Akana API Platform: Customizing Workflows

Version 8.0

Akana API Platform

Customizing Workflows
Version 8.0
January, 2016
Doc version 08-00-01

Copyright

Trademarks

Akana, Inc. (formerly SOA Software, Inc.)

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

Disclaimer

Contents

Chapter 1 | Akana API Platform 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.

The Akana API Platform 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 12—the workflows you can customize, how to get started, and how to implement a custom workflow on the platform.
- Workflow General Reference: Functions, Conditions, and Variables on page 20—the technical details about components that are common to all workflows for the developer portal.
- Individual chapters with specific information about the functions, conditions, and variables relating to workflow for a specific resource, such as app version, contract, or discussion.

Note: The Akana API Platform 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 the Akana API Platform:

- 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, the action can extend the default behavior in the platform. For example, you could set up the group membership workflow so that if a group is deleted all members receive a notification.
  - In other cases, a workflow action can replace the default behavior. 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.
- **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 37
- App: see API Version Workflow on page 52
- API Contract: see API Contract Workflow on page 55
- Group Membership: see Group Membership Workflow on page 69
- Ticket: see Ticket Workflow on page 67
- User: see User Workflow on page 91
- Reviews: see Review Workflow on page 122
- Discussions: see Discussions Workflow on page 129
- Comments: see Comments Workflow on page 135

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 14.

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 16. (**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 15. (**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 14. (**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.akana.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 the developer portal 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 Akana API Platform workflows, 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 ***Limit* actions**: 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.

   A common use of this type of reserved actions is to check for authorization. For example, a reserved action could check the user's role to make sure the user is a Site Admin or Business Admin before allowing a specific action.

2 ***Extend* behavior**: 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.

As an extension of #2, in some cases custom workflow can be used to ***replace*** default behavior. However, in most cases custom workflows extend the default behavior rather than replacing it.

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 69.

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 General 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.

Functions, Conditions, and Variables

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 for use with Akana API Platform custom workflows. There is also a set of general use functions, conditions and variables for use specifically with SOA Software Policy Manager workflows. Those are also valid for Akana API Platform 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>
```

The following general use functions are available for workflow in the Akana API Platform:

setAuthTokenProperty

Available in version: 7.2.4.3 and later

Used to specify a custom property for the auth token cookie.

Parameters

Name	Description/Values
PropertyName	The name of a specific property in the auth token cookie.
PropertyValue	The value being set for the property.

Examples/Notes/Additional Information

In the example below, the property name **2FAComplete** is set to a value of **Yes**.

```
<function type="setAuthTokenProperty">
  <arg name="PropertyName">2FAComplete</arg>
  <arg name="PropertyValue">Yes</arg>
</function>
```

removeAuthTokenProperty

Available in version: 7.2.4.3 and later

Used to remove a custom property from the auth token cookie.

Parameters

Name	Description/Values
PropertyName	The name of the property.

Examples/Notes/Additional Information

In the example below, the first function removes the existing property and the second one sets a new property, 2FA Complete, with a value of Yes.

```
<function type="removeAuthTokenProperty">
  <arg name="PropertyName">2FAData</arg>
</function>
<function type="setAuthTokenProperty">
  <arg name="PropertyName">2FAComplete</arg>
  <arg name="PropertyValue">Yes</arg>
</function>
```

markLoginComplete

Available in version: 7.2.4.3 and later

Marks the user's login action as fully complete, with no pending login tasks. This is used at the end of the default user workflow, after the user has been guided through all tasks relating to login, such as accepting a legal agreement or specifying the answers to security challenge questions.

Parameters

None.

Examples/Notes/Additional Information

In the example below, the workflow sets the LoginState to LoginComplete and then runs the markLoginComplete function.

```
<unconditional-result old-status="registered" status="registered" step="450">
  <pre-functions>
    <function type="setProperty">
      <arg name="LoginState">&LoginComplete;</arg>
    </function>
    <!--  invoke send Notification on first time login.  -->
    <function type="markLoginComplete"/>
  </pre-functions>
</unconditional-result>
```

setArgumentValue

Available in version: 7.2.4.3 and later

Creates an argument with a specific value. Use this function to make a specific argument available within the workflow execution.

Parameters

Name	Description/Values
ArgName	The name of the argument.
Value	The value of the argument.

setCookie

Available in version: 7.2.4.3 and later

Sets a cookie with a specific name, value, and expiration date/time.

Parameters

Name	Description/Values
CookieName	The name of the cookie.

CookieValue	The value of the cookie.
CookieExpirationTimeMillis	The expiration time of the cookie, in milliseconds. If not defined, the default is 1500000 (25 minutes).

Examples/Notes/Additional Information

The example below sets the value and expiration time of the D cookie.

```
<function type="setCookie">
  <arg name="CookieName">D</arg>
  <arg name="CookieValue">akana api platform</arg>
  <arg name="CookieExpirationTimeMillis">31556952000</arg>
</function>
```

removeCookie

Available in version: 7.2.4.3 and later

Removes the specified cookie.

Parameters

Name	Description/Values
CookieName	The name of the cookie.

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">.

The following general use conditions are available in developing all types of custom workflows on the platform:

authorizeByAtmosphereRole

Available in version: 7.1 and later

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 • Admin (administrator of context object: applies to App, App version, API, API version, and Group)

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>
```

authorizeByAtmosphereAction

Available in version: 7.1 and later

Tests to see if the workflow user has permission to perform a specific action on a specific type of resource.

Arguments

Name	Description/Values
Action	The specific action. Valid values: • Add
ResourceType	The specific resource. Valid values: • app • api • group

Examples/Notes/Additional Information

In the example below, the workflow tests that the current user has permission to add an API. If the user does not meet the condition, the action is not allowed.

```
<action id="409" name="@AddAPI">
  <restrict-to>
    <conditions type="AND">
      <condition type="authorizeByAtmosphereAction">
        <arg name="Action">Add</arg>
      <arg name="ResourceType">api</arg>
      </condition>
    </conditions>
  </restrict-to>
```

authorizeByDomain

Available in version: 7.2.4.2 and later

Tests to see if the domain of the workflow user matches one of the domains specified, and the user is therefore authorized to perform the workflow action; if so, returns **true**.

This can be used as a security measure. For example, actions can be limited to one specific domain for security purposes.

Arguments

Name	Description/Values
domain	One or more domains that authorization is restricted to. To include multiple values, you can either include multiple <domain> arguments or list multiple domains on one line, separated by commas. **Note**: the value for this parameter should be the domain name used in Policy Manager for the applicable identity system. For Policy Manager, use **Local Domain** as the value.

Examples/Notes/Additional Information

In the example below, this condition specifies that an action is valid only for users on the platform's domain, acmepaymentscorp.

```
<condition type="authorizeByDomain">
  <arg name="domain">acmepaymentscorp</arg>
</condition>
```

authorizeByDomainType

Available in version: 7.2.4.2 and later

Tests to see if the domain type of the workflow user matches one or more specified values. If so, returns **true**.

This can be used as a security measure to limit platform actions to a specific set of authorized users. For example, a workflow action can be limited to one specific domain type, such as an LDAP domain, for security purposes.

Arguments

Name	Description/Values
DomainType	One or more domain types (Identity System type in Policy Manager, Domain Type in the developer portal), that the action is restricted to. To include multiple values, you can either include multiple <DomainType> arguments or list multiple domain types on one line, separated by commas. Valid values: • Directory Server (domain type for LDAP) • CA SiteMinder (domain type for a CA SiteMinder domain) • com.soa.securitydomain.pingfederate.provider (PingFederate domain) • com.soa.securitydomain.fb.connector (Facebook Connector domain) • com.soa.securitydomain.openidconnect.relyingparty (OpenID Connect Relying Party domain) • com.soa.securitydomain.oauth.provider (OAuth Provider domain; not an identity store domain, but supports OAuth/OpenID for other domain users) • com.soa.securitydomain.google.connector (Google Connector domain) • SAML Web Browser SSO (SAML Web SSO SP domain)

Examples/Notes/Additional Information

In the example below, this condition specifies that only users on a directory server (LDAP) domain can perform the action.

```
<condition type="authorizeByDomainType">
  <arg name="DomainType">Directory Server</arg>
</condition>
```

authorizeByEmail

Available in version: 7.2.4.2 and later

Tests to see if the email address of the workflow user matches one or more specified values; returns Boolean true or false.

This can be used as a security measure to limit platform actions to a specific set of authorized users. For example, actions can be limited to one specific email domain for security purposes.

Arguments

Name	Description/Values
email	One or more specific email address patterns. To include multiple values, you can either include multiple <email> arguments or list multiple values on one line, separated by commas.

Examples/Notes/Additional Information

In the example below, this condition specifies that only users with email addresses on the platform's domain, acmepaymentscorp.com, can perform the action.

```
<condition type="authorizeByEmail">
  <arg name="email">.*@acmepaymentscorp.com</arg>
</condition>
```

authorizeByGroupName

Available in version: 7.2.4.2 and later

Tests to see if the workflow user is a member of one or more specified platform groups; returns Boolean true or false.

Arguments

Name	Description/Values
Domain	Optional: one or more domains that authorization is restricted to. Only needed if the group is not a group on the developer portal; for example, a Policy Manager group. Defaults to developer portal groups. To include multiple values, you can either include multiple <domain> arguments or list multiple domains on one line, separated by commas. **Note**: the value for this parameter should be the domain name used in Policy Manager for the applicable identity system. For Policy Manager groups, use **Local Domain** as the value.

group	One or more platform groups that authorization is restricted to. By default, a group name is interpreted to mean a platform group. To include multiple values, you can either include multiple <group> arguments or list multiple groups on one line, separated by commas.

