Community Manager®: Customizing Workflows

Version 7.2.2

SOA | software™

Community Manager

Customizing Workflows
Version 7.2.2
February, 2015

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, through initial actions that trigger the workflow, to additional actions that change the state of the resource, from beginning to end. Major components include conditions (for example, a ticket must be resolved before it can be closed, or, an action can only be completed by a Site Admin) and state (for example, a ticket can have states of Open, Resolved, and Closed).

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

Community Manager allows you to customize the workflow for several key resources, such as API contracts. It also includes examples of custom workflows for some resources.

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 on page 9—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 on page 17—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.

Examples of Custom Workflow Usage

This section includes just a few examples of how you could use workflow definitions to customize process flow in Community Manager:

- 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.
- **Tickets**: 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.
- **Contracts**: You might 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. Note that you would also need to update the site documentation.

- **APIs**: 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.
- **Apps**: If an app is deleted, you could use a reserved action to determine what happens to API contracts associated with the app.

Custom Workflows

The ability to customize the platform default workflow for resources gives you great flexibility.

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

Using Custom Workflows: Overview

By developing a custom workflow you can control how resources behave in the system as a result of certain actions.

The basic building blocks of workflow are:

- **Actions**. An **action** element changes a resource from one state to another by accomplishing some activity on the resource. An action element indicates what is happening to the resource. There are two main scenarios:
  - In some cases, a workflow action can replace the default behavior in the platform. For example, in the app version workflow, you could redefine the steps that are taken when the **@delete** action occurs, so that an administrator must approve the deletion. As another example, you could set up the workflow so that when the app certificate is modified (**@KeyInfoSaved** action), a notification is sent.
  - In other cases, the action can extend the default behavior. For example, you could set up the group membership workflow so that if a group is deleted all members receive a notification.
- **Steps**: A **step** element indicates the state of a resource and defines the actions that are valid when a resource is in that state.
- **Conditions**: A workflow action can be restricted by the results of one or more **condition** elements. The condition element determines whether or not the action is valid. For example, a condition might test that the user attempting to perform the action has a valid role and is therefore authorized to perform the action.

- **Functions**: A **function** element acts on the resource and changes it in some way; for example, the **deleteReview** function deletes a review, ending the workflow for that review. The function might have arguments. In a workflow step, functions can be either of the below:
  - **Pre-functions**: The pre-functions element indicates that the function is executed before entering the step or action.
  - **Post-functions**: the function is executed after leaving the step or action.

Resources With Customizable Workflow

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

- API: see App Version Workflow on page 20
- App: see API Version Workflow on page 31
- API Contract: see API Contract Workflow on page 33
- Group Membership: see Group Membership Workflow on page 45
- Ticket: see Ticket Workflow on page 43
- User: see User Workflow on page 60
- Reviews: see Review Workflow on page 67
- Discussions: see Discussions Workflow on page 73
- Comments: see Comments Workflow on page 78

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.

Note: It is best to test a new custom workflow in a sandbox environment before implementing in your production environment. For more information, see *Testing a New Custom Workflow* on page 11.

1 Optional: download existing workflow. You can download an existing workflow to use as a starting point for the customization (Administration > Workflows > View > Download). For more information, refer to the platform help.

2 Create the custom workflow outside the platform. For more information and technical details, refer to *The Workflow Definition XML* File on page 13. (**Admin or delegate**)

3 Upload the custom workflow to a sandbox environment. See *Uploading a New Custom* Workflow below. (**Site Admin only**)

4 Assign the new custom workflow as the default for new resources of that type. See *Implementing a New Custom Workflow* on page 12. (**Site Admin**)

5 Test the new custom workflow, 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 11. (**Site Admin, Business Admin, or resource admin**)

6 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. (**Site Admin or delegate**)

7 When you're sure that everything is functioning correctly, it's time to update the production environment:

- Upload the new workflow (Step 3 above).
- Assign it as the default for the resource type (Step 4 above).
- If needed, upload the updated documentation (Step 6 above).

Uploading a New Custom Workflow

Roles in custom workflow management are as follows:

- **Uploading a new custom workflow**: Site Admin or Business Admin
- **Changing the workflow definition for a specific type of resource**: Site Admin

To upload a custom workflow, log in as the Site Admin or Business 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, available at http://docs.soa.com/docs-test/cm/learning.html.

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 that:

– 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.

Only the Site Admin can change the workflow definition for a specific type of resource.

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.

- **To upload a custom workflow:** Log in as a Site Admin or Business Admin, go to Administration > Workflows, and then click **Add Workflow**. Specify Name and Description. In the **Object Type** drop-down list, make sure you choose the correct type of resource. Navigate to the location of the file and upload it.
- **To change the workflow for a type of resource:** Log in as a Site Admin, go to Administration > Settings, and then choose the resource type. Update the workflow definition 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 User Documentation

It is the Site Admin'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.

The Workflow Definition XML File

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

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

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. In Community Manager, initial actions always start with the @ sign.

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 changes that might commonly be made to objects as a result of something else that happens in the system, as distinct from changes that occur as a result of a user's action such as clicking a button. A reserved action is triggered automatically under certain conditions, as defined in the workflow. For example, the app version workflow could be set up so that if the app certificate is updated, an email is sent to your internal system administrator.

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 the conditions under which certain steps can be performed.

   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 this configurable behavior.

2. To initiate an action internally when something else happens, or to extend what happens as part of another action. This type of reserved action is called an *initial-action* as covered in the previous section.

   The naming of this type of reserved action generally indicates that an action has occurred. For example, in the app workflow, @KeyInfoRemoved indicates that the application security certificate was removed; in the user workflow, @ChallengeQuestionsAnswered indicates that the user has answered the challenge questions. Because these are defined as reserved actions, the workflow can then be built to trigger additional actions.

   For example, if a group is deleted, the reserved action **group.membership.action.group.deleted** is invoked. This 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.

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, with a couple of exceptions. Naming conventions are as follows:

- Action name begins with "**reserved-**"; for example, under certain conditions the "reserved-connect" reserved action is sent to the user interface (UI) so that the UI 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 an **@** 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)
- Action name is neither of the above, but is defined as a workflow action in the name. There are a few legacy workflow action names in the group membership workflow that do not follow the above naming conventions. They are:
  - 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

  For more information, refer to Group Membership Workflow: Reserved Actions on page 45.

