<action id="301" name="reserved-approve-api-connection.Production">
  <restrict-to>
    <conditions type="AND">
      <condition type="existAPIContractsForAllAPIsInEnvironments">
<arg name="EnvFrom">Sandbox</arg>
<arg name="EnvTo">Production</arg>
      </condition>
      <condition type="allAPIContractsInEnvironmentApproved">
<arg name="Environment">Production</arg>
      </condition>
      <condition type="authorizeByAtmosphereRole">
```

```
<arg name="role">BusinessAdmin</arg>
      </condition>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Review" status="Approved" step="400" owner="${caller}" />
  </results>
</action>
```

App Version Workflow: Variables

The following variables are available for the app version workflow:

- ${app.dn}
- ${app.team.group.dn}

${app.dn}

The unique ID for the app version as a string in the format **${app.dn}**.

${app.team.group.dn}

The group ID for the app team members as a string in the format **${app.team.group.dn}**.

API Version Workflow

This section provides information about functions, conditions, and variables for the API version workflow, as well as initial actions and reserved actions.

API Version Workflow: Initial Actions

There is only one initial action valid for Community Manager workflows relating to APIs/API versions, as shown below:

- @Create

API Version Workflow: Reserved Actions

There are no reserved actions currently defined for API version workflows.

API Version Workflow: Functions

The following functions are available for the API version workflow:

- exportAPIVersion on page 21
- exportAPIAllVersions on page 22

exportAPIVersion

Exports the specified version of the specified API.

Parameters

None.

Examples/Notes/Additional Information

The example below uses this function to export the API version.

```
<action id="100" name="Export-WF">
  <restrict-to>
    <conditions type="AND">
      <condition type="authorizeByAtmosphereRole">
<arg name="role">ApiAdmin</arg>
      </condition>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Ready" status="Ready" step="100" owner="${caller}" />
  </results>
  <post-functions>
    <function type="exportAPIVersion"/>
  </post-functions>
</action>
```

exportAPIAllVersions

Exports all versions of the specified API.

Parameters

None.

Examples/Notes/Additional Information

The example below uses this function to export all versions of the API.

```
<action id="101" name="Export-All-Versions-WF">
  <restrict-to>
    <conditions type="AND">
      <condition type="authorizeByAtmosphereRole">
<arg name="role">ApiAdmin</arg>
      </condition>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Ready" status="Ready" step="100" owner="${caller}" />
  </results>
  <post-functions>
    <function type="exportAPIAllVersions"/>
  </post-functions>
</action>
```

API Version Workflow: Conditions

There are no conditions for the API version workflow.

API Version Workflow: Variables

The API version workflow governs the API Version object. There is one variable available for the API version workflow:

- ${api.dn}

${api.dn}

The unique ID for the API that the API version is associated with (APIID).

API ContractWorkflow

This section provides information about functions, conditions, and variables for the API contract workflow, as well as initial actions and reserved actions.

API Contract Workflow: Initial Actions

The initial actions valid for Community Manager workflows relating to API contracts are:

- @Create

  When a completely new contract is created (the first contract between a specific app version and a specific API version in a specific environment), the @Create initial action is used to start the workflow for the new contract.

- @Revise

  When an existing contract is revised, a new contract is created that is a revision of the existing contract, and the existing contract remains in place. In this scenario, the @Revise initial action is used to start the workflow for the revised contract. When there is a revised contract in place, any subsequent requests to create a contract for the app/API/environment combination will fail, since only one revised contract is allowed at one time.

- @ImportContract
- @AutoConnectActivate

API Contract Workflow: Reserved Actions

The following reserved actions are defined for API contract workflows:

- @app_switch_to_production
- @AppDeleted
- @ApiDeleted
- @modify
- @Revise
- reserved-connect-from-app.Sandbox
- reserved-connect-from-app.Production

API Contract Workflow: Functions

The following functions are available for the API contract workflow:

updateAPIContractStatus

Updates the status of an API contract. The new status must be a valid transition from the current status.

Parameters

Name	Description/Values
Status	Updates the status of the connection to a new status. Valid values: • apicontract.status.approved • apicontract.status.pending_approval • apicontract.status.config_pending • apicontract.status.rejected • apicontract.status.resubmitted • apicontract.status.activated • apicontract.status.cancelled • apicontract.status.suspended

Examples/Notes/Additional Information

The example below shows the workflow when production requests are auto-approved. The **updateAPIContractStatus** function is used to update the status.