Examples/Notes/Additional Information

Example 1: One condition

In the example below, this condition checks that the logged-in user is a member of the **EngineeringGroup** group on the **ldap** domain. If the user does not meet the condition, the workflow action the condition is associated with is not allowed.

```
<condition type="authorizeByGroupName">
  <arg name="group">EngineeringGroup</arg>
  <arg name="domain">ldap</arg>
</condition>
```

Example 2: Multiple conditions

In the example below, this condition specifies that in order to perform the workflow action, users must meet any one of the following sets of conditions:

- Be a member of one of these two developer portal groups: CM_Group1 or CM_Group2
- Be a member of one of these two Policy Manager groups on the local domain: PM_Group1 or PM_Group2
- Be a member of one of these two groups on the LDAP domain: LDAP_Group1 or LDAP_Group2

```
<restrict-to>
  <conditions type="OR">
    <condition type="authorizeByGroupName">
      <arg name="group">CM_Group1, CM_Group2</arg>
    </condition>
    <condition type="authorizeByGroupName">
      <arg name="group">PM_Group1,PM_Group2</arg>
      <arg name="domain">Local Domain</arg>
    </condition>
          <condition type="authorizeByGroupName">
      <arg name="group">LDAP_Group1,LDAP_Group2</arg>
      <arg name="domain">ldap</arg>
    </condition>
  </conditions>
</restrict-to>
```

authTokenPropertyExists

Available in version: 7.2.4.3 and later

Checks to see if the specified property exists in the user's auth token cookie. Returns **true** if found.

Parameters

Name	Description/Values
PropertyName	The name of the auth token property being checked.
Message	A custom message designed by custom code or hardcoded into the workflow itself.

Examples/Notes/Additional Information

In the example below, two conditions are checked: first, is the session in login process (applicable when two-factor authentication is in effect and the user has already provided login credentials), and then, if the specific auth token property, **2FAData**, is there.

```
<conditions type="AND">
  <condition type="isSessionInLoginProcess"/>
  <condition type="authTokenPropertyExists">
    <arg name="PropertyName">2FAData</arg>
    <arg name="message">2FA not initiated</arg>
  </condition>
```

authTokenPropertyMatches

Available in version: 7.2.4.3 and later

Checks to see if the specified property in the user's auth token cookie matches the specified value. Returns **true** if so.

Parameters

Name	Description/Values
PropertyName	The name of the auth token property.
PropertyValue	The value of the auth token property being checked.

Examples/Notes/Additional Information

In the example below, the workflow checks if the auth token property is set to 2FAComplete indicating that the two-factor authentication process has completed successfully.

```
<condition negate="true" type="authTokenPropertyMatches">
  <arg name="PropertyName">2FAComplete</arg>
  <arg name="PropertyValue">Yes</arg>
</condition>
```

argumentExists

Available in version: 7.2.4.3 and later

Checks to see if the specified argument exists. Returns **true** if so.

Parameters

Name	Description/Values
ArgName	The name of the argument.
Message	A message to be generated if the argument exists.

Examples/Notes/Additional Information

In the example below, the custom workflow checks to see if a specific argument exists and, if it does, generates a specific message. The workflow then checks for a different argument and generates a message for that.

```
<action id="499" name="@ResolveLoginPendingTask">
  <restrict-to>
    <conditions type="AND">
      <condition type="isSessionInLoginProcess"/>
      <condition type="authTokenPropertyExists">
        <arg name="PropertyName">2FAData</arg>
        <arg name="message">2FA not initiated</arg>
      </condition>
      <condition negate="true" type="authTokenPropertyMatches">
          <arg name="PropertyName">2FAComplete</arg>
          <arg name="PropertyValue">Yes</arg>
      </condition>
      <condition type="argumentExists">
        <arg name="ArgName">Action</arg>
        <arg name="message">Action is required</arg>
      </condition>
      <condition type="argumentExists">
        <arg name="ArgName">task.id</arg>
        <arg name="message">Task id is required</arg>
      </condition>
    </conditions>
  </restrict-to>
```

argumentValueEquals

Available in version: 7.2.4.3 and later

Checks to see if a specific argument has a specific value. Further actions can then be taken based on the results.

Parameters

Name	Description/Values
ArgName	The name of the argument.
Value	The value of the argument.

Examples/Notes/Additional Information

In the example below, the **argumentValueEquals** condition is used to check if the value for **Action** is **regenerate**. If so (later part of the workflow not shown(, the 2FA code is regenerated.

```
<conditions type="AND">
   <condition type="argumentValueEquals">
   <arg name="ArgName">Action</arg>
   <arg name="Value">regenerate</arg>
  </condition>
```

argumentValueMatches

Available in version: 7.2.4.3 and later

Checks to see if a specific argument matches the specified regular expression. Further actions can then be taken based on the results.

Parameters

Name	Description/Values
ArgName	The name of the argument.
RegEx	The regular expression.

isSessionInLoginProcess

Available in version: 7.2.4.3 and later

Checks that the session is **not** in a LoginComplete state; if the session is still in the login process, indicating that login is not complete, returns **true**.

Parameters

None.

Examples/Notes/Additional Information

In the example below, the workflow action is restricted to the conditions specified. The first condition is to check that the session is in the login process. If this condition is not met, the step is not continued.

```
<action id="449" name="@ResolveLoginPendingTask">
  <restrict-to>
    <conditions type="AND">
      <condition type="isSessionInLoginProcess"/>
      <condition type="authTokenPropertyExists">
        <arg name="PropertyName">2FAData</arg>
        <arg name="message">2FA not initiated</arg>
      </condition>
      <condition negate="true" type="authTokenPropertyMatches">
          <arg name="PropertyName">2FAComplete</arg>
          <arg name="PropertyValue">Yes</arg>
```

```
    </condition>
```

cookieExists

Available in version: 7.2.4.3 and later

Checks whether the specified cookie exists. Returns **true** if so.

Parameters

Name	Description/Values
CookieName	The name of the cookie.

Examples/Notes/Additional Information

In the example below, the workflow action is restricted to the conditions specified. The first condition is to check that the cookie does not exist (negate-true on cookieExists). If the cookie **does** exist, the step is not continued.

```
<action id="401" name="@Login">
  <results>
    <result old-status="registered" status="registered" step="400">
      <conditions type="AND">
        <condition negate="true" type="cookieExists">
          <arg name="CookieName">D</arg>
        </condition>
        <condition negate="true" type="authTokenPropertyMatches">
          <arg name="PropertyName">2FAComplete</arg>
          <arg name="PropertyValue">Yes</arg>
        </condition>
        <condition negate="true" type="authTokenPropertyMatches">
          <arg name="PropertyName">2FASkipped</arg>
          <arg name="PropertyValue">Yes</arg>
        </condition>
        <condition negate="true" type="authTokenPropertyExists">
          <arg name="PropertyName">2FAData</arg>
        </condition>
      </conditions>
```

cookieValueEquals

Available in version: 7.2.4.3 and later

Checks to see if the user's cookie equals the specified value. Returns **true** if so.

Parameters

Name	Description/Values
CookieName	The name of the cookie.
CookieValue	The value of the cookie.

cookieValueMatches

Available in version: 7.2.4.3 and later

Checks to see if the user's cookie matches the specified regular expression. Returns **true** if so.

Parameters

Name	Description/Values
CookieName	The name of the cookie.
Expression	The regular expression

actionCommentMatchesRegEx

Available in version: 7.2.4.3 and later

Checks to see that the comment that's entered when a workflow action is performed matches the specified regular expression. Returns **true** if so. If it doesn't match, this condition returns false, and the user is given a message that can prompt the user to enter a valid comment.

When checking for valid actions, when the comment doesn't exist, this condition returns **true**. However, when performing the action, when the comment exists and doesn't follow the proper format, the condition returns **false** and returns the specified message.

Parameters

Name	Description/Values
RegEx	The regular expression that the comment is tested against for a match.
FailedMatchMessage	The message that will be returned if the comment that's entered when a workflow action is performed doesn't match the specified regular expression.

Examples/Notes/Additional Information

In the example below, the **actionCommentMatchesRegEx** condition is used to ensure that when performing a workflow action, the user enters a specific comment that includes the Support Ticket information (**RegEx** argument). If no comment is entered, the message specified in the **FailedMatchMessage** argument is displayed to the user.

```
<conditions type="AND">
  <condition type="actionCommentExists">
    <arg name="CommentMissingMessage">Comment needed with Support Ticket ID</arg>
  </condition>
  <condition type="actionCommentMatchesRegEx">
    <arg name="RegEx">.*Support Ticket*</arg>
    <arg name="FailedMatchMessage">Comment does not include Support Ticket ID</arg>
  </condition>
</conditions>
```

actionCommentExists

Available in version: 7.2.4.3 and later

Checks to see that a comment was entered when a workflow action is performed. Returns **true** if so.

Parameters

Name	Description/Values
CommentMissingMessage	The message that will be returned if a comment is **not** entered when a workflow action is performed.

Examples/Notes/Additional Information

In the example below, the **actionCommentExists** condition is used to ensure that when performing a workflow action, the user enters a comment. If no comment is entered, the message specified in the **CommentMissingMessage** argument is displayed to the user.

```
<conditions type="AND">
  <condition type="actionCommentExists">
    <arg name="CommentMissingMessage">Comment needed with Support Ticket ID</arg>
  </condition>
  <condition type="actionCommentMatchesRegEx">
    <arg name="RegEx">.*KYC.*</arg>
    <arg name="FailedMatchMessage">Comment does not include Support Ticket ID</arg>
  </condition>
</conditions>
```

General Use: Variable Resolvers

The following variable resolvers are available for all workflows in the Akana API Platform:

${workflow.step.name}

Available in version: 7.2.4.3 and later

The name for a specific workflow step, as a string in the format =${workflow.step.name}. Allows the workflow to dynamically reference another workflow step, by step name, as shown in the example below.

```
<result old-status="${workflow.step.name}" status="${workflow.step.name}" step="-1">
  <conditions type="AND">
    <condition type="isLastLoginEmpty"/>
    <condition type="class">
      <arg name="class.name">com.opensymphony.workflow.util.StatusCondition</arg>
      <arg name="status">managed</arg>
    </condition>
  </conditions>
```

```
  <post-functions>
    <function type="markUserPermanent"/>
    <function type="addBoardItem">
      <arg name="boardItemTemplateId">com.soa.board.item.user.logged.in.first.time</arg>
      <arg name="visibility">Limited</arg>
      <arg name="type">Discussion</arg>
      <arg name="author">${site.admin.dn}</arg>
      <arg name="targetBoard.apiversion">${connected.apiversion.ids}</arg>
      <arg name="targetBoard.api">${connected.apis.id}</arg>
      <arg name="viewers">${connected.apis.id},${business.dn},${site.admin.dn}</arg>
    </function>
    <function type="sendNotification">
      <arg name="role">ApiAdmins,SiteAdmin,BusinessAdmin</arg>
      <arg name="notificationType">com.soa.notification.type.user.logged.in.first.time</arg>
    </function>
    <function type="markLoginComplete"/>
  </post-functions>
</result>
```

${workflow.step.id}

Available in version: 7.2.4.3 and later

The name for a specific workflow step, as a string in the format =${workflow.step.id}. Allows the workflow to dynamically reference another workflow step, by step ID.