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="105" name="@RegeneratedSecret">

<action id="106" name="reserved-allow-cert-upload">

<action id="301" name="reserved-action.unpublish">
```

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 type of resource for which workflow can be customized, such as apps, tickets, or discussions

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 might have 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, the workflow snippet below restricts the workflow action to a specific role. The role specified in the argument is AppAdmin. One or more roles can be specified.

```
<restrict-to>
  <conditions type="AND">
    <condition type="authorizeByAtmosphereRole">
      <arg name="role">AppAdmin</arg>
    </condition>
  </conditions>
</restrict-to>
```

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, and is therefore authorized to perform the workflow action; 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 the contracts a specific app has with APIs, 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

There is no default workflow applied to app versions, but the platform supports implementation of custom workflow for app versions.

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:

- @Modify

  Used to determine whether an app version can be modified or should be read-only. If @Modify is not a valid action, the user interface disables the buttons/links leading to modifying anything about the app version.

- @Delete

  Indicates that the app version can be deleted at this point in the workflow, using the API or the user interface.

  If you want the app version to be immediately deleted, you can invoke the deleteAppVersion function for @Delete. However, if you want to set up an approval process, you can instead take the app to a different state. The custom out-of-the-box app workflow, **appversion-workflow-template1.xml**, uses workflow step 200, **Marked For Delete**, to initiate a Business Admin approval process. If the Business Admin approves the request, the app is deleted.

- @KeyInfoRemoved

  Invoked when the certificate/CSR info that was previously added for an app version is removed.

  You can use this to trigger additional actions in the workflow; for example, sending notifications or generating Board items.

- @KeyInfoSaved

  Invoked when the app version's certificate/CSR is added or modified.

  You can use this to trigger additional actions in the workflow; for example, sending notifications or generating Board items.

- @RegeneratedSecret

  Invoked when the app Shared Secret is regenerated.

You can use this to trigger additional actions in the workflow; for example, sending notifications or generating Board items.

- reserved-allow-cert-upload

  If this reserved action is present in the workflow, certificate upload is allowed.

- reserved-connect-to-api

  If this reserved action is present in the workflow, an authorized user can initiate a connection to an API in either environment.

- reserved-connect-to-api.Sandbox

  If this reserved action is present in the workflow, an authorized user can initiate a connection to an API in the Sandbox environment.

- reserved-connect-to-api.Production

  If this reserved action is present in the workflow, an authorized user can initiate a connection to an API in the Production environment.

- reserved-approve-api-connection.Production

  If this reserved action is present in the workflow, an authorized user can approve a connection to an API in the Production environment.

- reserved-cancel-api-connection.Sandbox

  If this reserved action is present in the workflow, an authorized user can approve a connection to an API in the Sandbox environment.

- reserved-cancel-api-connection

  If this reserved action is present in the workflow, an authorized user can cancel a connection to an API.

App Version Workflow: Functions

The following functions are available for the app version workflow:

- cloneAllAPIContracts on page 21
- activateAllAPIContractsInEnvironment on page 22
- cancelAllAPIContractsInEnvironment on page 22
- sendNotification on page 23
- deleteAppVersion on page 24
- addBoardItem on page 25

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. Values: Sandbox or Production
EnvTo	Target environment: The environment contracts are being cloned to. Values: 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. Values: 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. Values: 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>
```

sendNotification

Triggers the specified notification based on an event relating to an app version.

Parameters

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.app.marked.for.deletion.appteam • com.soa.notification.type.app.marked.for.deletion.apiadmin • com.soa.notification.type.app.marked.for.deletion.bizadmin
Role	The role to which the notifications will be sent. Valid values: • ApiAdmin • AppAdmin • BusinessAdmin

Examples/Notes/Additional Information

In the example below, three different notifications are sent out, one for each of three different types of users, to notify the users that the app was marked for deletion.

```
<function type="sendNotification">
  <arg name="notificationType">com.soa.notification.type.app.marked.for.deletion.appteam</arg>
  <arg name="role">AppAdmin</arg>
</function>
<function type="sendNotification">
  <arg name="notificationType">com.soa.notification.type.app.marked.for.deletion.apiadmin</arg>
  <arg name="role">ApiAdmin</arg>
</function>
<function type="sendNotification">
  <arg name="notificationType">com.soa.notification.type.app.marked.for.deletion.bizadmin</arg>
  <arg name="role">BusinessAdmin</arg>
</function>
```

deleteAppVersion

This function permanently deletes the application version from the database.

It does not add a Board item or notification, but you can add either or both of these as additional functions:

- To add a notification, use **sendNotification** (see above).
- To add a Board item, use **addBoardItem** (see addBoardItem on page 25).

The out-of-the-box app version workflow includes notifications and Board items as additional functions, by default. The **deleteAppVersion** function itself does not send them.

Note: If you want to add Board items or functions, add them before the deleteAppVersion function is run, so that information needed for the Board item, such as the App Version ID, is still available. Once app version is deleted, the App Version ID is no longer valid.

Parameters

None.

Examples/Notes/Additional Information

In the example below, **deleteAppVersion** is run as the last pre-function.

```
<pre-functions>
  <function type="addBoardItem">
    <arg name="boardItemTemplateId">com.soa.board.item.appversion.delete.request.approved</arg>
    <arg name="visibility">Limited</arg>
    <arg name="type">Discussion</arg>
    <arg name="targetBoard.appversion">${app.version.dn}</arg>
    <arg name="targetBoard.appdn">${app.dn}</arg>
    <arg name="targetBoard.appteamgrp">${app.team.group.dn}</arg>
    <arg name="targetBoard.apiversion">${connected.apiversion.ids}</arg>
    <arg name="targetBoard.apiadmin.groups">${connected.apis.admin.groups}</arg>
    <arg name="targetBoard.api">${connected.apis.id}</arg>
    <arg name="viewers">${app.dn},${app.team.group.dn},${connected.apis.id},${business.dn}</arg>
          </function>
  <function type="sendNotification">
    <arg name="notificationType">com.soa.notification.type.app.deletion.request.approved.apiadmin</arg>
    <arg name="role">ApiAdmin</arg>
  </function>
  <function type="sendNotification">
    <arg name="notificationType">com.soa.notification.type.app.deletion.request.approved.appteam</arg>
    <arg name="role">AppAdmin</arg>
  </function>
  <function type="sendNotification">
    <arg name="notificationType">com.soa.notification.type.app.deletion.request.approved.bizadmin</arg>
    <arg name="role">BusinessAdmin</arg>
  </function>
  <function type="deleteAppVersion" />
</pre-functions>
```

addBoardItem

Adds a Board item to one or more boards.

