

Actions, Triggers, and Conditions

Document ID: Q000075

Last Revised On: Monday, January 12, 2009

This article applies to the following:

Product Version: IssueNet 4.0 and later

Component(s): Architect, Administrator

Solutions(s): All

Actions, triggers and conditions are the building blocks from which IssueNet workflows are built. When combined together, these elements provide IssueNet solution designers with near infinite flexibility in workflow design. By mastering actions conditions and triggers, a solution designer can use a single set of skills to create:

- Task workflows in which each task has definitive steps and rules for completion
- Event driven actions that send notifications and automate workflow steps
- Jobs that perform scheduled tasks and implement escalation rules
- Rules that determine forms and fields displayed for different users and groups

Although actions, conditions, and triggers are themselves very simple, some of the techniques for using them in different contexts are not obvious. In addition, some of the more advanced uses of actions, conditions, and triggers require attention to the application context. As a user performs different kinds of actions in an IssueNet solution, different kinds of information are available. For example, an action or condition that works as expected when a user is creating a new issue may not have the same effect when applied to a workflow transition.

The purpose of this month's TechTip is to explain the most important techniques for using actions, conditions and triggers in a variety of common contexts. The techniques you will learn are based on principles you can use in developing your own workflows and business rules.

Using Actions and Conditions in Workflows

The best description of an IssueNet workflow is that it is a set of states and transitions that defines the business process for a task. One or more task workflows define the business process for the issue the tasks are linked to. This approach is very similar to documenting a business process using a cross-functional flowchart. As a cross-functional flow chart organizes parts of a complete workflow into functional roles, IssueNet organizes the complete workflow for an issue into a set of workflow steps defined for each task.

Workflow Execution and Context

To understand how to best use actions and conditions in workflows it is important to understand how workflows and workflow steps are executed as well as the information the workflow context defines. On one level an IssueNet workflow is like a macro or program. One key difference is that the steps in a workflow are always executed in the context of a task. Let's consider a task that assigns a user to complete the steps required to review and approve or disapprove work on an issue. When the task is created it is assigned to a specific resource and is associated with a workflow. To complete the task the user selects transitions that 'move' the task from one workflow state to another until the task is completed. For example, at a certain point in the workflow the user may be allowed to transition the task from a state of "Reviewing" to "Completed" by picking one of two transition which indicates whether the work has been "Approved" or "Declined".

In this type of workflow, actions occupy two important roles:

- One or more actions create the tasks(s) the workflow will be transitioned from.
- Tasks linked to states and transitions automate parts of the workflow process such as sending notifications, updating issue status, or creating other tasks.

When a workflow is executed actions added to the workflow diagram are executed first. If a task is created, the workflow state for the task is set to the entry state of the workflow. Actions added to specific states are transitions are executed when a user, or an automated process, executes transitions which move the task from one workflow state to another.

If you open a typical IssueNet workflow and click on the workflow diagram, states, and transitions, you will observe in the workflow properties panel:

- At least one create object action on the workflow canvas that creates the task that will be governed by the workflow
- Actions on workflow states and transitions that automate portions of the workflow
- Conditions that must return a true value to allow the transition to be completed

To better understand how these elements are combined together to form a complete workflow, review the elements of the **Review Software Issue** workflow and **Implement Software Change** workflows in the IssueNet Intercept solution. These workflows implement the process of reviewing and implementing a change to a software product before it is sent to QA. You can open these workflows using the IssueNet Administrator to review the workflow design or execute the workflow steps using the IssueNet Manager or Workspace integration. The basic steps in initiating a progressing through a workflow are:

- An issue such as a defect or enhancement is submitted into the Intercept solution.
- Once an issue has been created, a workflow can be executed on the issue. A workflow can be executed either automatically or using a trigger or manually by selecting the issue and executing the workflow.
- Once the workflow is executed, actions added to the workflow are executed first. These actions will typically include a Create Object action that will create the task from which the workflow

will be transitioned. If you execute the Review Software Issue workflow, a Create Object action will create a task set for that workflow and assign it to the correct resource.

- Once a task has been created its workflow state will be set to the entry state for the workflow and subsequent actions will be executed either directly or indirectly as the user selects workflow transitions to indicate steps completed and decisions made during the workflow.