Chapter 3 | 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

The following initial actions are valid for Akana API Platform workflows relating to apps/app versions:

- @Create

@Create

Available in version: 7.1 and later

Starts the workflow, for a new app.

For more information, see Workflow Initial Actions on page 16.

App Version Workflow: Reserved Actions

The following reserved actions are defined for app version workflows:

@Modify

Available in version: 7.1 and later

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.

Note: @Modify controls permission to modify the entire app version. A user with this permission is authorized to perform all edit activities for the app version, such as saving or removing the app certificate/CSR, regenerating the shared secret, adding an app version, or editing the app's public profile. If @Modify is *not* present in the workflow, the platform checks for additional workflow actions that give more granular permission to perform a specific action.

@Delete

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

reserved-connect-to-api

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

@ModifyCert

Available in version: 7.1 and later

If you want to give users with one or more roles permission to modify the app certificate, you can use this reserved action. You can then specify which roles can perform the action.

The platform only looks for this workflow action if the @Modify action, which gives broad permission to modify all aspects of the app version, is not present in the custom workflow.

@RemoveCert

Available in version: 7.1 and later

Invoked when the app's certificate is removed.

If you want to give users with one or more roles permission to remove the app certificate, you can use this reserved action. You can then specify which roles can perform the action.

The platform only looks for this workflow action if the @Modify action, which gives broad permission to modify all aspects of the app version, is not present in the custom workflow.

@EditApp

Available in version: 7.1 and later

If you want to give users with one or more roles permission to edit the app, you can use this reserved action. You can then specify which roles can perform the action.

The platform only looks for this workflow action if the @Modify action, which gives broad permission to modify all aspects of the app version, is not present in the custom workflow.

@RegenerateSecret

Available in version: 7.1 and later

If you want to give users with one or more roles permission to regenerate the app's Shared Secret, you can use this reserved action. You can then specify which roles can perform the action.

The platform only looks for this workflow action if the @Modify action, which gives broad permission to modify all aspects of the app version, is not present in the custom workflow.

@AddVersion

Available in version: 7.1 and later

If you want to give users with one or more roles permission to add an app version, you can use this reserved action. You can then specify which roles can perform the action.

The platform only looks for this workflow action if the @Modify action, which gives broad permission to modify all aspects of the app version, is not present in the custom workflow.

@EditPublicProfile

Available in version: 7.1 and later

If you want to give users with one or more roles permission to modify the app's public profile, you can use this reserved action. You can then specify which roles can perform the action.

The platform only looks for this workflow action if the @Modify action, which gives broad permission to modify all aspects of the app version, is not present in the custom workflow.

@EditOAuthDetails

Available in version: 7.1 and later

If you want to give users with one or more roles permission to modify the app's OAuth details, you can use this reserved action. You can then specify which roles can perform the action.

The platform only looks for this workflow action if the @Modify action, which gives broad permission to modify all aspects of the app version, is not present in the custom workflow.

reserved-allow-oauth-auz-details-update

Available in version: 8.0 and later

Each app has a page where the app developer can specify the app's settings and preferences for OAuth. This page includes an additional section, for advanced **Authorization** settings, that isn't normally available to app developers. Only Site Admins or Business Admins see the **Authorization** section at the bottom of the OAuth Profile page.

If you want to allow app developers to modify values in the **Authorization** section of the App OAuth Profile, you'll need to upload a custom workflow that includes the **reserved-allow-oauth-auz-details-update** reserved workflow action.

The workflow section below shows how this would look in a custom workflow.

```
<action id="116" name="reserved-allow-oauth-auz-details-update">
  <restrict-to>
    <conditions type="AND">
```

```
    <condition type="authorizeByAtmosphereRole">
      <arg name="role">&AtmoRoleAdmin;</arg>
    </condition>
   </conditions>
  </restrict-to>
  <results>
   <unconditional-result old-status="Setup" status="Setup" step="100" owner="${caller}" />
  </results>
</action>
```

In the default developer portal, there is a template, **workflow-workflow_definition_appversion-workflow-template1.xml**, that you can use to implement this functionality.

App Version Workflow: Functions

The following functions are available for the app version workflow:

cloneAllAPIContracts

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Note: The email isn't sent instantly; it is queued to be sent. It goes to the notifications queue, and the job runs every 60 seconds. There might be a short delay before the user receives the email.

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

Available in version: 7.1 and later

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 46).

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

Available in version: 7.1 and later

Adds a Board item to one or more boards for the app.

This function can load a message template and dynamically fill in some values into the template with the help of a parameter resolver. When the workflow action is invoked, a comment is added by the user. The parameter resolver can be used to include this comment in the board item title/description, email message subject/body, and/or a dashboard notification. To use this feature, reference the parameter resolver {**action.comment}** in the notification template. When the action is performed, this parameter resolver will be replaced by the comment entered by the user performing the workflow action.

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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}

Available in version: 7.1 and later

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

${app.team.group.dn}

Available in version: 7.1 and later

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

${app.version.dn}

Available in version: 7.1 and later

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

${app.version.name}

Available in version: 7.1 and later

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

${business.dn}

Available in version: 7.1 and later

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

${connected.apis.admin.groups}

Available in version: 7.1 and later

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}

Available in version: 7.1 and later

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

${connected.apis.id}

Available in version: 7.1 and later

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

Chapter 4 | 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

The following initial actions are valid for Akana API Platform workflows relating to APIs/API versions:

- @Create

@Create

Available in version: 7.1 and later

Starts the workflow, for a new API version.

For more information, see Workflow Initial Actions on page 16.

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 52
- exportAPIAllVersions on page 53

exportAPIVersion

Available in version: 7.1 and later

Used to control workflow actions associated with export of 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

Available in version: 7.1 and later

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}

Available in version: 7.1 and later

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

Chapter 5 | 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 Akana API Platform workflows relating to API contracts are:

@Create

Available in version: 7.1 and later

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.

For more information, see Workflow Initial Actions on page 16.

@Revise

Available in version: 7.1 and later

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

Available in version: 7.1 and later

Used to start the workflow for an imported contract.

@AutoConnectActivate

Available in version: 7.1 and later

Used to start the workflow for a new contract that's created as a result of an AutoConnect setting.

API Contract Workflow: Reserved Actions

The following reserved actions are defined for API contract workflows:

@app_switch_to_production

Available in version: 7.1 and later

Used to control workflow actions that occur when an app is switched to production.

@AppDeleted

Available in version: 7.1 and later

Used to control workflow actions that occur when an app is deleted.

@ApiDeleted

Available in version: 7.1 and later

Used to control workflow actions that occur when an API is deleted.

@modify

Available in version: 7.1 and later

Used to control workflow actions that occur when an app is modified.

@Revise

Available in version: 7.1 and later

Used to control workflow actions that occur when an app is revised.

reserved-connect-from-app.Sandbox

Available in version: 7.1 and later

Used to control workflow actions that occur when an app connects to an API in the Sandbox environment.

reserved-connect-from-app.Production

Available in version: 7.1 and later

Used to control workflow actions that occur when an app connects to an API in the Production environment.

API Contract Workflow: Functions

The following functions are available for the API contract workflow:

updateAPIContractStatus

Available in version: 7.1 and later

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

Available in version: 7.1 and later

Triggers the specified notification based on an event relating to an API contract.

Note: The email isn't sent instantly; it is queued to be sent. It goes to the notifications queue, and the job runs every 60 seconds. There might be a short delay before the user receives the email.

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Parameters

None.

isAtmosphereSandboxApiContract

Available in version: 7.1 and later

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

Parameters

None.

isAtmosphereProductionApiContract

Available in version: 7.1 and later

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

Parameters

None.

isAtmosphereSandboxAutoApprove

Available in version: 7.1 and later

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

Parameters

None.

isAtmosphereProductionAutoApprove

Available in version: 7.1 and later

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

Parameters

None.

isRemoteFedMemberApp

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Parameters

None.

isAPIContractScopeNotEmpty

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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}

Available in version: 7.1 and later

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}

Available in version: 7.1 and later

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

${contract.app.dn}

Available in version: 7.1 and later

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

${contract.app.version.dn}

Available in version: 7.1 and later

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

${contract.dn}

Available in version: 7.1 and later

The APIContractID of the contract.

${contract.state}

Available in version: 7.1 and later

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}

Available in version: 7.1 and later

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

Chapter 6 | 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

The following initial actions are defined for ticket workflows:

- @Create

@Create

Available in version: 7.1 and later

Starts the workflow, for a new ticket.

For more information, see Workflow Initial Actions on page 16.

Ticket Workflow: Reserved Actions

The following reserved actions are defined for ticket workflows:

- @modify

@modify

Available in version: 7.1 and later

Used to control workflow actions that occur when a ticket is modified.

Ticket Workflow: Functions

There is one function available for the ticket workflow:

- updateTicketStatus on page 67

updateTicketStatus

Available in version: 7.1 and later

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.

Chapter 7 | 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 Akana API Platform workflows relating to group membership are:

@Invite

Available in version: 7.1 and later

Starts the workflow for a new group member. Used to control workflow actions that occur when a member is invited to a group.

This is the default initial action, used in most cases.

@Import

Available in version: 7.1 and later

Starts the workflow when group membership information is imported. Takes the group membership direct to the Approved state.

Group Membership Workflow: Reserved Actions

The following reserved actions are defined for group membership workflows:

- group.membership.action.make.leader on page 71
- group.membership.action.make.member on page 71
- group.membership.action.group.deleted on page 71

@RecreateInPendingState

Available in version: 7.1 and later

Used to control workflow actions that occur when a group membership action is recreated in pending state.

@RecreateInAcceptedState

Available in version: 7.1 and later

Used to control workflow actions that occur when a group membership action is recreated in accepted state.

@RecreateInDeclinedState

Available in version: 7.1 and later

Used to control workflow actions that occur when a group membership action is recreated in declined state.