Parameters

Name	**Description/Values**
boardItemTemplateId	The ID of the Board item notification template, from the database, to be used for the Board item title and description. For example: • com.soa.board.item.appversion.delete.request.approved
Visibility	The visibility of the Board item. Valid values: • Public • Limited • RegisteredUsers
Type	The Board item type. Currently, the only valid value is **Discussion**.
targetBoard	Used to specify that one item could be added to multiple boards. For example, the Delete App board item could be added to the app board and also the API boards for connected APIs. It could also be added to other boards, such as the board for each team member. There are two ways to add the targetBoard information (see example below): • **targetBoard.{parametername}**, listing separately each parameter to be added to the target board. • **targetBoards**, plural parameter, with a comma-separated list of Board items to be added.
viewers	Indicates who can view the Board item. For example, providing the App ID as a value indicates that anyone who can administer the app can view the Board item. There are two ways to specify the viewers: • **viewer.{parametername}**, listing separately each applicable ID to indicate that users who have view of that resource have view of the Board item. • **viewers**, plural parameter, with a comma-separated list of Board items to be added. Examples of values:

	• app.team.group.dn: app team • connected.apis.id: API Admins for connected APIs • business.dn: Business Admins for the business

Examples/Notes/Additional Information

The example below shows adding a Board item as a pre-function, to announce that an app deletion request was approved. In this example, **targetBoard** is a set of separate arguments.

```
<pre-functions>
  <function type="addBoardItem">
    <arg name="boardItemTemplateId">com.soa.board.item.appversion.delete.request.approved</arg>
    <arg name="visibility">Limited</arg>
    <arg name="type">Discussion</arg>
    <arg name="targetBoard.appversion">${app.version.dn}</arg>
    <arg name="targetBoard.appdn">${app.dn}</arg>
    <arg name="targetBoard.appteamgrp">${app.team.group.dn}</arg>
    <arg name="targetBoard.apiversion">${connected.apiversion.ids}</arg>
    <arg name="targetBoard.apiadmin.groups">${connected.apis.admin.groups}</arg>
    <arg name="targetBoard.api">${connected.apis.id}</arg>
    <arg name="viewers">${app.dn},${app.team.group.dn},${connected.apis.id},${business.dn}</arg>
  </function>
```

The example below shows adding a Board item as a post-function, to announce that an app version has been marked for deletion. In this example, the **targetBoards** parameter is used, with multiple values in a comma-separated list.

```
<post-functions>
  <function type="addBoardItem">
    <arg name="boardItemTemplateId">com.soa.board.item.appversion.mark.for.delete</arg>
    <arg name="visibility">Limited</arg>
    <arg name="type">Discussion</arg>
    <arg name="targetBoards">${app.version.dn},${app.dn},${app.team.group.dn},${connected.apiversion.ids},
${connected.apis.admin.groups},${connected.apis.id},${app.version.dn},${app.dn}</arg>
    <arg name="viewers">${app.dn},${app.team.group.dn},${connected.apis.id},${business.dn}</arg>
  </function>
```

App Version Workflow: Conditions

The following conditions apply to the app version workflow:

- isAppTeamMemberUserLeaderOfAnyOtherGroup on page 26
- atleastOneValidAPIContractInEnvironment on page 27
- allAPIContractsInEnvironmentApproved on page 27
- existAPIContractsForAllAPIsInEnvironments on page 28

isAppTeamMemberUserLeaderOfAnyOtherGroup

Tests to see if the user is an app team member and is also a leader of at least one other 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="${caller}">
      <conditions type="AND">
<condition type="atleastOneValidAPIContractInEnvironment">
  <arg name="Environment">Sandbox</arg>
</condition>
      </conditions>
    </result>
    <unconditional-result old-status="Sandbox" status="Sandbox" step="200" owner="${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. Values: 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: Variable Resolvers

The following variable resolvers are available for the app version workflow:

- ${app.dn} on page 29
- ${app.team.group.dn} on page 29
- ${app.version.dn} on page 29
- ${app.version.name} on page 29
- ${business.dn} on page 29
- ${connected.apis.admin.groups} on page 30
- ${connected.apiversion.ids} on page 30
- ${connected.apis.id} on page 30

${app.dn}

The unique ID for the app, 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}**.

${app.version.dn}

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

${app.version.name}

The text name for the app version, as a string in the format **${app.version.name}**.

${business.dn}

The unique ID for the business, a string in the format **${business.dn}**.

${connected.apis.admin.groups}

The unique IDs for the API Admin groups of any APIs connected to a specific app, in the format **${connected.apis.admin.groups}**.

${connected.apiversion.ids}

The unique API Version IDs for all API versions connected to a specific app, in the format **${connected.apiversion.ids}**.

${connected.apis.id}

The unique API IDs for all APIs connected to a specific app, in the format **${connected.apis.ids}**.

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 31
- exportAPIAllVersions on page 32

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: Variable Resolvers

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 Contract Workflow

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 on page 34
- sendNotification on page 35
- synchronizeAppVersion on page 36
- invokeAppVersionAction on page 36

- invokeApiVersionAction on page 37
- updateContractActiveStatus on page 37

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 based on an event relating to an API contract.