```
<action id="102" name="Auto-Approve Production Requests From Production App" auto="TRUE">
  <restrict-to>
    <conditions type="AND">
      <condition type="isAtmosphereProductionApiContract" />
      <condition type="isAtmosphereProductionAutoApprove" />
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Pending" status="Activated" step="600" owner="${caller}"/>
  </results>
  <post-functions>
    <function type="updateAPIContractStatus">
      <arg name="status">apicontract.status.activated</arg>
    </function>
    <function type="sendNotification">
      <arg name="notificationType">com.soa.notification.type.api.access.requested.both.apiadmin</arg>
      <arg name="role">ApiAdmin</arg>
      <arg name="status">apicontract.status.activated</arg>
    </function>
    <function type="sendNotification">
      <arg name="notificationType">com.soa.notification.type.api.access.requested.production.appteam</arg>
      <arg name="role">AppAdmin</arg>
      <arg name="status">apicontract.status.activated</arg>
    </function>
  </post-functions>
</action>
```

sendNotification

Triggers the specified notification.

Parameters

Note: In some cases, some additional parameters are specific to individual notifications. For example, **param.contract.oldstate** is specific to a notification that identifies a change of state for a contract. Notification-specific parameters begin with "param." as in this example.

Name	Description/Values
notificationType	The type of notification being sent. Can be any valid notification existing in the platform. For example: • com.soa.notification.type.api.access.requested.both.apiadmin • com.soa.notification.type.api.access.requested.sandbox.appteam • com.soa.notification.type.privateapi.membership.status.changed • com.soa.notification.type.group.membership.role.changed • com.soa.notification.type.appteam.member.invited.team **Note**: if either the notificationType.production or notificationType.sandbox argument is provided, it is used to load the message template. If not, the notificationType argument is used to load the message template.
notificationType.production	If the connection is for the production environment, the notificationType.production argument is used to load the message template. If this argument is not provided, the notificationType argument is used for loading the message template.
notificationType.sandbox	If the connection is for the sandbox environment, the notificationType.sandbox argument is used to load the message template. If this argument is not provided, the notificationType argument is used for loading the message template.
role	The role to which the notifications will be sent—only users who hold this role for the specified API contract. Valid values: • ApiAdmin • AppAdmin
Any parameter with name prefixed with "param"	Some specific notifications have additional parameters. These parameters always begin with "param"—for example, ${contract.oldstate} is used in some cases to indicate the old state of the API contract, in scenarios where the state has changed.

Examples/Notes/Additional Information

In the example below, two notifications are sent when an API access request is auto-approved. Each notification goes to a different group of users as specified in the **role** argument.

```
<post-functions>
  <function type="autoApproveAccessRequest">
    <arg name="status">apicontract.status.activated</arg>
  </function>
  <function type="sendNotification">
     <arg name="notificationType">com.soa.notification.type.api.access.requested.both.apiadmin</arg>
     <arg name="role">ApiAdmin</arg>
```

```
  </function>
  <function type="sendNotification">
     <arg name="notificationType">com.soa.notification.type.api.access.requested.sandbox.appteam</arg>
     <arg name="role">AppAdmin</arg>
  </function>
</post-functions>
```

synchronizeAppVersion

If the app version for which the connection is set up is a remote federation member app, this function synchronizes the app identity with its system of record (home instance record). This ensures that the app identity information is up to date for the local tenant.

Parameters

None.

Examples/Notes/Additional Information

In the example below, the app version is synchronized as a last action, after the main function, to make sure the data is up to date before moving on to the next step in the workflow.

```
<action id="311" name="apicontract.action.sync.app.version">
  <restrict-to>
    <conditions type="AND">
      <condition type="authorizeByAtmosphereRole">
        <arg name="role">ApiAdmin</arg>
      </condition>
      <condition type="isRemoteFedMemberApp"/>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Pending Approval" status="Pending Approval" step="-1"/>
  </results>
  <post-functions>
    <function type="synchronizeAppVersion"/>
  </post-functions>
</action>
```

invokeAppVersionAction

Invokes the workflow action specified on the app version workflow.

By default, the workflow action is invoked on the app version associated with the connection unless the AppVersionDN argument is provided.

If the AppVersionDN argument is provided, the action is executed on the specified app version.

Parameters

Name	Description/Values
AppVersionDN (optional)	A specified app version the workflow action will be invoked on.

	If this parameter is not included, the workflow action is invoked on the app version associated with the connection.
ActionName	The workflow action to be invoked.

invokeApiVersionAction

Invokes the workflow action specified on the API version workflow.

By default, the workflow action is invoked on the API version associated with the connection unless the ApiVersionDN argument is provided.

If the ApiVersionDN argument is provided, the action is executed on the specified API version.

Parameters

Name	Description/Values
ApiVersionDN (optional)	A specified API version the workflow action will be invoked on. If this parameter is not included, the workflow action is invoked on the API version associated with the connection.
ActionName	The workflow action to be invoked.

updateContractActiveStatus

Updates the status of an active contract. The new status must be a valid transition from the current status.

Parameters

Name	Description/Values
status	The new status for the contract. Possible values: • com.soa.apicontract.inforce (used when the contract is activated). Indicates that the contract is currently in use when the app is invoking the API calls. • com.soa.apicontract.archived (when the contract is cancelled or the app or API version is deleted). Indicates that the contract was in use but is no longer in use. • com.soa.apicontract.draft: used when the contract hasn't yet reached the state in which it can be used in runtime requests; for example, pending acceptance.

Examples/Notes/Additional Information

In the example below, the API contract active status is updated to a status of com.soa.apicontract.inforce after the contract is activated.