Group.membership.action.accept

Available in version: 7.1 and later

Used to control workflow actions that occur when a group membership invitation is accepted.

group.membership.action.decline

Available in version: 7.1 and later

Used to control workflow actions that occur when a group membership invitation is declined.

group.membership.action.resend

Available in version: 7.1 and later

Used to control workflow actions that occur when a group membership invitation is resent.

group.membership.action.remove

Available in version: 7.1 and later

Used to control workflow actions that occur when a group member is removed from the group.

group.membership.action.make.admin

Available in version: 7.1 and later

Used to control workflow actions that occur when a group member who was a member or leader is made a group admin.

group.membership.action.make.leader

Available in version: 7.1 and later

Used to control workflow actions that occur when a group member who was a member or admin is made a group leader.

group.membership.action.make.member

Available in version: 7.1 and later

Used to control workflow actions that occur when a group member who was a member or leader is made a group member.

group.membership.action.group.deleted

Available in version: 7.1 and later

Used to control workflow actions that occur when a group is deleted.

Group Membership Workflow: Functions

The following functions are available for the group membership workflow:

setGroupMembershipRequestState

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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 76
- isCallerSiteAdmin on page 77

isSelfMembership

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Arguments

None.

isCallerGroupLeader

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Arguments

None.

isMemberMembership

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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.

authorizeInviteeByDomain

Available in version: 7.1 and later

Checks the domain of the invitee to make sure the domain is one of the domains specified as valid. If so, returns **true**. If the condition is not met, the invitation is not allowed.

This can be used as a security measure to help prevent user error in potentially inviting someone who should not be invited. For example, invitations can be limited to one specific domain for security purposes.

Arguments

Name	Description/Values
domain	One or more domains that authorization is restricted to. To include multiple values, you can either include multiple <domain> arguments or list multiple domains on one line, separated by commas. **Note**: the value for this parameter should be the domain name used in Policy Manager for the applicable identity system. For Policy Manager, use **Local Domain** as the value.

Examples/Notes/Additional Information

In the example below, this condition specifies that only users on the platform's domain, acmepaymentscorp, can be invited.

```
<condition type="authorizeInviteeByDomain">
  <arg name="domain">acmepaymentscorp</arg>
</condition>
```

authorizeInviteeByDomainType

Available in version: 7.1 and later

Checks the domain type of the invitee to make sure it is one of the domain types specified as valid. If so, returns **true**. If the condition is not met, the invitation is not allowed.

This can be used as a security measure to help prevent user error in potentially inviting someone who should not be invited. For example, invitations can be limited to one type of domain, such as LDAP, for security purposes.

Arguments

Name	Description/Values
DomainType	One or more domain types (Identity System type in Policy Manager, Domain Type in the developer portal), that the action is restricted to. To include multiple values, you can either include multiple <DomainType> arguments or list multiple domain types on one line, separated by commas. Valid values: • Directory Server (domain type for LDAP) • CA SiteMinder (domain type for a CA SiteMinder domain) • com.soa.securitydomain.pingfederate.provider (PingFederate domain) • com.soa.securitydomain.fb.connector (Facebook Connector domain) • com.soa.securitydomain.openidconnect.relyingparty (OpenID Connect Relying Party domain) • com.soa.securitydomain.oauth.provider (OAuth Provider domain; not an identity store domain, but supports OAuth/OpenID for other domain users) • com.soa.securitydomain.google.connector (Google Connector domain) • SAML Web Browser SSO (SAML Web SSO SP domain)

Examples/Notes/Additional Information

The example below tests that the domain type is Directory Server. This limits invitees to LDAP users.

```
<condition type="authorizeInviteeByDomainType">
  <arg name="DomainType">Directory Server</arg>
</condition>
```

authorizeInviteeByEmail

Available in version: 7.1 and later

Checks the email address of the invitee to make sure it matches the email address patterns specified in the argument. If so, returns **true**. If the condition is not met, the invitation is not allowed.

This can be used as a security measure to help prevent user error in potentially inviting someone who should not be invited. For example, invitations can be limited to one specific email domain for security purposes.

Arguments

Name	Description/Values
email	One or more specific email address patterns valid for invitations. To include multiple values, you can either include multiple <email> arguments or list multiple email values on one line, separated by commas.

Examples/Notes/Additional Information

The example below shows the general structure of this argument.

```
<condition type="authorizeInviteeByEmail">
  <arg name="email">Email Address,Pattern</arg>
</condition>
```

The two examples below show different approaches to restricting email address to two domains.

Example 1: Both domains on the same line, with comma separators, using the AND condition:

```
<restrict-to>
  <conditions type="AND">
    <condition type="authorizeByEmail">
      <arg name="email">.*@soa.com,.*@akana.com</arg>
    </condition>
  </conditions>
</restrict-to>
```

Example 2: Each domain on a separate line, using the OR condition:

```
<restrict-to>
  <conditions type="OR">
    <condition type="authorizeByEmail">
      <arg name="email">.*@soa.com</arg>
    </condition>
    <condition type="authorizeByEmail">
      <arg name="email">.*@akana.com</arg>
    </condition>
  </conditions>
</restrict-to>
```

authorizeInviteeByGroupName

Available in version: 7.1 and later

Checks to make sure the invitee is a member of one of the specified groups. If so, returns **true**. If the condition is not met, the invitation is not allowed.

This can be used as a security measure to help prevent user error in potentially inviting someone who should not be invited. For example, invitations can be limited to one specific group for security purposes.

Note: If no domain argument is passed, the condition is applied to a platform group.

Arguments

Name	Description/Values
domain	Optional: one or more domains that authorization is restricted to. Only needed if the group is not a group on the developer portal; for example, a Policy Manager group. Defaults to developer portal groups. To include multiple values, you can either include multiple <domain> arguments or list multiple domains on one line, separated by commas. **Note**: the value for this parameter should be the domain name used in Policy Manager for the applicable identity system. For Policy Manager groups, use **Local Domain** as the value.
group	One or more groups that authorization is restricted to. By default, if the <domain> argument is not present, a group name is interpreted to mean a platform group. To include multiple values, you can either include multiple <group> arguments or list multiple groups on one line, separated by commas.

Examples/Notes/Additional Information

In the example below, this condition specifies that in order to be invited, users must meet any one of the following sets of conditions:

- Be a member of one of these two developer portal groups: CM_Group1 or CM_Group2
- Be a member of one of these two Policy Manager groups on the local domain: PM_Group1 or PM_Group2
- Be a member of one of these two groups on the LDAP domain: LDAP_Group1 or LDAP_Group2

```
<restrict-to>
  <conditions type="OR">
    <condition type="authorizeInviteeByGroupName">
      <arg name="group">CM_Group1, CM_Group2</arg>
    </condition>
    <condition type="authorizeInviteeByGroupName">
      <arg name="domain">Local Domain</arg>
      <arg name="group">PM_Group1,PM_Group2</arg>
    </condition>
          <condition type="authorizeInviteeByGroupName">
      <arg name="domain">ldap</arg>
      <arg name="group">LDAP_Group1,LDAP_Group2</arg>
    </condition>
```

```
  </conditions>
</restrict-to>
```

Group Membership Workflow: Variable Resolvers

The following variables are available for the group membership workflow:

${group.dn}

Available in version: 7.1 and later

The unique GroupID for the group.

${group.type}

Available in version: 7.1 and later

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}

Available in version: 7.1 and later

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}

Available in version: 7.1 and later

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

${member.dn}

Available in version: 7.1 and later

The unique User ID of the group member.

${groupmembership.oldrole}

Available in version: 7.1 and later

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}

Available in version: 7.1 and later

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}

Available in version: 7.1 and later

The group membership role.

${groupmembership.state}

Available in version: 7.1 and later

The group membership state.

Chapter 8 | 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

Available in version: 7.1 and later

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

@AddApp

Available in version: 7.1 and later

Determines whether the current user has permission to add an app.

By default, all registered users have permission to add an app. However, this action can be combined with a condition, such as **authorizeByAtmosphereRole**, **authorizeByGroup**, **authorizeByDomain**, or **authorizeByEmail** to determine whether the current user has permission to add an app.

For example, you might want to configure a custom workflow to support one or more of the following scenarios:

- Only tenant business administrators can create apps.
- Users that are self-registered can create apps, but not users added by the Site Admin.
- All users can browse APIs, but only certain approved users can create apps. This implies users that are in a particular workflow state can create apps.

@AddGroup

Available in version: 7.1 and later

Determines whether the current user has permission to add a group.

By default, all registered users have permission to add a group. However, this action can be combined with a condition, such as **authorizeByAtmosphereRole**, **authorizeByGroup**, **authorizeByDomain**, or **authorizeByEmail** to determine whether the current user has permission to add a group.

For example, you might want to configure a custom workflow to support one or more of the following scenarios:

- Only tenant business administrators can create groups.
- Users that are self-registered can create groups, but not users added by the Site Admin.
- Only certain approved users can create groups.

@AgreementsAccepted

Available in version: 7.1 and later

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

@ChallengeQuestionsAnswered

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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 (Managed Users), they cannot modify the profile unless they are site administrators.
- If the platform setting allowing users to modify their own profiles is turned on, even a Site Admin cannot modify his/her own profile; it must be modified by another Site Admin.

Examples/Notes/Additional Information

In the example below, the workflow for the @ModifyProfile workflow action checks to make sure that either the setting to allow users who are not Managed Users to modify their own profiles is turned on (condition **userSettingsAllowModifyProfile**), or else the current user is a Site Admin. If one of these conditions is met, the action can go forward.

Note that this condition only applies to users who signed up for the platform themselves (whether by invitation, by setting up a platform account, or by setting up an account with a third-party identity provider such as Google). Managed Users, added by the Site Admin, cannot modify their own profiles, even if the platform setting is turned on.

If the setting to allow users to modify their own profiles is disabled, a user who is a Site Admin cannot modify his/her own profile. In this case, only another Site Admin can modify this Site Admin's profile.

```
<action id="456" name="@ModifyProfile">
  <restrict-to>
    <conditions type="OR">
      <conditions type="AND">
        <condition type="authorizeSelf"/>
        <condition type="userSettingsAllowModifyProfile"/>
      </conditions>
      <conditions type="AND">
        <condition type="authorizeByAtmosphereRole">
          <arg name="role">&RoleSiteAdmin;</arg>
        </condition>
        <condition negate="true" type="authorizeSelf"/>
      </conditions>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="registered" status="registered" step="450"/>
  </results>
</action>
```

@PasswordChanged

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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.akana.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.

@UserDisabled

Available in version: 7.2.4.2 and later

Can be used to initiate one or more actions when the current user's account is disabled.