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 on page 38
- isAtmosphereSandboxApiContract on page 39
- isAtmosphereProductionApiContract on page 39
- isAtmosphereSandboxAutoApprove on page 39
- isAtmosphereProductionAutoApprove on page 39
- isRemoteFedMemberApp on page 39
- apiContractUsesRestrictedScope on page 39
- apiVersionSupportsResourceLevelPermissions on page 40
- isAPIContractScopeNotEmpty on page 40
- checkAPIVersionValidWFAction on page 40
- checkAppVersionValidWFAction on page 40
- checkAppVersionStateMatches on page 40
- checkAppVersionFedMemberMatches on page 40

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: Variable Resolvers

The following variables are available for the API contract workflow:

- ${contract.api.dn} on page 41
- ${contract.api.version.dn} on page 41
- ${contract.app.dn} on page 41
- ${contract.app.version.dn} on page 41
- ${contract.dn} on page 41
- ${contract.state} on page 41
- ${contract.old.state} on page 42

${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

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: Variable Resolvers

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 45
- setGroupMembershipRole on page 47
- sendGroupMembershipNotification on page 48

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 on page 50
- isCallerSiteAdmin on page 51
- isCallerGroupAdmin on page 52
- isCallerGroupAdminMember on page 53
- isCallerGroupLeader on page 53
- isCallerGroupMember on page 54

- isMemberMembership on page 54
- isLeaderMembership on page 56
- isAdminMembership on page 57

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: Variable Resolvers

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.

User Workflow

This section provides information about functions, conditions, and variable resolvers available for the user workflow, as well as initial actions and reserved actions.

User Workflow: Reserved Actions

The following reserved actions are defined for user workflows:

- @Add

  Used when the Site Admin adds a user using the UserAPI.addUser operation (see POST /api/users on docs.soa.com).

  The user is automatically registered before this initial action is added. This action does not affect the user's registered state, but can be used for post-functions such as sending notifications.

- @AgreementsAccepted

  Indicates that any required platform legal agreement was accepted by the user. This is only used during the login process.

- @ChallengeQuestionsAnswered

  Indicates that the user has provided answers to any required security challenge questions.

  This reserved action is a hook that you can use to extend the workflow with additional actions such as generating a notification and/or Board item, or initiating a back-end process.

- @ForcedPasswordChanged

  Indicates that the user has changed the password. Invoked when the user changes the password as the result of a change password requirement.

  This reserved action is a hook that you can use to extend the workflow with additional actions such as generating a notification and/or Board item or initiating a back-end process.

- @Invite

  Checks the user settings to determine whether inviting a non-platform user to a team or group is allowed or not. The default workflow uses this setting to decide whether this initial action is allowed.

  If you are creating a custom user workflow, include this initial action if you want to allow inviting users that are not registered in the platform. You can include one or more additional conditions as needed.

- @Login

  This reserved action is invoked whenever the user tries to log in.

  Note: this is currently used only for platform users, not third-party domain users.

When the user logs in, the workflow determines the status value returned in the login response—which then determines the next step. If there is a pending task to be performed, these are prompted before login is complete. The possible pending tasks for @Login are:

- **ChangePassword**: forces the user to change password as part of the login process.. In the default workflow, this is applicable when the Site Admin has provided a temporary password for initial user login.
- **ForceAcceptAgreements**: forces the user to accept the platform legal agreement as part of the login process.
- **securityQuestionsAnswered**: forces the user to provide answers for security challenge questions as part of the login process.. The number required depends on platform security settings.

As part of creating a default workflow, you could create additional pending tasks. However, you would also have to develop the UI to support those tasks so that users could log in via the UI.

- @ModifyProfile

  Used to determine whether the platform settings allow the user to modify the user profile.

  In the default platform workflow, this is how it is used:

  - If users do self-signup, they can modify the profile or not based on this setting.
  - If users are added by administrators, they cannot modify the profile unless they are site administrators.

- @PasswordChanged

  Indicates that the user has changed the password. Invoked when the user voluntarily changes the password via the Profile > Password page.

  This reserved action is a hook that you can use to extend the workflow with additional actions such as generating a notification and/or Board item or initiating a back-end process.

- @Setup

  Used in an upgrade scenario, where existing users are added to the scenario because an upgraded installation requires all users to be in the workflow. During the upgrade process, when the **Upgrade CM Models** admin action is run, this reserved action is used to add all registered users to the workflow. Currently this is applicable only to platform users, not third-party users.

- @Signup

  Used to determine whether the platform settings allow the user to sign up to the platform (user-initiated signup).

  The UsersAPI.signupUser operation (see POST /api/users/signupUser[/{InvitationCode}] on docs.soa.com) uses the availability of this initial action to either reject the message request or proceed with the signup.

  The platform user interface references the user settings to determine whether signup is allowed or not. The default workflow allows or rejects signup based on the UI setting.

User Workflow: Functions

The following functions are available for the user workflow:

- MarkUserPermanent on page 62
- markUserPermanentIfFirstLogin on page 62
- sendNotification on page 63
- setProperty on page 63

MarkUserPermanent

This function marks the user as permanent in the database. Until a user logs in for the first time, the user is not marked as permanent; the user's invitation expires after a pre-set time period, and invited users who did not complete the process are periodically purged from the database. This function marks the record as permanent so that it does not get purged.

Note: Two separate functions exist, **MarkUserPermanent** and **markUserPermanentIfFirstLogin**, to offer flexibility in implementation. In the out-of-the box user workflow there is no difference in implementation between these two, but a custom workflow could be designed to differentiate between a user's first login and subsequent logins. For example, a custom workflow could require users to renew the account every year, and could enforce the account renewal process.

markUserPermanentIfFirstLogin

If this is the first login by the user, this function marks the user as permanent in the database. Until a user logs in for the first time, the user is not marked as permanent; the user's invitation expires after a pre-set time period, and invited users who did not complete the process are periodically purged from the database. This function marks the record as permanent so that it does not get purged.

The default user workflow uses this function to mark the user as permanent when the user logs in for the first time.

Note: Two separate functions exist, **MarkUserPermanent** and **markUserPermanentIfFirstLogin**, to offer flexibility in implementation. In the out-of-the box user workflow there is no difference in implementation between these two, but a custom workflow could be designed to differentiate between a user's first login and subsequent logins. For example, a custom workflow could require users to renew the account every year, and could enforce the account renewal process.