```
<action id="2" name="@AutoConnectActivate">
  <results>
    <unconditional-result old-status="Received" status="Activated" step="600" owner="${caller}"/>
  </results>
  <post-functions>
     <function type="updateAPIContractStatus">
       <arg name="status">apicontract.status.activated</arg>
```

```
    </function>
    <function type="updateContractActiveStatus">
      <arg name="status">com.soa.apicontract.inforce</arg>
    </function>
    <function type="addAPIContractToHistory"/>
    <function type="sendNotification">
      <arg name="notificationType">com.soa.notification.type.api.access.state.change.apiadmin</arg>
      <arg name="role">ApiAdmin</arg>
      <arg name="param.contract.oldstate">apicontract.status.pending_approval</arg>
    </function>
    <function type="sendNotification">
      <arg name="notificationType">com.soa.notification.type.api.access.activated.production.appteam</arg>
      <arg name="role">AppAdmin</arg>
      <arg name="param.contract.oldstate">${contract.oldstate}</arg>
    </function>
  </post-functions>
</action>
```

API Contract Workflow: Conditions

The following conditions apply to the API contract workflow:

isAtmosphereApiContract

Returns **true** if used in the API contract workflow.

Parameters

None.

isAtmosphereSandboxApiContract

Returns **true** if the API contract is for the Sandbox environment.

Parameters

None.

isAtmosphereProductionApiContract

Returns **true** if the API contract is for the Production environment.

Parameters

None.

isAtmosphereSandboxAutoApprove

Returns **true** if the API contract is with an API version that has sandbox connections set up to be auto-approved.

Parameters

None.

isAtmosphereProductionAutoApprove

Returns **true** if the API contract is with an API version that has sandbox connections set up to be auto-approved.

Parameters

None.

isRemoteFedMemberApp

Returns **true** if the API contract is with an app that belongs to a federation member and the federation member is not in the same deployment.

Parameters

None.

apiContractUsesRestrictedScope

Returns **true** if the API contract is not unrestricted (restricted to one or more licenses/scopes). If the API contract is unrestricted, with full access to the API version, this condition returns **false**.

Parameters

None.

apiVersionSupportsResourceLevelPermissions

Returns **true** if the API contract is with an API version that supports resource-level permissions.

Parameters

None.

isAPIContractScopeNotEmpty

Tests to make sure that the contract gives the app access to at least one operation in the API; returns **true** if the contract doesn't give access to any operations in the app.

Tests whether the API contract is restricted AND either of the following is true:

- The API contract does not include any licenses in the scope.
- The aggregate of all licenses does not give it access to any of the operations based on how scope mapping is configured (mapping of License > Scope and mapping of Operation > Scope).

Parameters

None.

checkAppVersionStateMatches

Returns **true** if the app version state is one of the states specified in the State parameter.

Parameters

Name	Description/Values
AppVersionDN (optional)	A specified app version. If AppVersionDN is not defined, it will be the AppVersionID that the API contract is for.
State	A comma-separate list of State values.

checkAppVersionFedMemberMatches

Returns **true** if the app version belongs to one of the federation members specified.

Parameters

Name	Description/Values
AppVersionDN (optional)	A specified app version. If AppVersionDN is not defined, it will be the AppVersionID that the API contract is for.
FedMemberID	A comma-separated list of federation members.

API Contract Workflow: Variables

The following variables are available for the API contract workflow:

${contract.api.dn}

The API ID for the API version that the contract applies to. Note this is the API ID, not the API Version ID.

${contract.api.version.dn}

The APIVersionID of the API version that the connection is for.

${contract.app.dn}

The AppID of the app version that the connection is for.

${contract.app.version.dn}

The AppVersionID of the app version that the connection is for.

${contract.dn}

The APIContractID of the contract.

${contract.state}

Returns the current state of the API contract.

Note: there are two sets of values relating to contracts; contract state and contract status. Contract **state** indicates what point in the contract lifecycle the contract is currently in; for example, pending

approval, approved, activated. Contract **status** indicates whether the contract is active, not yet active, or archived, and determines which workflow actions are valid for the contract.

${contract.old.state}

When using the updateContractStatus function, the contract.old.state variable will be set for the subsequent functions/actions to use.

Ticket Workflow

This section provides information about functions, conditions, and variables for the ticket workflow, as well as initial actions and reserved actions.

Ticket Workflow: Initial Actions

There is only one initial action valid for Community Manager workflows relating to tickets, as shown below:

- @Create

Ticket Workflow: Reserved Actions

The following reserved actions are defined for ticket workflows:

- @reset
- @modify
- Open New Ticket
- ticket.action.resolve
- ticket.action.close
- ticket.action.delete
- ticket.action.edit.priority
- ticket.action.close
- ticket.action.reopen

Ticket Workflow: Functions

There is one function available for the ticket workflow:

- updateTicketStatus

updateTicketStatus

Updates the status of a ticket to the value provided in the argument.

Parameters

Name	Description/Values
status	The new status for the ticket after updating. Valid values: OPEN, RESOLVED, CLOSED, REOPEN.

Examples/Notes/Additional Information

The example below shows the ticket workflow when a ticket is closed. The **updateTicketStatus** function is used to update the ticket status to **CLOSED**.