For example, you might want to configure a custom workflow so that when the current user's account is disabled, a Board item is created and a notification is sent to the user and, perhaps, a different notification to Site Admins and/or Business Admins.

Applicable board items and notifications:

- com.soa.board.item.user.disabled
- User notification: com.soa.notification.type.user.disabled
- Admin notification: com.soa.notification.type.user.account.status.change

@UserEnabled

Available in version: 7.2.4.2 and later

Can be used to initiate one or more actions when the current user's account is enabled.

For example, you might want to configure a custom workflow so that when the current user's account is enabled, a notification is sent to the user and, perhaps, a different notification to Site Admins and/or Business Admins.

Applicable board items and notifications:

- com.soa.board.item.user.enabled
- User notification: com.soa.notification.type.user.disabled
- Admin notification: com.soa.notification.type.user.account.status.change

@UserLocked

Available in version: 7.2.4.2 and later

Can be used to initiate one or more actions when a user's account is locked.

For example, you might want to configure a custom workflow so that when the current user's account is locked, a notification is sent to the user and, perhaps, a different notification to Site Admins and/or Business Admins.

Applicable board items and notifications:

- com.soa.board.item.user.locked
- User notification: com.soa.notification.type.user.locked
- Admin notification: com.soa.notification.type.user.account.status.change

@UserUnlocked

Available in version: 7.2.4.2 and later

Can be used to initiate one or more actions when a user's account is unlocked.

For example, you might want to configure a custom workflow so that when the current user's account is unlocked, a notification is sent to the user and, perhaps, a different notification to Site Admins and/or Business Admins.

Applicable board items and notifications:

- com.soa.board.item.user.unlocked
- User notification: com.soa.notification.type.user.unlocked
- Admin notification: com.soa.notification.type.user.account.status.change

@ResolveLoginPendingTask

Available in version: 7.2.4.3 and later

Used to determine whether there are pending tasks the user must complete before login is complete and, if so, to guide the user through those tasks; for example, validating a two-factor verification code or requesting a new verification code.

This reserved action is valid only when the user is in the login process; for example, if two-factor authentication is turned on and the user has authenticated with credentials but has not yet provided the verification code.

The default workflow restricts this action to be valid only when the **isSessionInLoginProcess** condition is true.

Additional functions can be used to specify actions that will be taken to resolve the pending task, such as:

- If 2FA is turned on but has not been initiated, generate the verification code, send the cookie, and generate a notification.
- If change password is required, forcing the user to change the password.
- If change password is not required, check if agreements are accepted, and if there are pending legal agreements, guiding the user to accept the legal agreement.
- If security questions are required to complete login, guiding the user to provide answers to the security questions.

Examples/Notes/Additional Information

In the example below, this reserved action is invoked if the pre-conditions are met. Pre-conditions are that the login is in process, the auth token property exists (indicating that 2FA is required) and the auth token property does **not** match **2FAComplete** (indicating that the two-factor authentication process is not complete).

```
<action id="499" name="@ResolveLoginPendingTask">
  <restrict-to>
    <conditions type="AND">
      <condition type="isSessionInLoginProcess"/>
      <condition type="authTokenPropertyExists">
        <arg name="PropertyName">2FAData</arg>
        <arg name="message">2FA not initiated</arg>
      </condition>
      <condition negate="true" type="authTokenPropertyMatches">
          <arg name="PropertyName">2FAComplete</arg>
          <arg name="PropertyValue">Yes</arg>
      </condition>
      <condition type="argumentExists">
        <arg name="ArgName">Action</arg>
        <arg name="message">Action is required</arg>
      </condition>
      <condition type="argumentExists">
        <arg name="ArgName">task.id</arg>
        <arg name="message">Task id is required</arg>
      </condition>
    </conditions>
  </restrict-to>
```

User Workflow: Functions

The following functions are available for the user workflow:

- MarkUserPermanent on page 98
- markUserPermanentIfFirstLogin on page 98
- addBoardItem on page 99

MarkUserPermanent

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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>
```

addBoardItem

Available in version: 7.1 and later

Adds a Board item to one or more boards for the user.

This function can load a message template and dynamically fill in some values into the template with the help of a parameter resolver. When the workflow action is invoked, a comment is added by the user. The parameter resolver can be used to include this comment in the board item title/description, email message subject/body, and/or a dashboard notification. To use this feature, reference the parameter resolver {**action.comment}** in the notification template. When the action is performed, this parameter resolver will be replaced by the comment entered by the user performing the workflow action.

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.user.logged.in.first.time
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. There are two ways to add the targetBoard information: • **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 post-function, to announce that a user added by the Site Admin has logged in for the first time. In this example, **targetBoard** is a set of separate arguments, and **viewers** is a plural parameter.

```
<post-functions>
  <function type="markUserPermanent"/>
  <function type="addBoardItem">
    <arg name="boardItemTemplateId">com.soa.board.item.user.logged.in.first.time</arg>
    <arg name="visibility">Limited</arg>
    <arg name="type">Discussion</arg>
    <arg name="author">${site.admin.dn}</arg>
    <arg name="targetBoard.apiversion">${connected.apiversion.ids}</arg>
    <arg name="targetBoard.api">${connected.apis.id}</arg>
    <arg name="viewers">${connected.apis.id},${business.dn},${site.admin.dn}</arg>
  </function>
  <function type="sendNotification">
    <arg name="role">ApiAdmins,SiteAdmin,BusinessAdmin</arg>
    <arg name="notificationType">com.soa.notification.type.user.logged.in.first.time</arg>
  </function>
```

sendNotification

Available in version: 7.1 and later

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

Note: The email isn't sent instantly; it is queued to be sent. It goes to the notifications queue, and the job runs every 60 seconds. There might be a short delay before the user receives the email.

This function can load a message template and dynamically fill in some values into the template with the help of a parameter resolver. When the workflow action is invoked, a comment is added by the user. The parameter resolver can be used to include this comment in the board item title/description, email message subject/body, and/or a dashboard notification. To use this feature, reference the parameter resolver {**action.comment}** in the notification template. When the action is performed, this parameter resolver will be replaced by the comment entered by the user performing the workflow action.

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 • com.soa.notification.type.user.2fa.verification.code
role	The role to which the notifications will be sent. Valid values: • ApiAdmins • SiteAdmin • BusinessAdmin • 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="isLocalDomainUser"/>
     </conditions>
  <post-functions>
       <function type="sendNotification">
        <arg name="notificationType">com.soa.notification.type.user.admin.added</arg>
        <arg name="role">ApiAdmins,SiteAdmin,BusinessAdmin</arg>
       </function>
      </post-functions>
  </result>
  <unconditional-result old-status="none" status="unknown" step="-1"/>
  </results>
 </action>
</actions>
</step>
```

setProperty

Available in version: 7.1 and later

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 91).

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>
```

setTwofaDeliveryOptions

Available in version: 7.2.4.3 and later

When two-factor authentication is in use, indicates the delivery options supported for the 2FA verification code.

Note: In the platform's out-of-the-box 2FA use workflow, if more than one delivery option is defined as valid in the workflow, the platform does not generate the verification code immediately. Instead, it returns the valid delivery options. The user can then select an option; based on that, the verification code is generated.

Parameters

Name	Description/Values
VoiceSupported	Indicates whether a voice option is supported for the 2FA verification code. Boolean **true**/**false**.
TextSupported	Indicates whether text messaging is supported for the 2FA verification code. Boolean **true**/**false**.
EmailSupported	Indicates whether email is supported for the 2FA verification code. Boolean **true**/**false**.

Examples/Notes/Additional Information

In the example below, **setTwofaDeliveryOptions** is invoked as the first pre-function to specify the supported delivery options. Depending on the delivery options supported, subsequent workflow actions guide the process.

```
<pre-functions>
 <function type="setTwofaDeliveryOptions">
  <arg name="VoiceSupported">true</arg>
```

```
   <arg name="TextSupported">true</arg>
   <arg name="EmailSupported">true</arg>
  </function>
  <function type="handle2FATask">
   <arg name="PropertyName">2FAData</arg>
   <arg name="2FAData">${authtoken.property.2FAData}</arg>
   <arg name="2FACode">${arg.2fa.code}</arg>
   <arg name="2FASalt">${authtoken.property.2FASalt}</arg>
  </function>
</pre-functions>
```

setTwofaDeliveryTarget

Available in version: 7.2.4.3 and later

When two-factor authentication is in use, indicates the delivery option that the user has chosen for the 2FA verification code.

Parameters

Name	Description/Values
DeliveryOptions	${arg.2fa.task.data}. This resolves to the value of **2fa.task.data**. See ${arg.xxxx} on page 114.
DeliveryMechanism	Indicates the way the verification code will be delivered. Valid values, if defined as **true** in arguments for the **setTwofaDeliveryOptions** function: • Email • Voice • Text

Examples/Notes/Additional Information

In the example below, **setTwofaDeliveryTarget** is set to **text**.

```
<pre-functions>
  <function type="setTwoFADeliveryTarget">
   <arg name="DeliveryOptions">${arg.2fa.task.data}</arg>
   <arg name="DeliveryMechanism">Text</arg>
   <arg name="DeliveryTargetAddress">${arg.twofa.delivery.target.address}</arg>
   <!--<arg name="Index">0</arg>--><!-- Either or -->
  </function>
  <function type="generate2FACode"/>
  <function type="setAuthTokenProperty">
   <arg name="PropertyName">2FAData</arg>
   <arg name="PropertyValue">${arg.2fa.authtoken.property}</arg>
  </function>
```

send2FACodeToEmail

Available in version: 7.2.4.3 and later

When two-factor authentication is in use, indicates that the delivery mechanism by which the 2FA verification code will be sent to the user is email, and conveys the applicable information.

The platform's out-of-the-box user workflow includes support for sending the 2FA code to an email address.

Note: The email isn't sent instantly; it is queued to be sent. It goes to the notifications queue, and the job runs every 60 seconds. There might be a short delay before the user receives the email.

Parameters

Name	Description/Values
DeliveryTargetAddress	${arg.twofa.delivery.target.address}: the specific email address the verification code will be sent to. If this parameter is used in a custom function to send the code to a phone, it could be the phone number, including country code.
DeliveryMechanism	${arg.DeliveryMechanism}: indicates the way the verification code will be sent to the user. The only valid value for this function is Email. If this parameter is used in a custom function to send the code to a phone, the value could be Voice or Text.
2FAData	A value used internally. Not currently needed.
2FAVerificationCode	${arg.verificationCode}: the specific verification code sent to the user.
2FAVerificationCodeValidMinutes	${arg.2fa.verification.code.valid.minutes}: the validity period for the 2FA code, as set up in the platform settings.

Examples/Notes/Additional Information

The example below is a template for this function.