Examples/Notes/Additional Information

In the example below, the user has completed the registration process. LoginState is set to LoginComplete and the user is therefore marked as permanent. The notification is currently commented out.

```
<unconditional-result old-status="registered" status="registered" step="400">
  <pre-functions>
    <function type="setProperty">
      <arg name="LoginState">&LoginComplete;</arg>
    </function>
    <function type="markUserPermanentIfFirstLogin"/>
```

```
        <!--  invoke send Notification on first time login.  -->
      </pre-functions>
    </unconditional-result>
```

sendNotification

Triggers the specified email/Dashboard notification based on an event relating to a user.

Parameters

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.user.admin.added
role	The role to which the notifications will be sent. The only valid value is Self.

Examples/Notes/Additional Information

In the example below, a notification is sent as a post-function when the Site Admin adds a user.

```
<step id="20" name="Route Admin Add" >
  <actions>
    <action id="21" name="init-admin-add" auto="TRUE">
      <results>
        <result old-status="none" status="registered" step="400" >
          <conditions type="AND">
            <!-- <condition type="requiredProfileExists" /> -->
            <condition type="isLocalDomainUser"/>
          </conditions>
          <!-- <pre-functions>
            <function type="updateUserStatus">
              <arg name="status">registered</arg>
            </function>
    </pre-functions> -->
    <post-functions>
            <function type="sendNotification">
              <arg name="notificationType">com.soa.notification.type.user.admin.added</arg>
              <arg name="role">Self</arg>
            </function>
          </post-functions>
    </result>
    <unconditional-result old-status="none" status="unknown" step="-1"/>
    </results>
  </action>
</actions>
</step>
```

setProperty

Sets any property defined by the workflow, allowing the workflow to communicate back to the application or to the response stream by setting a property based on the workflow. The property can be used inside the workflow to give feedback to the application and thus guide the process flow.

setProperty can be used to set any property in any workflow document.

For example, in the default user workflow, the @Login reserved action, invoked at login, uses setProperty to determine the next step—based on the workflow and existing conditions/information, whether the login action is complete or whether one or more required actions must be completed first.

In this scenario, the argument is PendingTask and there are three possible values (see User Workflow: Reserved Actions on page 60).

Examples/Notes/Additional Information

In the example below, this function is used to evaluate whether the user logging in is a local domain user and, if so, whether a password change is required. If a password change is required, it is set as a pending task that must be completed prior to login.

```
<step id="400" name="managed">
  <actions>
    <action id="401" name="@Login">
      <results>
        <result old-status="registered" status="registered" step="400">
          <conditions type="AND">
            <condition type="isLocalDomainUser"/>
            <condition type="isChangePasswordRequired"/>
          </conditions>
          <pre-functions>
            <function type="setProperty">
              <arg name="PendingTask">&ChangePassword;</arg>
            </function>
          </pre-functions>
```

User Workflow: Conditions

The following conditions apply to the user workflow:

- isLocalDomainUser on page 65
- IsRegisteredUser on page 65
- IsLocalRegisteredUser on page 65
- IsLastLoginEmpty on page 65
- IsChangePasswordRequired on page 65
- AgreementsAccepted on page 65
- IsForceChallengeQuestionesAnsweredOnLoginSetup on page 65
- SecurityQuestionsAnswered on page 65
- IsSelfSignupAllowed on page 65
- UserSettingsAllowModifyProfile on page 65
- IsInviteUnRegisteredUserAllowed on page 66
- authorizeSelf on page 66

isLocalDomainUser

Tests to see if the user is a local domain user.

IsRegisteredUser

Tests to see that the user is a registered platform user.

IsLocalRegisteredUser

Tests to see that the user is a registered platform user with a local account.

IsLastLoginEmpty

Tests to see whether this is the first login.

IsChangePasswordRequired

Tests to see if a change of password for the user is required; for example, at first login when the Site Admin has provided the user with a temporary password.

AgreementsAccepted

Tests to see whether the user has accepted the platform legal agreement, if required.

IsForceChallengeQuestionesAnsweredOnLoginSetup

Tests to see whether the platform user setting, **EnforceChallengesSetupOnLogin**, is enabled. If the setting is enabled, the user must provide answers to the challenge questions as part of login.

SecurityQuestionsAnswered

Tests to see if the user has provided answers to the required number of security questions.

IsSelfSignupAllowed

Tests to see whether the platform user setting, **SelfSignup**, is enabled. If the setting is disabled, the user cannot sign themselves up for the platform.

UserSettingsAllowModifyProfile

Tests to see whether the platform business security setting, **UserModifyEmail**, is enabled. If the setting is disabled, the user cannot modify his/her profile.

IsInviteUnRegisteredUserAllowed

Tests to see whether the platform user setting, **InviteUnregisteredUsers**, is enabled; if so, returns **true**. If the setting is disabled, group members cannot invite unregistered users to join platform groups or teams.

authorizeSelf

Tests to see whether the user running this action is the user whose workflow document is being used. For example, it is used at login by default. It is also used for modify profile. If the Site Admin is modifying the user's profile, authorizeSelf is **false**. If the user is modifying his/her own profile, it is **true**.

User Workflow: Variable Resolvers

There are no variable resolvers for the user workflow.

Review Workflow

This section provides information about functions, conditions, and variable resolvers for the review workflow, as well as initial actions and reserved actions.

Review Workflow: Initial Actions

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

- @StartReview

  This is how the reviews get introduced into the workflow. If you are customizing the initial behavior when a review is added, this is where the customization needs to go.

Reviews Workflow: Reserved Actions

The following reserved actions are defined for Community Manager workflows relating to reviews:

- @read

  Applies to unpublished reviews only.

  This action controls who can read a review. When the @read action is available, read permission is available for someone to read the review in an unpublished state. If @read is not available for a specific user for the current state of the review, the user cannot see it.

  Once the review is published, all users who have visibility of the resource can see it.

- @modify

  Used to determine whether a review can be modified.

- @cancel

  Used to cancel the review and end the workflow.

Reviews Workflow: Functions

The following functions are available for Community Manager workflows relating to reviews:

- markPublished on page 67
- markUnPublished on page 68
- deleteReview on page 69
- cancelOldReviewForTheSubjectBySameUser on page 70
- sendNotification on page 70

markPublished

Marks a review as published. This might be used in a scenario where the Moderator has determined that the review meets guidelines, or if reviews are not moderated.