```
<action id="201" name="ticket.action.close">
  <restrict-to>
    <conditions type="AND">
      <condition type="authorizeByAtmosphereRole">
<arg name="role">Admin</arg>
      </condition>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Open" status="Closed" step="400" owner="${caller}"/>
  </results>
  <post-functions>
    <function type="updateTicketStatus">
      <arg name="status">CLOSED</arg>
    </function>
  </post-functions>
</action>
```

Ticket Workflow: Conditions

There are no conditions for the ticket workflow.

Ticket Workflow: Variables

There are no variables for the ticket workflow.

Group Membership Workflow

This section provides information about functions, conditions, and variables for the group membership workflow, as well as initial actions and reserved actions.

Group Membership Workflow: Initial Actions

The initial actions valid for Community Manager workflows relating to group membership are:

- @Invite (the default, used in most cases)
- @Import: takes the group membership direct to the Approved state

Group Membership Workflow: Reserved Actions

The following reserved actions are defined for group membership workflows:

- @Invite
- @RecreateInPendingState
- @RecreateInAcceptedState
- @RecreateInDeclinedState
- group.membership.action.accept
- group.membership.action.decline
- group.membership.action.resend
- group.membership.action.remove
- group.membership.action.make.admin
- group.membership.action.make.leader
- group.membership.action.make.member
- group.membership.action.group.deleted

Group Membership Workflow: Functions

The following functions are available for the group membership workflow:

- setGroupMembershipRequestState on page 34
- setGroupMembershipRole on page 35
- sendGroupMembershipNotification on page 36

setGroupMembershipRequestState

Changes the state of a group membership request to a new state specified in the parameter.

Parameters

Name	Description/Values
State	The state that the group membership request is being set to. Valid values: • com.soa.group.membership.state.approved • com.soa.group.membership.state.disapproved • com.soa.group.membership.state.pending • com.soa.group.membership.state.removed • com.soa.group.membership.state.group.deleted

Examples/Notes/Additional Information

The example below shows the workflow step when an invited group member declines the invitation. As a result, the group membership request state is set to **com.soa.group.membership.state.disapproved**.

```
<action id="102" name="group.membership.action.decline">
  <restrict-to>
    <conditions type="AND">
      <condition type="isSelfMembership"></condition>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Pending" status="Declined" step="300" />
  </results>
  <!--  update status to declined -->
  <post-functions>
    <function type="setGroupMembershipRequestState">
      <arg name="state">com.soa.group.membership.state.disapproved</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.appteam.membership.rejected</arg>
      <arg name="groupType">com.soa.group.type.appteam</arg>
      <arg name="roles">role.group.all.members</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.privateapi.membership.rejected</arg>
      <arg name="groupType">com.soa.group.type.private.apigroup</arg>
      <arg name="roles">role.group.all.members</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.apiadmin.membership.rejected</arg>
      <arg name="groupType">com.soa.group.type.api.admingroup</arg>
      <arg name="roles">role.group.all.members</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.bizadmin.membership.rejected</arg>
      <arg name="groupType">com.soa.group.type.business.admingroup</arg>
      <arg name="roles">role.group.all.members</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.siteadmin.membership.rejected</arg>
      <arg name="groupType">com.soa.group.type.tenant.admingroup</arg>
      <arg name="roles">role.group.all.members</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.group.membership.rejected</arg>
      <arg name="groupType">com.soa.group.type.independent</arg>
      <arg name="roles">role.group.all.members</arg>
    </function>
  </post-functions>
</action>
```

setGroupMembershipRole

Sets the group membership role for the specified group member to a new role.

Parameters

Name	Description/Values
role	The group role which is now assigned to the specified group member. Valid values: • com.soa.group.membership.role.admin • com.soa.group.membership.role.leader • com.soa.group.membership.role.member

Examples/Notes/Additional Information

In the example below, the group member is made into a leader. The setGroupMembershipRole function is used to change the user's role to **com.soa.group.membership.role.leader**.