```
<function type="send2FACodeToEmail">
  <arg name="DeliveryTargetAddress">${arg.twofa.delivery.target.address}</arg>
  <arg name="DeliveryMechanism">${arg.DeliveryMechanism}</arg>
  <arg name="2FAData">${arg.2fa.task.data}</arg>
  <arg name="2FAVerificationCode">${arg.verificationCode}</arg>
  <arg name="2FAVerificationCodeValidMinutes">${arg.2fa.verification.code.valid.minutes}</arg>
</function>
```

generate2FACode

Available in version: 7.2.4.3 and later

Generates the code to be used in the two-factor authentication process, and:

- Sets the generated 2FA code details into the argument properties.

- Sets an argument property with an opaque token that must be saved so that it can be used later for validating the 2FA code.

setPendingTask

Available in version: 7.2.4.3 and later

Specifies a pending task, and allows some associated data relating to the pending tasks to be included. For example, this can be used to specify that two-factor authentication is required and is a pending task that must be completed before user login is complete; in this scenario, this could then give the 2FA data.

unmarshall2FACode

Available in version: 7.2.4.3 and later

Extracts the 2FA object from the auth token data, and converts the opaque 2FA token into a structure that other 2FA functions can use.

This function allows the 2FA data to be read if needed; for example, by the `GET /api/login/status` operation that returns information about the user's current login status.

handle2FATask

Available in version: 7.2.4.3 and later

Relating to two-factor verification codes generated by the Generate2FACode function, **handle2FATask** takes care of any of the following:

- Generate the verification code
- Validate the verification code

Examples/Notes/Additional Information

In the example below, the **handle2FATask** function sets a value for the opaque code, 2FACode, and sets the structure into the auth token property, **2FAData**.

```
<pre-functions>
  <function type="handle2FATask">
    <arg name="PropertyName">2FAData</arg>
    <arg name="2FAData">${authtoken.property.2FAData}</arg>
    <arg name="2FACode">${arg.2fa.code}</arg>
  </function>
</pre-functions>
```

validate2FACode

Available in version: 7.2.4.3 and later

Validates the two-factor verification code. The session must be in login process for this function to be valid.

terminateSession

Available in version: 7.2.4.3 and later

Allows the workflow to force logout and end the current user's session. This function removes the browser cookie, which means the session is terminated.

There might be a short delay before it's apparent to the user that the UI session has ended. However, if this is used as part of the ResolveLoginPendingTask action, the UI will recognize it immediately and give the user a notification message.

Examples/Notes/Additional Information

In the example below, the workflow checks whether the code is expired or max attempts is exceeded. If either of these conditions is met, 2FA required is set as a pending task and the user's cookie is removed. This means that the user must start the login process from the beginning.

```
<conditions type="OR">
  <condition type="is2FACodeExpired"/>
  <condition type="is2FAMaxAttemptsExceeded"/>
</conditions>
<pre-functions>
  <function type="setPendingTask">
    <arg name="PendingTask">2fa.required</arg>
    <arg name="TaskData">${arg.2fa.task.data}</arg>
  </function>
  <function type="removeAuthTokenProperty">
    <arg name="PropertyName">2FAData</arg>
  </function>
  <function type="terminateSession"/>
</pre-functions>
```

User Workflow: Conditions

The following conditions apply to the user workflow:

isLocalDomainUser

Available in version: 7.1 and later

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

IsRegisteredUser

Available in version: 7.1 and later

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

IsLocalRegisteredUser

Available in version: 7.1 and later

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

IsLastLoginEmpty

Available in version: 7.1 and later

Tests to see whether this is the first login.

IsChangePasswordRequired

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

IsForceChallengeQuestionsAnsweredOnLoginSetup

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

IsSelfSignupAllowed

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Examples/Notes/Additional Information

In the example below, the workflow for the @ModifyProfile workflow action checks to make sure that either the setting to allow users who are not Managed Users to modify their own profiles is turned on (condition **userSettingsAllowModifyProfile**), or else the current user is a Site Admin. If one of these conditions is met, the action can go forward.

Note that this condition only applies to users who signed up for the platform themselves (whether by invitation, by setting up a platform account, or by setting up an account with a third-party identity provider such as Google). Managed Users, added by the Site Admin, cannot modify their own profiles, even if the platform setting is turned on.

If the setting to allow users to modify their own profiles is disabled, a user who is a Site Admin cannot modify his/her own profile. In this case, only another Site Admin can modify this Site Admin's profile.

For more information, refer to @ModifyProfile on page 93.

```
<action id="456" name="@ModifyProfile">
  <restrict-to>
    <conditions type="OR">
      <conditions type="AND">
        <condition type="authorizeSelf"/>
        <condition type="userSettingsAllowModifyProfile"/>
      </conditions>
      <conditions type="AND">
        <condition type="authorizeByAtmosphereRole">
          <arg name="role">&RoleSiteAdmin;</arg>
        </condition>
        <condition negate="true" type="authorizeSelf"/>
      </conditions>
    </conditions>
  </restrict-to>
  <results>
    <unconditional-result old-status="registered" status="registered" step="450"/>
  </results>
</action>
```

IsInviteUnRegisteredUserAllowed

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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**.

Available in version: 7.1 and later

Is2FAEnabled

Available in version: 7.2.4.3 and later

Checks the platform settings to see whether two-factor authentication is enabled for user login; returns Boolean true or false.

```
<condition type="is2FAEnabled"/>
```

is2FARequired

Available in version: 7.2.4.3 and later

Checks to see whether two-factor authentication is required for login of the current user, based on the recurrence mode specified in the platform's 2FA settings and the user's history of completing the 2FA process. Returns Boolean true or false.

```
<condition type="is2FARequired"/>
```

isNoDeliveryOption

Available in version: 7.2.4.3 and later

When used with negative=true, as shown below, checks to make sure that there is at least one 2FA delivery option specified. Returns Boolean true or false.

Arguments

Name	Description/Values
PropertyName	${arg.2fa.task.data}. This resolves to the value of **2fa.task.data**. See ${arg.xxxx} on page 114.

Examples/Notes/Additional Information

```
<condition negate="true" type="isNoDeliveryOption">
  <arg name="PropertyName">${arg.2fa.task.data}</arg>
</condition>
```

isDeliveryOptionsOnlyOne

Available in version: 7.2.4.3 and later

Tests to see if there is only one delivery option defined for the 2FA verification code. If there is only one delivery option, the user is not offered a choice. Instead, the code is generated and issued to the user by the specified delivery option and the user is directed to the UI page for entering the verification code.

Returns Boolean true or false.

Arguments

Name	Description/Values
PropertyName	${arg.2fa.task.data}. This resolves to the value of **2fa.task.data**. See ${arg.xxxx} on page 114.

Examples/Notes/Additional Information

```
<condition type="is2FAEnabled"/>
<condition type="is2FARequired"/>
<condition type="isDeliveryOptionsOnlyOne">
```

```
  <arg name="PropertyName">${arg.2fa.task.data}</arg>
</condition>
```

isDeliveryOptionsOnlyEmail

Available in version: 7.2.4.3 and later

Checks to see if the only valid delivery option for the 2FA verification code is email (this is the platform default).

Returns Boolean true or false.

Arguments

Name	Description/Values
PropertyName	${arg.2fa.task.data}. This resolves to the value of **2fa.task.data**. See ${arg.xxxx} on page 114.

Examples/Notes/Additional Information

```
<condition type="isDeliveryOptionsOnlyEmail"><!-- out of the box support -->
  <arg name="PropertyName">${arg.2fa.task.data}</arg>
</condition>
```

isDeliveryTypeEmail

Available in version: 7.2.4.3 and later

Checks to see if the only valid delivery option for the 2FA verification code is email (this is the platform default).

Arguments

Name	Description/Values
PropertyName	${arg.2fa.task.data}. This resolves to the value of **2fa.task.data**. See ${arg.xxxx} on page 114.

Examples/Notes/Additional Information

```
<condition type="isDeliveryOptionsOnlyEmail"><!-- out of the box support -->
  <arg name="PropertyName">${arg.2fa.task.data}</arg>
</condition>
```

isDeliveryTypeVoice

Available in version: 7.2.4.3 and later

Checks to see if the only valid delivery option for the 2FA verification code is email (this is the platform default).

Arguments

Name	Description/Values
PropertyName	${arg.2fa.task.data}. This resolves to the value of **2fa.task.data**. See ${arg.xxxx} on page 114.

Examples/Notes/Additional Information

```
<condition type="isDeliveryOptionsOnlyEmail"><!-- out of the box support -->
  <arg name="PropertyName">${arg.2fa.task.data}</arg>
</condition>
```

isDeliveryTypeText

Available in version: 7.2.4.3 and later

Checks to see if the only valid delivery option for the 2FA verification code is email (this is the platform default).

Arguments

Name	Description/Values
PropertyName	${arg.2fa.task.data}. This resolves to the value of **2fa.task.data**. See ${arg.xxxx} on page 114.

Examples/Notes/Additional Information

```
<condition type="isDeliveryOptionsOnlyEmail"><!-- out of the box support -->
  <arg name="PropertyName">${arg.2fa.task.data}</arg>
</condition>
```

is2FACodeValid

Available in version: 7.2.4.3 and later

Tests to see if the verification code that the user provided is valid. Returns **true** if the code is valid.

In the example below, **is2FACodeValid** is the second condition. Not shown: if both conditions evaluate to true, the workflow sets the auth token cookie and checks if there are any other pending login steps to complete the user's login.

```
<result old-status="registered" status="registered" step="450">
  <conditions type="AND">
```

```
  <condition type="argumentValueEquals">
    <arg name="ArgName">Action</arg>
    <arg name="Value">validate</arg>
  </condition>
  <condition type="is2FACodeValid"/>
 </conditions>
```

is2FATerminated

Available in version: 7.2.4.3 and later

Checks to make sure that the 2FA session has not been terminated. 2FA is terminated if conditions are violated such as if the number of attempts with an invalid code has been exceeded or if the code has expired. If the session has been terminated, this condition returns **true**.

Note: The valid number of attempts for a specific instance of the platform, and the timeout period for the cookie, are determined by the Site Admin in the 2FA settings.

Examples/Notes/Additional Information

In the example below, if **is2FATerminated** resolves to **Yes**, **2fa.required** is set as a pending task, the current auth token property is removed, and the user's session is terminated. The user must start the login process from the beginning.