Parameters

None.

Examples/Notes/Additional Information

In the example below, the workflow is set up so that a review added by an administrator is approved automatically. The workflow checks that the user has a role of Site Admin or Business Admin. If so, the status of the review is automatically changed to Published. The markPublished function is invoked as a post-function.

```
<step id="100" name="Route Add New Review">
 <actions>
  <action id="101" name="Auto Approve Add" auto="TRUE">
   <restrict-to>
    <conditions type="OR">
     <condition type="authorizeByAtmosphereRole">
      <arg name="role">SiteAdmin,BusinessAdmin</arg>
     </condition>
     <condition type="isAutoPublishEnabled"/>
    </conditions>
   </restrict-to>
   <results>
    <unconditional-result old-status="none"  status="Published" step="300" owner="${caller}" />
   </results>
   <post-functions>
    <function type="markPublished"/>
    <function type="sendNotification">
     <arg name="subjectType">apiversion</arg>
     <arg name="role">SubjectAdmin</arg>
     <arg name="notificationType">com.soa.notification.type.api.review.created</arg>
    </function>
    <function type="sendNotification">
     <arg name="subjectType">app-version</arg>
     <arg name="role">SubjectAdmin</arg>
     <arg name="notificationType">com.soa.notification.type.app.review.created</arg>
    </function>
   </post-functions>
  </action>
```

markUnPublished

Marks a review as not published. This might be used in a scenario where the Moderator has determined that the review does not meet guidelines.

Parameters

None.

Examples/Notes/Additional Information

In the example below, an existing, published review is being cancelled because another review on the same subject by the same user was received. The workflow checks that the action is performed by an authorized user, and then changes the status of the original review from Approved to Pending. The markUnPublished function is invoked as a post-function.

```
<step id="300" name="Published">
  <pre-functions>
    <function type="cancelOldReviewForTheSubjectBySameUser" />
  </pre-functions>
  <actions>
    <action id="301" name="review.action.unpublish">
      <restrict-to>
        <conditions type="AND">
          <condition type="authorizeByAtmosphereRole">
            <arg name="role">SiteAdmin,BusinessAdmin,Author,SubjectAssociatedApiAdmin</arg>
          </condition>
        </conditions>
      </restrict-to>
      <results>
        <unconditional-result old-status="Approved" status="Pending" step="200" owner="${caller}" />
      </results>
      <post-functions>
        <function type="markUnPublished"/>
      </post-functions>
    </action>
```

deleteReview

Deletes a review.

Parameters

None.

Examples/Notes/Additional Information

In the example below, the @cancel reserved action is invoked. The workflow first checks that the user is authorized to perform the action. If so, the review is moved from a Published status to a Finished status. As a post-function, the review is deleted.

```
<action id="303" name="@cancel">
  <restrict-to>
    <conditions type="AND">
      <condition type="authorizeByAtmosphereRole">
        <arg name="role">SiteAdmin,BusinessAdmin,SubjectAssociatedApiAdmin</arg>
      </condition>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Published" status="Finished" step="500" owner="${caller}" />
  </results>
  <post-functions>
    <function type="deleteReview"/>
  </post-functions>
</action>
```

cancelOldReviewForTheSubjectBySameUser

Cancels an existing review. This function is generally used in a scenario where a reviewer submits a second review on the same subject. The subsequent review replaces the earlier one, which is cancelled.

When someone publishes a second review for the same subject, the platform does not modify the existing review that was already published. Instead, a new review is created. At that point, there might be two reviews on the same subject by the same author, one in the Published state and one in the Draft state. When the draft review is published, the previous review is automatically deleted.

The states for both reviews are as follows:

- First review: published / Second review: draft
- Second review: published / first review: deleted.

Parameters

None.

Examples/Notes/Additional Information

In the example below, a review is published. As a pre-function, the workflow checks for an old review for the same subject by the same user, and cancels it if found. It first checks that the user is authorized to perform the action, changes the old review status from Approved to Pending, and runs the markUnPublished function.

```
<step id="300" name="Published">
  <pre-functions>
    <function type="cancelOldReviewForTheSubjectBySameUser" />
  </pre-functions>
  <actions>
    <action id="301" name="review.action.unpublish">
      <restrict-to>
        <conditions type="AND">
          <condition type="authorizeByAtmosphereRole">
            <arg name="role">SiteAdmin,BusinessAdmin,Author,SubjectAssociatedApiAdmin</arg>
          </condition>
        </conditions>
      </restrict-to>
      <results>
        <unconditional-result old-status="Approved" status="Pending" step="200" owner="${caller}" />
      </results>
      <post-functions>
        <function type="markUnPublished"/>
      </post-functions>
    </action>
```

sendNotification

Triggers the specified notification based on an event relating to a review.

Parameters

Name	Description/Values
subjectType	Indicates the type of resource the notification relates to. Applicable to anywhere a review can be added. Valid values: • apiversion • app-version • group
Role	The role to which the notifications will be sent—only users who hold this role for the type of resource as specified in the subjectType parameter. Valid values: • ApiAdmin • AppAdmin
notificationType	The type of notification being sent. Can be any valid notification existing in the platform. For example: • com.soa.notification.type.api.review.created (for an API) • com.soa.notification.type.app.review.created (for an app)

Examples/Notes/Additional Information

In the example below, the workflow is set up so that a review added by an administrator is approved automatically. When the review is published, a notification is sent to applicable admins. A different notification is sent for an API review versus an app review.

```
<step id="100" name="Route Add New Review">
  <actions>
    <action id="101" name="Auto Approve Add" auto="TRUE">
      <restrict-to>
        <conditions type="OR">
          <condition type="authorizeByAtmosphereRole">
            <arg name="role">SiteAdmin,BusinessAdmin</arg>
          </condition>
          <condition type="isAutoPublishEnabled"/>
        </conditions>
      </restrict-to>
      <results>
        <unconditional-result old-status="none"  status="Published" step="300" owner="${caller}" />
      </results>
      <post-functions>
        <function type="markPublished"/>
        <function type="sendNotification">
          <arg name="subjectType">apiversion</arg>
          <arg name="role">SubjectAdmin</arg>
          <arg name="notificationType">com.soa.notification.type.api.review.created</arg>
        </function>
        <function type="sendNotification">
          <arg name="subjectType">app-version</arg>
          <arg name="role">SubjectAdmin</arg>
          <arg name="notificationType">com.soa.notification.type.app.review.created</arg>
        </function>
      </post-functions>
    </action>
```

Reviews Workflow: Conditions

The following conditions apply to the reviews workflow:

- isAutoPublishEnabled

isAutoPublishEnabled

Checks whether a review can be automatically published; returns **true** if so. If **false**, new reviews require moderation.