```
<action id="106" name="group.membership.action.make.leader">
  <restrict-to>
    <conditions type="OR">
      <condition type="isCallerGroupAdmin" />
      <condition type="isCallerSiteAdmin"/>
      <conditions type="AND">
<condition type="isCallerGroupLeader" />
<conditions type="OR">
  <condition type="isLeaderMembership" />
  <condition type="isMemberMembership" />
        </conditions>
      </conditions>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Pending" status="Pending" step="100" />
  </results>
  <post-functions>
    <function type="setGroupMembershipRole">
      <arg name="role">com.soa.group.membership.role.leader</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.privateapi.membership.status.changed</arg>
      <arg name="groupType">com.soa.group.type.private.apigroup</arg>
      <arg name="roles">role.group.all.members,role.invited.user</arg>
      <arg name="param.groupmembership.oldrole">${groupmembership.oldrole}</arg>
      <arg name="param.groupmembership.role">${groupmembership.role}</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.group.membership.role.changed</arg>
      <arg name="groupType">com.soa.group.type.independent</arg>
      <arg name="roles">role.group.all.members,role.invited.user</arg>
      <arg name="param.groupmembership.oldrole">${groupmembership.oldrole}</arg>
      <arg name="param.groupmembership.role">${groupmembership.role}</arg>
    </function>
  </post-functions>
</action>
```

sendGroupMembershipNotification

Sends group membership email and dashboard notifications when parameters match the group membership. Since the same function is used for all groups and all membership, this function is

designed so that different notifications can be sent to different roles or different group membership depending on the event.

Parameters

Name	Description/Values
notificationType	Valid notification ID of the notification message template for the notification to be sent. For example: • com.soa.notification.type.appteam.member.invited.team • com.soa.notification.type.independent.group.deleted
groupType	Checks whether the sendGroupMembershipNotification function is for the current group type for which the membership is being updated. When the value of this parameter matches the group type of the group membership that is being affected, a notification is sent. Otherwise, no action is taken. The group workflow is designed to accommodate several different group types. Depending on the group type, a different notification might be sent. This parameter tests the current group type. Valid values: • com.soa.group.type.appteam: App team • com.soa.group.type.private.apigroup: Private API group • com.soa.group.type.tenant.admingroup: Site Administrator • com.soa.group.type.api.admingroup: API Administrator • com.soa.group.type.independent: Independent group • com.soa.group.type.business.admingroup: Business Administrator
Roles	Defines the roles to which the notification is sent, as a comma-separated list of role names. Role names used for this parameter are: • **role.group.all.members**: the notification is sent to all confirmed members of the specified group or groups (admins, leaders, and members). • **role.group.leader**: Notification is sent to all leaders of the group. • **role.group.admin**: Notification is sent to all admins of the group. • **role.group.member**: Notification is sent to all members of the group. • **role.invited.user.unregistered**: Notification is sent to users who were invited but have not yet accepted the invitation (only if user is not yet registered on the platform). • **role.invited.user.registered**: Notification is sent to users who were invited but have not yet accepted the invitation (only if user is registered on the platform). • **role.invited.user**: Notification is sent to users who were invited but have not yet accepted the invitation (whether registered or unregistered). • **role.inviting.user**: Notification is sent to the user that is sending the invitation (applies to only invite and resend actions).
Any parameter with name prefixed with “param”	Parameter values for parameters with names starting with “param.” Can be used in the notification template data. Examples: • param.groupmembership.oldrole: Optional, used when a group member

	has a role change, to indicate the old role. • param.groupmembership.role: Optional, used when a group member has a role change, to indicate the new role.

Examples/Notes/Additional Information

The example below shows workflow steps when a group is deleted. One of two possible notifications is sent out, depending on the group type, to notify group members that the group was deleted.

```
<action id="108" name="group.membership.action.group.deleted">
  <results>
    <unconditional-result old-status="Pending" status="Group Deleted" step="400" />
  </results>
  <post-functions>
    <function type="setGroupMembershipRequestState">
      <arg name="state">com.soa.group.membership.state.group.deleted</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.independent.group.deleted</arg>
      <arg name="groupType">com.soa.group.type.independent</arg>
      <arg name="roles">role.invited.user</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.privateapi.group.deleted</arg>
      <arg name="groupType">com.soa.group.type.private.apigroup</arg>
      <arg name="roles">role.invited.user</arg>
    </function>
  </post-functions>
</action>
```

Group Membership Workflow: Conditions

The following conditions apply to the group membership workflow:

isSelfMembership

Checks whether the group membership is for the logged-in user; returns Boolean true if so.

Arguments

None.

Examples/Notes/Additional Information

The example below shows workflow for when a group member accepts an invitation to join the group. Since the invited person is the only one who is authorized to accept the invitation, the workflow uses the **isSelfMembership** condition to check that the individual performing the action is the invited member.