```
<result old-status="registered" status="registered" step="400">
 <conditions type="AND">
  <condition type="is2FATerminated"/>
 </conditions>
 <pre-functions>
  <function type="setPendingTask">
   <arg name="PendingTask">2fa.required</arg>
   <arg name="TaskData">${arg.2fa.task.data}</arg>
  </function>
  <function type="removeAuthTokenProperty">
   <arg name="PropertyName">2FAData</arg>
  </function>
  <function type="terminateSession"/>
 </pre-functions>
</result>
```

User Workflow: Variable Resolvers

The following variable resolvers are available for the user workflow.

- ${business.twofa.recurrence.mode} on page 115
- ${business.twofa.recurrence.interval} on page 115
- ${business.twofa.device.cookie.name} on page 115
- ${business.twofa.code.validity} on page 116

${arg.xxxx}

Available in version: 7.2.4.3 and later

A dynamic variable resolver that can take any value as the argument. It resolves to a parameter provided in place of the **xxx** value, and returns the value of the parameter. For example, **${arg.2fa.task.data}** would resolve to the **2fa.task.data** parameter. If the parameter is found, the value is returned.

${authtoken.property.xxx}

Available in version: 7.2.4.3 and later

A dynamic variable resolver that can take any value as the argument. It resolves to a function provided in place of the **xxx** value, and returns the result of the function. For example, **${authtoken.property.2FAData}** would resolve to the 2FA data.

${cookie.xxx}

Available in version: 7.2.4.3 and later

A dynamic variable resolver that can take any value as the argument. It looks for the cookie specified in the xxx value, and returns the value of the cookie.

${sessionuser.xxx}

A dynamic variable resolver that looks for the specified sessionuser information and returns the value of it. It looks for the specific value provided in place of the **xxx**, and returns the value of it.

The valid values for **xxx** in the above—the sessionuser information that can be checked in this variable resolver—are:

- **${sessionuser.phone}**

  Looks for the user's phone number and returns the value of it.

- **${sessionuser.firstname}**

  Looks for the user's first name and returns the value of it.

- **${sessionuser.lastname}**

  Looks for the user's last name and returns the value of it.

- **${sessionuser.email}**

  Looks for the user's email address and returns the value of it.

- **${sessionuser.username}**

  Looks for the username and returns the value of it.

- **${sessionuser.name}**

  Looks for the user's profile name (same as username) and returns the value of it.

- **${sessionuser.domain}**

  Looks for the user's domain and returns the value of it.

${business.twofa.maxattempts}

Available in version: 7.2.4.3 and later

Checks the business settings for the platform tenant, as specified by the Site Admin in the 2FA settings, to determine the maximum number of attempts allowed with a single two-factor verification code.

${business.twofa.recurrence.mode}

Available in version: 7.2.4.3 and later

Checks the business settings for the platform tenant, as specified by the Site Admin in the 2FA settings, to determine the recurrence mode—that is, the frequency with which the user must perform two-factor authentication. There are three options for recurrence mode:

- **twofa.each.login**: each time the user logs in.
- **twofa.login.new.device**: the first time the user logs in on a different device (includes cookie name).
- **twofa.once.each.interval**: after the specified time interval has passed (cookie name and also time interval setting).

${business.twofa.recurrence.interval}

Available in version: 7.2.4.3 and later

Checks the business settings for the platform tenant, as specified by the Site Admin in the 2FA settings, to determine the recurrence interval. Only applicable when the setting for recurrence mode is **twofa.once.each.interval**. For example, the interval might be 15 days.

${business.twofa.device.cookie.name}

Available in version: 7.2.4.3 and later

Checks the business settings for the platform tenant, as specified by the Site Admin in the 2FA settings, to determine the cookie name. Only applicable when the setting for recurrence mode is **twofa.once.each.interval** or **twofa.login.new.device**.

${business.twofa.code.validity}

Available in version: 7.2.4.3 and later

Checks the business settings for the platform tenant, as specified by the Site Admin in the 2FA settings, to determine how long the 2FA verification code is valid for. Code validity period is specified in milliseconds.

User Workflow: Implementing Two-Factor Authentication

The platform supports setting up a second factor, a verification code generated and sent to the user, as an additional security feature on the login process.

There is a specific version of the user workflow that is provided out of the box to help get you started on setting up two-factor authentication for login.

Two-factor authentication is also supported by additional platform settings specified by the Site Admin (**Administration** > **Settings** > **User 2FA**).

Workflow Example Implementing Two-Factor Authentication

The example below shows the first action of a step in a custom User Workflow that implements two-factor authentication for registered users. This workflow:

- Sets the 2FA delivery options that will be available to users.

  Note: If you want to use the platform default, leave **EmailSupported** set to **true** and set the other two options to **false**.

- Tests to see whether the authentication cookie already exists. If so, the workflow step ends.
- Tests to make sure that the custom property 2FAComplete does **not** exist in the user's auth token; that is, the two-factor authentication process is not complete. If this property does exist, the workflow step ends.
- Generates the verification code for the second factor of the authentication process.
- Generates a user notification.
- Sets "2FA required" as a pending task. The user must enter the code as part of the login process in order for login to be complete.

```
<action id="451" name="@Login">
  <pre-functions>
    <function type="setTwofaDeliveryOptions">
      <arg name="VoiceSupported">true</arg>
      <arg name="TextSupported">true</arg>
      <arg name="EmailSupported">true</arg>
    </function>
  </pre-functions>
  <results>
    <result old-status="registered" status="registered" step="450">
      <conditions type="AND">
        <condition type="is2FAEnabled"/>
        <condition type="is2FARequired"/>
```

```
    <condition type="isDeliveryOptionsOnlyOne">
      <arg name="PropertyName">${arg.2fa.task.data}</arg>
    </condition>
    <condition negate="true" type="isNoDeliveryOption">
      <arg name="PropertyName">${arg.2fa.task.data}</arg>
    </condition>
    <condition type="isDeliveryOptionsOnlyEmail"><!-- out of the box support -->
      <arg name="PropertyName">${arg.2fa.task.data}</arg>
    </condition>
    <condition negate="true" type="authTokenPropertyMatches">
       <arg name="PropertyName">2FAComplete</arg>
       <arg name="PropertyValue">Yes</arg>
    </condition>
    <condition negate="true" type="authTokenPropertyMatches">
       <arg name="PropertyName">2FASkipped</arg>
       <arg name="PropertyValue">Yes</arg>
    </condition>
    <condition negate="true" type="authTokenPropertyExists">
       <arg name="PropertyName">2FAData</arg>
    </condition>
  </conditions>

  <pre-functions>
    <function type="setTwoFADeliveryTarget">
      <arg name="DeliveryOptions">${arg.2fa.task.data}</arg>
      <arg name="DeliveryMechanism">Email</arg>
      <!--<arg name="Index">0</arg>--><!-- Either or -->
    </function>
    <function type="generate2FACode"/>
    <function type="setAuthTokenProperty">
      <arg name="PropertyName">2FAData</arg>
      <arg name="PropertyValue">${arg.2fa.authtoken.property}</arg>
    </function>
    <!-- Implement sendToPhone custom function similar to this function -->
    <function type="send2FACodeToEmail">
      <arg name="DeliveryTargetValue">${arg.twofa.delivery.target.val}</arg>
      <arg name="DeliveryMechanism">${arg.DeliveryMechanism}</arg>
      <arg name="2FAData">${arg.2fa.task.data}</arg>
      <arg name="2FAVerificationCode">${arg.verificationCode}</arg>
      <arg name="2FAVerificationCodeValidMinutes">${arg.2fa.verification.code.valid.minutes}</arg>
    </function>
    <!-- <function type="sendNotification">
      <arg name="notificationType">com.soa.notification.type.user.2fa.verification.code</arg>
      <arg name="role">Self</arg>
    </function> -->
    <function type="setPendingTask">
      <arg name="PendingTask">2fa.required</arg>
      <arg name="TaskData">${arg.2fa.task.data}</arg>
    </function>
  </pre-functions>
</result>
<result old-status="registered" status="registered" step="450">
  <conditions type="AND">
    <condition type="is2FAEnabled"/>
    <condition type="is2FARequired"/>
    <condition type="isDeliveryOptionsOnlyOne">
      <arg name="PropertyName">${arg.2fa.task.data}</arg>
    </condition>
    <condition negate="true" type="isNoDeliveryOption">
      <arg name="PropertyName">${arg.2fa.task.data}</arg>
```

```
    </condition>
    <condition type="isDeliveryOptionsOnlyText"><!-- out of the box support -->
      <arg name="PropertyName">${arg.2fa.task.data}</arg>
    </condition>
    <condition negate="true" type="authTokenPropertyMatches">
       <arg name="PropertyName">2FAComplete</arg>
       <arg name="PropertyValue">Yes</arg>
    </condition>
    <condition negate="true" type="authTokenPropertyMatches">
       <arg name="PropertyName">2FASkipped</arg>
       <arg name="PropertyValue">Yes</arg>
    </condition>
    <condition negate="true" type="authTokenPropertyExists">
       <arg name="PropertyName">2FAData</arg>
    </condition>
  </conditions>