Reviews Workflow: Variable Resolvers

There are currently no variable resolvers for the Community Manager workflows relating to reviews.

Discussions Workflow

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

Discussions Workflow: Initial Actions

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

- @StartDiscussion

  This is how a discussion is introduced into the workflow. If you are customizing the initial behavior when a discussion is added, this is where the customization needs to go.

- @Audit

  An initial action that could be used to initiate a discussion for audit purposes.

- @Alert

  An initial action that could be used to initiate a discussion as the result of an alert—for example, SLA or quota alerts.

Discussions Workflow: Reserved Actions

The following reserved actions are defined for Community Manager workflows relating to discussions:

- @read

  Applies to unpublished discussions only.

  This action controls who can read a discussion. When the @read action is available, read permission is available for someone to read the discussion in an unpublished state. If @read is not available for a specific user for the current state of the discussion, the user cannot see it.

  Once the discussion is published, all users who have visibility of the resource can see it.

Discussions Workflow: Functions

The following functions are available for Community Manager workflows relating to discussions:

- MarkPublished on page 73
- markUnPublished on page 74
- deleteDiscussion on page 75
- sendNotification on page 75

MarkPublished

Marks a discussion as published. This might be used in a scenario where the Moderator has determined that the discussion meets guidelines, or if discussions are not moderated.

Parameters

None.

Examples/Notes/Additional Information

The example below shows part of the workflow for a pending discussion. It can be approved for publication; the workflow restricts the action to certain authorized roles. The status is changed from Pending to Published, and the markPublished function is run.

```
<step id="200" name="Pending">
  <actions>
    <action id="201" name="discussion.action.approve.publish">
      <restrict-to>
        <conditions type="OR">
          <condition type="authorizeByAtmosphereRole">
            <arg name="role">SiteAdmin,BusinessAdmin,SubjectAssociatedApiAdmin</arg>
          </condition>
        </conditions>
      </restrict-to>
      <results>
        <unconditional-result old-status="Pending" status="Published" step="300" owner="${caller}" />
      </results>
      <post-functions>
        <function type="markPublished"/>
      </post-functions>
    </action>
```

markUnPublished

Marks a discussion as not published. This might be used in a scenario where the Moderator has determined that the discussion does not meet guidelines.

Parameters

None.

Examples/Notes/Additional Information

The example below defines valid actions for a discussion in the Published state. One valid action is for the discussion to be unpublished. The workflow first checks that the action is being performed by an authorized user. If so, the discussion is moved from an Approved state to Pending, and the **markUnPublished** function is run as a post-function.

```
<step id="300" name="Published">
  <actions>
    <action id="301" name="discussion.action.unpublish">
      <restrict-to>
        <conditions type="AND">
          <condition type="authorizeByAtmosphereRole">
            <arg name="role">SiteAdmin,BusinessAdmin,Author,SubjectAssociatedApiAdmin</arg>
          </condition>
        </conditions>
      </restrict-to>
      <results>
        <unconditional-result old-status="Approved" status="Pending" step="200" owner="${caller}" />
      </results>
      <post-functions>
        <function type="markUnPublished"/>
      </post-functions>
    </action>
```

deleteDiscussion

Deletes a discussion.

Parameters

None.

Examples/Notes/Additional Information

In the example below, for a discussion in the Rejected state, delete is a valid action. The workflow first checks that the user is authorized to perform the action. If so, the discussion is moved from the Rejected state to the Finished state, and the **deleteDiscussion** function is run.

```
<step id="400" name="Rejected">
  <actions>
    <action id="402" name="discussion.action.delete">
      <restrict-to>
        <conditions type="AND">
          <condition type="authorizeByAtmosphereRole">
            <arg name="role">SiteAdmin,BusinessAdmin,Author,SubjectAssociatedApiAdmin</arg>
          </condition>
        </conditions>
      </restrict-to>
      <results>
        <unconditional-result old-status="Rejected" status="Finished" step="500" owner="${caller}" />
      </results>
      <post-functions>
        <function type="deleteDiscussion"/>
      </post-functions>
    </action>
```

sendNotification

Triggers the specified notification based on an event relating to a discussion.

Parameters

Name	Description/Values
subjectType	Indicates the type of resource the notification relates to. Valid values: • apiversion • app-version • group • board
Role	The role to which the notifications will be sent—only users who hold this role for the type of resource as specified in the subjectType parameter. Valid values: • SubjectAdmin
notificationType	The type of notification being sent. Can be any valid notification existing in the platform. For example: • com.soa.notification.type.api.post.created (for an API) • com.soa.notification.type.app.post.created (for an app) • com.soa.notification.type.group.post.created (for a group) • com.soa.notification.type.board.post.created (for a board)

Examples/Notes/Additional Information

In the example below, the workflow is set up so that a review added by an administrator is approved automatically. When the review is published, a notification is sent to applicable admins. A different notification is sent for an API review versus an app review.

```
<action id="102" name="Auto Approve Auto Publish" auto="TRUE">
  <restrict-to>
    <conditions type="OR">
      <condition type="authorizeByAtmosphereRole">
        <arg name="role">SiteAdmin,BusinessAdmin</arg>
      </condition>
      <condition type="isAutoPublishEnabled"/>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="none"  status="Published" step="300" owner="${caller}" />
  </results>
  <post-functions>
    <function type="markPublished"/>
    <function type="sendNotification">
      <arg name="subjectType">apiversion</arg>
      <arg name="role">SubjectAdmin</arg>
      <arg name="notificationType">com.soa.notification.type.api.post.created</arg>
    </function>
    <function type="sendNotification">
      <arg name="subjectType">app-version</arg>
      <arg name="role">SubjectAdmin</arg>
      <arg name="notificationType">com.soa.notification.type.app.post.created</arg>
    </function>
    <function type="sendNotification">
      <arg name="subjectType">group</arg>
      <arg name="role">SubjectAdmin</arg>
      <arg name="notificationType">com.soa.notification.type.group.post.created</arg>
    </function>
```

```
      <function type="sendNotification">
        <arg name="subjectType">board</arg>
        <arg name="role">SubjectAdmin</arg>
        <arg name="notificationType">com.soa.notification.type.board.post.created</arg>
      </function>
    </post-functions>
  </action>
```

Discussions Workflow: Conditions

The following conditions apply to the discussions workflow:

- isAutoPublishEnabled

isAutoPublishEnabled

Checks whether a discussion can be automatically published; returns **true** if so. If **false**, new discussions require moderation.