```
<actions>
  <action id="101" name="group.membership.action.accept">
    <restrict-to>
      <conditions type="AND">
        <condition type="isSelfMembership"></condition>
      </conditions>
    </restrict-to>
    <results>
      <unconditional-result old-status="Pending" status="Accepted" step="200" />
    </results>
    <!--  update status to approved -->
    <post-functions>
      <function type="setGroupMembershipRequestState">
        <arg name="state">com.soa.group.membership.state.approved</arg>
      </function>
      <!--
      <function type="addBoardComment">
        <arg name="notificationType">com.soa.notification.type.accept.appteam.invite</arg>
      </function>
      -->
      <function type="sendGroupMembershipNotification">
        <arg name="notificationType">com.soa.notification.type.appteam.membership.accepted</arg>
        <arg name="groupType">com.soa.group.type.appteam</arg>
        <arg name="roles">role.group.all.members</arg>
      </function>
      <function type="sendGroupMembershipNotification">
        <arg name="notificationType">com.soa.notification.type.privateapi.membership.accepted</arg>
        <arg name="groupType">com.soa.group.type.private.apigroup</arg>
        <arg name="roles">role.group.all.members</arg>
      </function>
      <function type="sendGroupMembershipNotification">
        <arg name="notificationType">com.soa.notification.type.apiadmin.membership.accepted</arg>
        <arg name="groupType">com.soa.group.type.api.admingroup</arg>
        <arg name="roles">role.group.all.members</arg>
      </function>
      <function type="sendGroupMembershipNotification">
        <arg name="notificationType">com.soa.notification.type.bizadmin.membership.accepted</arg>
        <arg name="groupType">com.soa.group.type.business.admingroup</arg>
        <arg name="roles">role.group.all.members</arg>
      </function>
      <function type="sendGroupMembershipNotification">
        <arg name="notificationType">com.soa.notification.type.siteadmin.membership.accepted</arg>
        <arg name="groupType">com.soa.group.type.tenant.admingroup</arg>
        <arg name="roles">role.group.all.members</arg>
      </function>
      <function type="sendGroupMembershipNotification">
        <arg name="notificationType">com.soa.notification.type.group.membership.accepted</arg>
```

```
      <arg name="groupType">com.soa.group.type.independent</arg>
      <arg name="roles">role.group.all.members</arg>
    </function>
  </post-functions>
 </action>
```

isCallerSiteAdmin

Checks whether the logged-in user is a Site Admin; returns Boolean true if so.

Arguments

None.

Examples/Notes/Additional Information

In the example below, the action being performed changes a group member's role to **admin**. The conditions section tests that the user performing the action is either a group admin or a site admin, since these are the two roles authorized to perform the action.

```
<action id="105" name="group.membership.action.make.admin">
  <restrict-to>
    <conditions type="OR">
      <condition type="isCallerGroupAdmin" />
      <condition type="isCallerSiteAdmin"/>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Pending" status="Pending" step="100" />
  </results>
  <post-functions>
    <function type="setGroupMembershipRole">
      <arg name="role">com.soa.group.membership.role.admin</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.privateapi.membership.status.changed</arg>
      <arg name="groupType">com.soa.group.type.private.apigroup</arg>
      <arg name="roles">role.group.all.members,role.invited.user</arg>
      <arg name="param.groupmembership.oldrole">${groupmembership.oldrole}</arg>
      <arg name="param.groupmembership.role">${groupmembership.role}</arg>
    </function>
<function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.group.membership.role.changed</arg>
      <arg name="groupType">com.soa.group.type.independent</arg>
      <arg name="roles">role.group.all.members,role.invited.user</arg>
      <arg name="param.groupmembership.oldrole">${groupmembership.oldrole}</arg>
      <arg name="param.groupmembership.role">${groupmembership.role}</arg>
    </function>
  </post-functions>
</action>
```

isCallerGroupAdmin

Checks whether the individual performing the action is an admin for the group; returns Boolean true if so.

Arguments

None.

Examples/Notes/Additional Information

In the example below, the action being performed changes a group member's role to **admin**. The conditions section tests that the user performing the action is either a group admin or a site admin, since these are the two roles authorized to perform the action.