Variables and Workflow Context

If you open practically any action linked to a workflow you will see that most or all of the action properties are specified using variables. Variables will be discussed in more detail. However, when discussing workflow, it is worth noting that during the execution a workflow and workflow transitions there are certain variables which are specific to the workflow context. Whenever a workflow or workflow transition is executed, there is always a task and an issue specified. There is also, optionally, a project for the tasks. Accordingly, when executing a workflow or workflow transition, you can use the following variable types:

\$(WorkflowIssue)

\$(WorkflowTask)

\$(WorkflowProject)

Using the \$(WorkflowIssue) variable you can, for example, easily update the issue status or include the issue status in a notification triggered by a workflow transition by specifying it as \$(WorkflowIssue.Status). There are other ways to access the same information. However, the workflow context makes it easier to get the information through simple variables.

Triggers

IssueNet formally defines workflow as a series of states and transitions that govern the completion of a task. However, workflow as generally understood encompasses more than the steps for a task. A complete workflow process will typically involve:

- Sending notifications when a field changes from one value to another
- Automatically executing workflows when certain issue types are created
- Escalating issues when certain criteria are met

For these automated tasks which fall outside the context of a task workflow there are Triggers. Every trigger has three elements:

- The trigger event - The event determines when the trigger is executed and the condition is evaluated.
- A condition – The condition is evaluated when the trigger event occurs. If the condition returns a true value, the trigger actions are executed.

- One or more actions – The trigger actions are executed in the order specified in the trigger when the condition returns a true value

The only behavior that is specific to triggers is the event. To use triggers properly it is important to understand the functions of the following event types:

- AfterObjectInsert, AfterObjectUpdate, and AfterObjectDelete – These are the most commonly used events. Generally, when you want an action to occur when an item is created or a value is set use one of these events.
- BeforeObjectInsert, BeforeObjectUpdate, and BeforeObjectDelete – These events play a specialized role. Before an object is updated the application has access to values and item has before the update as well as the values properties will be updated to. Therefore, the BeforeObjectUpdate event is used when you want to use the trigger condition to compare property values to detect a specific kind of value change. For example, if you wanted to trigger a notification when the priority of an issue changed from “Urgent” to some other value, you would create a trigger which would use the BeforeObjectUpdate event and a condition which used operators to compare the current and original values. BeforeObjectInsert and BeforeObjectDelete events are included for completeness but are rarely, if ever, used. More information on the comparison operators is below.
- Asynchronous - As the name suggests, asynchronous triggers are not linked to specific events for execution. These triggers are executed on demand by services. IssueNet Monitor jobs can be configured to execute triggers when the Monitor Job identifies items such as tasks or issue that have exceeded a particular service level agreement threshold.

Current and Original Operators with Triggers

One set of operators is specific to use with triggers. These operators are: **Current** and **Original**. When used in a trigger set for the BeforeObjectUpdate event they can be used to execute actions when a specific value change is detected. To use the previous example, the following condition would return true if the priority of an issue changed from “Urgent” to another value:

If $\$(CurrentObject.Priority:Current)$ is not equal to $\$(CurrentObject.Priority:Original)$

Conditions

Conditions are used in conjunction with a variety of IssueNet workflow features. A condition is simply a logical statement IssueNet can evaluate to a true or false value. Therefore conditions are used whenever you need IssueNet to determine if something is true. Conditions are used to:

- Determine if a workflow transition should be allowed
- Determine if trigger actions should be executed
- Determine if a user should be allowed to view a particular form or list view definition

Because conditions use standard IssueNet variables, constructing them is simply the process of using the condition builder to create a logical statement where the values are specified using variables and

literals. The only features specific to conditions are some of the comparison operators. Most of the operators are common logical operators. However, some of the operators leverage features specific to IssueNet:

is group member – allows a condition to compare the current users group membership to a specified group.

is kind of – allows you to compare the class membership of an object to a specified value. The operation of this operator is recursive. If a condition uses this operator to determine if an object is a member of the issue class it will return a true value for objects that are members of sub-classes.

all elements equal, one or more elements equal, no elements equal – compares the values of items in a collection to a specified value. These operators are particularly useful for conditions that are used in workflow approval processes. Using the all elements equal operator you can, for example, determine if all of the tasks for an issue are completed. That condition could be specified as:
\$(WorkflowIssue.Tasks.CurrentState) all elements equal "Completed".

For more information on topics not covered by this Tech Tip please review video tutorials and TechTips at www.elsitech.com. If you have questions beyond the scope of this Tech Tip, you may also contact Elsinore Technical Support Services at support@elsitech.com or 866.866.0034, option 2.