Arguments

None.

Discussions Workflow: Variable Resolvers

There are currently no variable resolvers for the Community Manager workflows relating to discussions.

Comments Workflow

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

Comments Workflow: Initial Actions

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

- @AddComment

  This is how a comment is introduced into the workflow. If you are customizing the initial behavior when a comment is added, this is where the customization needs to go.

- @Audit

  An initial action that could be used to initiate a comment for audit purposes.

- @Alert

  An initial action that could be used to initiate a comment as the result of an alert—for example, SLA or quota alerts.

Comments Workflow: Reserved Actions

The following reserved actions are defined for comments workflows:

- @read

  Applies to unpublished comments only.

  This action controls who can read a comment. When the @read action is available, read permission is available for someone to read the comment in an unpublished state. If @read is not available for a specific user for the current state of the comment, the user cannot see it.

  Once the comment is published, all users who have visibility of the resource can see it.

Comments Workflow: Functions

The following functions are available for the comments workflow:

- MarkPublished on page 79
- markUnPublished on page 79
- deleteComment on page 80

MarkPublished

Marks a comment as published. This might be used in a scenario where the Moderator has determined that the comment meets guidelines, or if comments are not moderated.

Parameters

None.

Examples/Notes/Additional Information

In the example below, when a new comment is added, the workflow checks the platform settings to see if a comment added by an admin is auto-approved, and also checks the role of the user. If both conditions are met, the status is changed from **none** to **Published**, and the workflow runs the **markPublished** function.

```
<step id="100" name="Route Add New Comment">
  <actions>
    <action id="101" name="Auto Approve Admin Add" auto="TRUE">
      <restrict-to>
        <conditions type="AND">
          <condition type="authorizeByCMRole">
            <arg name="role">SiteAdmin,BusinessAdmin</arg>
          </condition>
        </conditions>
      </restrict-to>
      <results>
        <unconditional-result old-status="none"  status="Published" step="300" owner="${caller}" />
      </results>
      <post-functions>
        <function type="markPublished"/>
      </post-functions>
    </action>
```

markUnPublished

Marks a comment as not published. This might be used in a moderated scenario where the Moderator has determined that the comment does not meet guidelines.

Parameters

None.

Examples/Notes/Additional Information

In the example below, when a new comment is added, the workflow checks the platform settings to see if a comment added by an admin is auto-approved. If manual approval is needed (action id 102), the status is changed from **none** to **Pending**, and the workflow runs the **markUnPublished** function.

```
<step id="100" name="Route Add New Comment">
  <actions>
```

```
    <action id="101" name="Auto Approve Admin Add" auto="TRUE">
      <restrict-to>
        <conditions type="AND">
          <condition type="authorizeByCMRole">
            <arg name="role">SiteAdmin,BusinessAdmin</arg>
          </condition>
        </conditions>
      </restrict-to>
      <results>
        <unconditional-result old-status="none"  status="Published" step="300" owner="${caller}" />
      </results>
      <post-functions>
        <function type="markPublished"/>
      </post-functions>
    </action>
    <action id="102" name="Manual Approval Needed" auto="TRUE">
      <results>
        <unconditional-result old-status="none" status="Pending" step="200" owner="${caller}" />
      </results>
      <post-functions>
        <function type="markUnPublished"/>
      </post-functions>
    </action>
  </actions>
</step>
```

deleteComment

Deletes a comment.

Parameters

None.

Examples/Notes/Additional Information

In the example below, an action is taken the delete the comment. The workflow first verifies that the user is authorized. If the condition is met, the status is changed from **Published** to **Finished**, and the workflow runs the **deleteComment** function.

```
<action id="302" name="comment.action.delete">
  <restrict-to>
    <conditions type="AND">
      <condition type="authorizeByCMRole">
        <arg name="role">SiteAdmin,BusinessAdmin,Author,SubjectAssociatedApiAdmin</arg>
      </condition>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="Published" status="Finished" step="500" owner="${caller}" />
  </results>
  <post-functions>
    <function type="deleteComment"/>
  </post-functions>
```

```
</action>
```

Comments Workflow: Conditions

The following conditions apply to the comments workflow:

- authorizeByCMRole
- isCommentAutoPublishEnabled

authorizeByCMRole

Tests to see if the workflow user has one or more specific roles in the platform, and is therefore authorized to perform the workflow action; returns Boolean true or false.

Arguments

Name	Description/Values
Role	One or more roles that are authorized to perform the workflow action. Valid values: • Author • BusinessAdmin • SiteAdmin • SubjectAssociatedApiAdmin

Examples/Notes/Additional Information

In the example below, when a new comment is added, the workflow checks the platform settings to see if a comment added by an admin is auto-approved, and then checks the role of the user. If the user is a Site Admin or Business Admin, the workflow proceeds. If not, the action is not allowed.

```
<step id="100" name="Route Add New Comment">
  <actions>
    <action id="101" name="Auto Approve Admin Add" auto="TRUE">
      <restrict-to>
        <conditions type="AND">
          <condition type="authorizeByCMRole">
            <arg name="role">SiteAdmin,BusinessAdmin</arg>
          </condition>
```

isCommentAutoPublishEnabled

Checks whether a comment can be automatically published; returns **true** if so. If **false**, new comments require moderation.

Arguments

None.

Comments Workflow: Variable Resolvers

There are currently no variable resolvers for the Community Manager workflows relating to comments.

–⧫–