```
<action id="105" name="group.membership.action.make.admin">
  <restrict-to>
    <conditions type="OR">
      <condition type="isCallerGroupAdmin" />
      <condition type="isCallerSiteAdmin"/>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Pending" status="Pending" step="100" />
  </results>
  <post-functions>
    <function type="setGroupMembershipRole">
      <arg name="role">com.soa.group.membership.role.admin</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.privateapi.membership.status.changed</arg>
      <arg name="groupType">com.soa.group.type.private.apigroup</arg>
      <arg name="roles">role.group.all.members,role.invited.user</arg>
      <arg name="param.groupmembership.oldrole">${groupmembership.oldrole}</arg>
      <arg name="param.groupmembership.role">${groupmembership.role}</arg>
    </function>
<function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.group.membership.role.changed</arg>
      <arg name="groupType">com.soa.group.type.independent</arg>
      <arg name="roles">role.group.all.members,role.invited.user</arg>
      <arg name="param.groupmembership.oldrole">${groupmembership.oldrole}</arg>
      <arg name="param.groupmembership.role">${groupmembership.role}</arg>
    </function>
  </post-functions>
</action>
```

isCallerGroupAdminMember

Checks whether the logged-in user (the user triggering the action) is a group admin member; returns Boolean true if so.

Arguments

None.

isCallerGroupLeader

Checks whether the logged-in user is a group leader; returns Boolean true if so.

Arguments

None.

Examples/Notes/Additional Information

In the example below, the action being performed makes someone a group member. The conditions section specifies that a user who is a group admin or site admin can perform the action; additionally, a user who is a group leader can perform the action on another user who is a leader or member.

This limits the action to authorized individuals.

```
<action id="107" name="group.membership.action.make.member">
  <restrict-to>
    <conditions type="OR">
      <condition type="isCallerGroupAdmin" />
      <condition type="isCallerSiteAdmin"/>
      <conditions type="AND">
<condition type="isCallerGroupLeader" />
<conditions type="OR">
  <condition type="isLeaderMembership" />
  <condition type="isMemberMembership" />
        </conditions>
      </conditions>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Pending" status="Pending" step="100" />
  </results>
  <post-functions>
    <function type="setGroupMembershipRole">
      <arg name="role">com.soa.group.membership.role.member</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.privateapi.membership.status.changed</arg>
      <arg name="groupType">com.soa.group.type.private.apigroup</arg>
      <arg name="roles">role.group.all.members,role.invited.user</arg>
      <arg name="param.groupmembership.oldrole">${groupmembership.oldrole}</arg>
      <arg name="param.groupmembership.role">${groupmembership.role}</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.group.membership.role.changed</arg>
      <arg name="groupType">com.soa.group.type.independent</arg>
      <arg name="roles">role.group.all.members,role.invited.user</arg>
      <arg name="param.groupmembership.oldrole">${groupmembership.oldrole}</arg>
      <arg name="param.groupmembership.role">${groupmembership.role}</arg>
    </function>
  </post-functions>
```

```
</action>
```

isCallerGroupMember

Checks whether the logged-in user is a group member; returns Boolean true if so.

Arguments

None.

isMemberMembership

Checks the role of the group membership being managed with the workflow to see if the role is Member (group membership represents member membership). If the role is Member, returns **true**.

Used in combination with isCallerGroupAdminMember, isCallerGroupLeader, and isCallerGroupMember:

- **To check the role of the user calling the workflow**:
  - isCallerGroupAdminMember
  - isCallerSiteAdmin
  - isCallerGroupAdmin
  - isCallerGroupLeader
  - isCallerGroupMember
- **To check the role of the group membership being managed with the action**:
  - isAdminMembership
  - isLeaderMembership
  - isMemberMembership
  - isSelfMembership

This combination of conditions verifies that the person calling the workflow has adequate rights to perform the action. A group admin can perform actions relating to admins, leaders, or members; a leader can perform actions relating to leaders or members; a member can perform actions relating to members.

Arguments

None.

Examples/Notes/Additional Information

In the example below, the action being performed is to assign an existing group member the role of **member** (as distinct from **leader** or **admin**). The “restrict-to” section tests that the user performing the action is either a group admin or a site admin, or that the user is a group leader and the action is being

performed on a group member who has the role of **leader** or **member**; and, therefore, that the user performing the action is authorized to do so.

```
<action id="107" name="group.membership.action.make.member">
  <restrict-to>
    <conditions type="OR">
      <condition type="isCallerGroupAdmin" />
      <condition type="isCallerSiteAdmin"/>
      <conditions type="AND">
<condition type="isCallerGroupLeader" />
<conditions type="OR">
  <condition type="isLeaderMembership" />
  <condition type="isMemberMembership" />
        </conditions>
      </conditions>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Pending" status="Pending" step="100" />
  </results>
  <post-functions>
    <function type="setGroupMembershipRole">
      <arg name="role">com.soa.group.membership.role.member</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.privateapi.membership.status.changed</arg>
      <arg name="groupType">com.soa.group.type.private.apigroup</arg>
      <arg name="roles">role.group.all.members,role.invited.user</arg>
      <arg name="param.groupmembership.oldrole">${groupmembership.oldrole}</arg>
      <arg name="param.groupmembership.role">${groupmembership.role}</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.group.membership.role.changed</arg>
      <arg name="groupType">com.soa.group.type.independent</arg>
      <arg name="roles">role.group.all.members,role.invited.user</arg>
      <arg name="param.groupmembership.oldrole">${groupmembership.oldrole}</arg>
      <arg name="param.groupmembership.role">${groupmembership.role}</arg>
    </function>
  </post-functions>
</action>
```

isLeaderMembership

Checks the role of the group membership being managed with the workflow to see if the role is Leader. If the role is Leader, returns **true**.

Used in combination with isCallerGroupAdminMember, isCallerGroupLeader, and isCallerGroupMember:

- **To check the role of the user calling the workflow**:
  - isCallerGroupAdminMember
  - isCallerGroupLeader
  - isCallerGroupMember

- **To check the role of the group member being managed with the action**:
  - isAdminMembership
  - isLeaderMembership
  - isMemberMembership

This combination of conditions verifies that the person calling the workflow has adequate rights to perform the action. A group admin can perform actions relating to admins, leaders, or members; a leader can perform actions relating to leaders or members; a member can perform actions relating to members.

Arguments

None.

Examples/Notes/Additional Information

In the example below, the action being performed is to assign an existing group member the role of **leader** (as distinct from **member** or **admin**). The "restrict-to" section tests that the user performing the action is either a group admin or a site admin, or that the user is a group leader and the action is being performed on a group member who has the role of **leader** or **member**; and, therefore, that the user performing the action is authorized to do so.