  <pre-functions>
    <function type="setTwoFADeliveryTarget">
      <arg name="DeliveryOptions">${arg.2fa.task.data}</arg>
      <arg name="DeliveryMechanism">Text</arg>
      <!--<arg name="Index">0</arg>--><!-- Either or -->
    </function>
    <function type="generate2FACode"/>
    <function type="setAuthTokenProperty">
      <arg name="PropertyName">2FAData</arg>
      <arg name="PropertyValue">${arg.2fa.authtoken.property}</arg>
    </function>
    <!-- <function type="send2FACodeToPhone"> custom workflow function to send text
      <arg name="DeliveryTargetValue">${arg.twofa.delivery.target.val}</arg>
      <arg name="DeliveryMechanism">${arg.DeliveryMechanism}</arg>
      <arg name="2FAData">${arg.2fa.task.data}</arg>
      <arg name="2FAVerificationCode">${arg.verificationCode}</arg>
      <arg name="2FAVerificationCodeValidMinutes">${arg.2fa.verification.code.valid.minutes}</arg>
    </function> -->
    <function type="setPendingTask">
      <arg name="PendingTask">2fa.required</arg>
      <arg name="TaskData">${arg.2fa.task.data}</arg>
    </function>
  </pre-functions>
</result>
<result old-status="registered" status="registered" step="450">
  <conditions type="AND">
    <condition type="is2FAEnabled"/>
    <condition type="is2FARequired"/>
    <condition type="isDeliveryOptionsOnlyOne">
      <arg name="PropertyName">${arg.2fa.task.data}</arg>
    </condition>
    <condition negate="true" type="isNoDeliveryOption">
      <arg name="PropertyName">${arg.2fa.task.data}</arg>
    </condition>
    <condition type="isDeliveryOptionsOnlyVoice"><!-- out of the box support -->
      <arg name="PropertyName">${arg.2fa.task.data}</arg>
    </condition>
    <condition negate="true" type="authTokenPropertyMatches">
       <arg name="PropertyName">2FAComplete</arg>
       <arg name="PropertyValue">Yes</arg>
    </condition>
    <condition negate="true" type="authTokenPropertyMatches">
       <arg name="PropertyName">2FASkipped</arg>
```

```
        <arg name="PropertyValue">Yes</arg>
      </condition>
      <condition negate="true" type="authTokenPropertyExists">
        <arg name="PropertyName">2FAData</arg>
      </condition>
    </conditions>

    <pre-functions>
      <function type="setTwoFADeliveryTarget">
        <arg name="DeliveryOptions">${arg.2fa.task.data}</arg>
        <arg name="DeliveryMechanism">Voice</arg>
        <!--<arg name="Index">0</arg>--><!-- Either or -->
      </function>
      <function type="generate2FACode"/>
      <function type="setAuthTokenProperty">
        <arg name="PropertyName">2FAData</arg>
        <arg name="PropertyValue">${arg.2fa.authtoken.property}</arg>
      </function>
      <!-- <function type="send2FACodeToPhone"> custom workflow function to send voice
        <arg name="DeliveryTargetValue">${arg.twofa.delivery.target.val}</arg>
        <arg name="DeliveryMechanism">${arg.DeliveryMechanism}</arg>
        <arg name="2FAData">${arg.2fa.task.data}</arg>
        <arg name="2FAVerificationCode">${arg.verificationCode}</arg>
        <arg name="2FAVerificationCodeValidMinutes">${arg.2fa.verification.code.valid.minutes}</arg>
      </function> -->
      <function type="setPendingTask">
        <arg name="PendingTask">2fa.required</arg>
        <arg name="TaskData">${arg.2fa.task.data}</arg>
      </function>
    </pre-functions>
  </result>
  <result old-status="registered" status="registered" step="450">
    <conditions type="AND">
      <condition type="is2FAEnabled"/>
      <condition type="is2FARequired"/>
      <condition type="isDeliveryOptionsMultiple">
        <arg name="PropertyName">${arg.2fa.task.data}</arg>
      </condition>
      <condition negate="true" type="authTokenPropertyMatches">
        <arg name="PropertyName">2FAComplete</arg>
        <arg name="PropertyValue">Yes</arg>
      </condition>
      <condition negate="true" type="authTokenPropertyMatches">
        <arg name="PropertyName">2FASkipped</arg>
        <arg name="PropertyValue">Yes</arg>
      </condition>
      <condition negate="true" type="authTokenPropertyExists">
        <arg name="PropertyName">2FAData</arg>
      </condition>
    </conditions>
    <pre-functions>
      <!-- returns delivery options -->
      <function type="setPendingTask">
        <arg name="PendingTask">2fa.required</arg>
        <arg name="TaskData">${arg.2fa.task.data}</arg>
      </function>
    </pre-functions>
  </result>
  <result old-status="registered" status="registered" step="450">
    <conditions type="AND">
```

```
    <condition type="is2FAEnabled"/>
    <condition type="is2FARequired"/>
    <condition type="isNoDeliveryOption">
      <arg name="PropertyName">${arg.2fa.task.data}</arg>
    </condition>
    <condition negate="true" type="authTokenPropertyMatches">
       <arg name="PropertyName">2FAComplete</arg>
       <arg name="PropertyValue">Yes</arg>
    </condition>
    <condition negate="true" type="authTokenPropertyMatches">
       <arg name="PropertyName">2FASkipped</arg>
       <arg name="PropertyValue">Yes</arg>
    </condition>
    <condition negate="true" type="authTokenPropertyExists">
       <arg name="PropertyName">2FAData</arg>
    </condition>
  </conditions>
  <pre-functions>
    <function type="generate2FACode"/>
    <function type="setAuthTokenProperty">
      <arg name="PropertyName">2FAData</arg>
      <arg name="PropertyValue">${arg.2fa.authtoken.property}</arg>
    </function>
    <!-- Add a default delivery option here as there are no delivery options defined, this has to be custom -->
    <!-- <function type="send2FACodeToPhone"> custom workflow function
      <arg name="userPhoneNumber">${sessionuser.phone.number}</arg>
      <arg name="2FAData">${arg.2fa.task.data}</arg>
      <arg name="2FAVerificationCode">${arg.verificationCode}</arg>
      <arg name="2FAVerificationCodeValidMinutes">${arg.2fa.verification.code.valid.minutes}</arg>
    </function> -->
    <function type="setPendingTask">
      <arg name="PendingTask">2fa.required</arg>
      <arg name="TaskData">${arg.2fa.task.data}</arg>
    </function>
  </pre-functions>
</result>
<result old-status="registered" status="registered" step="450">
  <conditions type="AND">
    <condition type="is2FAEnabled"/>
    <condition negate="true" type="authTokenPropertyMatches">
       <arg name="PropertyName">2FAComplete</arg>
       <arg name="PropertyValue">Yes</arg>
    </condition>
    <condition negate="true" type="authTokenPropertyMatches">
       <arg name="PropertyName">2FASkipped</arg>
       <arg name="PropertyValue">Yes</arg>
    </condition>
    <condition type="authTokenPropertyExists">
       <arg name="PropertyName">2FAData</arg>
    </condition>
  </conditions>
  <pre-functions>
    <function type="unmarshall2FACode"/>
    <function type="setPendingTask">
      <arg name="PendingTask">2fa.required</arg>
      <arg name="TaskData">${arg.2fa.task.data}</arg>
    </function>
  </pre-functions>
</result>
<result old-status="registered" status="registered" step="450">
```

```
    <conditions type="AND">
      <condition type="isLocalDomainUser"/>
      <condition type="isChangePasswordRequired"/>
    </conditions>
    <pre-functions>
      <function type="setProperty">
        <arg name="PendingTask">&ChangePassword;</arg>
      </function>
    </pre-functions>
  </result>
  <result old-status="registered" status="registered" step="450">
    <conditions type="AND">
      <condition negate="true" type="isChangePasswordRequired"/>
      <condition negate="true" type="agreementsAccepted"/>
    </conditions>
    <pre-functions>
      <function type="setProperty">
        <arg name="PendingTask">&ForceAcceptAgreements;</arg>
      </function>
    </pre-functions>
  </result>
  <result old-status="registered" status="registered" step="450">
    <conditions type="AND">
      <condition type="isLocalDomainUser"/>
            <condition negate="true" type="isChangePasswordRequired"/>
      <condition type="agreementsAccepted"/>
      <condition negate="true" type="securityQuestionsAnswered"/>
      <condition type="isForceChallengeQuestionsAnsweredOnLoginSetup"/>
    </conditions>
    <pre-functions>
      <function type="setProperty">
        <arg name="PendingTask">&CollectSecurityQuestionAnswers;</arg>
      </function>
    </pre-functions>
  </result>
  <unconditional-result old-status="registered" status="registered" step="450">
    <pre-functions>
      <function type="setProperty">
        <arg name="LoginState">&LoginComplete;</arg>
      </function>
      <!--  invoke send Notification on first time login.  -->
      <function type="markLoginComplete"/>
    </pre-functions>
  </unconditional-result>
 </results>
</action>
```

Chapter 9 | 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 Akana API Platform workflows relating to reviews are:

- @StartReview

@StartReview

Available in version: 7.1 and later

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.

Review Workflow: Reserved Actions

The following reserved actions are defined for Akana API Platform workflows relating to reviews:

- @read on page 122
- @modify on page 122
- @cancel on page 123

@read

Available in version: 7.1 and later

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

Available in version: 7.1 and later

Used to determine whether a review can be modified.

@cancel

Available in version: 7.1 and later

Used to cancel the review and end the workflow.

Review Workflow: Functions

The following functions are available for Akana API Platform workflows relating to reviews:

markPublished

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Note: The email isn't sent instantly; it is queued to be sent. It goes to the notifications queue, and the job runs every 60 seconds. There might be a short delay before the user receives the email.

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>
```

Review Workflow: Conditions

The following conditions apply to the reviews workflow:

- isAutoPublishEnabled on page 127

isAutoPublishEnabled

Available in version: 7.1 and later

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

Review Workflow: Variable Resolvers

There are currently no variable resolvers for Akana API Platform workflows relating to reviews.

Chapter 10 | 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 Akana API Platform workflows relating to discussions are:

@StartDiscussion

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

@Alert

Available in version: 7.1 and later

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 Akana API Platform workflows relating to discussions:

- @read

@read

Available in version: 7.1 and later

Applies to unpublished discussions only.

This action controls who can read an unpublished 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 Akana API Platform workflows relating to discussions:

MarkPublished

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Note: The email isn't sent instantly; it is queued to be sent. It goes to the notifications queue, and the job runs every 60 seconds. There might be a short delay before the user receives the email.

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

Available in version: 7.1 and later

Checks the platform settings to determine 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 Akana API Platform workflows relating to discussions.

Chapter 11 | 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 Akana API Platform workflows relating to comments are:

@AddComment

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

@Alert

Available in version: 7.1 and later

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

@read

Available in version: 7.1 and later

Applies to unpublished comments only. This action controls who can read an unpublished 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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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 on page 138
- isCommentAutoPublishEnabled on page 139

authorizeByCMRole

Available in version: 7.1 and later

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

Available in version: 7.1 and later

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 Akana API Platform workflows relating to comments.