```
<action id="106" name="group.membership.action.make.leader">
  <restrict-to>
    <conditions type="OR">
      <condition type="isCallerGroupAdmin" />
      <condition type="isCallerSiteAdmin"/>
      <conditions type="AND">
<condition type="isCallerGroupLeader" />
<conditions type="OR">
  <condition type="isLeaderMembership" />
  <condition type="isMemberMembership" />
        </conditions>
      </conditions>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Pending" status="Pending" step="100" />
  </results>
  <post-functions>
    <function type="setGroupMembershipRole">
      <arg name="role">com.soa.group.membership.role.leader</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.privateapi.membership.status.changed</arg>
      <arg name="groupType">com.soa.group.type.private.apigroup</arg>
      <arg name="roles">role.group.all.members,role.invited.user</arg>
      <arg name="param.groupmembership.oldrole">${groupmembership.oldrole}</arg>
      <arg name="param.groupmembership.role">${groupmembership.role}</arg>
    </function>
    <function type="sendGroupMembershipNotification">
      <arg name="notificationType">com.soa.notification.type.group.membership.role.changed</arg>
      <arg name="groupType">com.soa.group.type.independent</arg>
      <arg name="roles">role.group.all.members,role.invited.user</arg>
```

```
    <arg name="param.groupmembership.oldrole">${groupmembership.oldrole}</arg>
    <arg name="param.groupmembership.role">${groupmembership.role}</arg>
   </function>
  </post-functions>
</action>
```

isAdminMembership

Checks the role of the group membership being managed with the workflow to see if the role is Admin. If the role is Admin, returns **true**.

Used in combination with isCallerGroupAdminMember, isCallerGroupLeader, and isCallerGroupMember:

- **To check the role of the user calling the workflow**:
  - isCallerGroupAdminMember
  - isCallerGroupLeader
  - isCallerGroupMember
- **To check the role of the group member being managed with the action**:
  - isAdminMembership
  - isLeaderMembership
  - isMemberMembership

This combination of conditions verifies that the person calling the workflow has adequate rights to perform the action. A group admin can perform actions relating to admins, leaders, or members; a leader can perform actions relating to leaders or members; a member can perform actions relating to members.

Arguments

None.

Group Membership Workflow: Variables

The following variables are available for the group membership workflow:

- ${group.dn}
- ${group.type}
- ${group.membership.request.dn}
- ${membership.id}
- ${member.dn}
- ${group.membership.old.role}
- ${group.membership.old.state}
- ${group.membership.role}
- ${group.membership.state}

${group.dn}

The unique GroupID for the group.

${group.type}

A value indicating the group type as a string in the format **${group.type}**.

Valid values:

- com.soa.group.type.tenant.admingroup
- com.soa.group.type.business.admingroup
- com.soa.group.type.internal
- com.soa.group.type.appteam
- com.soa.group.type.api.admingroup
- com.soa.group.type.independent
- com.soa.group.type.private.apigroup

${group.membership.request.dn}

The unique GroupMembershipRequestID. This is the Board Item ID corresponding to the group membership request. All audits related to the group membership are tracked under this request ID.

${membership.id}

The unique ID for the individual's membership in the group; a number.

${member.dn}

The unique User ID of the group member.

${groupmembership.oldrole}

The previous group membership role.

When the setGroupMembershipRole function is used, this variable is set with old role name of the group membership for use in subsequent conditions and functions.

${groupmembership.oldstate}

The previous group membership state.

When the setGroupMembershipRequestState function is used, this variable is set with the state name of the group membership for use in subsequent conditions and functions for the duration of the workflow action being performed.

${groupmembership.role}

The group membership role.

${groupmembership.state}

The group membership state.