Triggers

Overview

Triggers, implemented with the **{trigger}** macro, wait for Events then perform one or more Actions.

Multiple triggers can listen to the same event, and by using filters you can perform different sets of actions for different situations relating to that event.

Filtering events

Events are generic. For example, the **statechanged** event will happen every time there is a state transition, regardless of which state is being transitioned to or from. So, the trigger has to do some filtering to make sure it's acting not just on the correct event, but also that it's acting in the right situation in relation to that event. For example, you might want to trigger the actions only when the workflow is in a specific state, or when the current user is in a specific group.

To achieve this, the **{trigger}** macro has a number of parameters which allow it to filter events for common situations. In the case of the **statechanged** event, for example, we want to know which **state** the transition has sent us to – because that determines which actions should be performed.

```
{workflow:Filters}
   {state:First}
   {state}
   {state:Second}
   {state}
   {state:Third}
   {state}
   {trigger:statechanged|state=Second}
      {set-message:duration=PT1M}
         We're in the second state
      {set-message}
   {trigger}
{workflow}
```

Example – filtering by `state`

In this example above, each state can transition to any of the other states, so the **statechanged** trigger will be checked every time. However, we've filtered the trigger to only act when the current **state** is called **Second**.

Required filters

Most of the filters are optional, however some are required depending on which event you are listening to – these are marked with ℹ in the **{trigger}** macro documentation.

The **state** filter, for example, is mandatory if you are listening to a **statechanged** event. It can still be used, optionally, for all other events, but there are also events where it must always be used.

Event-specific filters

Just as some events require specific filters, some filters require specific events. One example is the **label** filter which can only be used with events relating to labels – because those events send details of the label which was added or removed at that point in time.

Condition filters

Triggers support the use of Conditions as a filtering mechanism which can be used for any event. For example, you could use the **haslabel** condition on any of the available events. For example:

```
{workflow:name=haslabel condition}
   {state:First}
   {state}
   {state:Second}
   {state}
   {trigger:statechanged|state=Second|haslabel=test}
      {set-message:duration=PT1M}
         We are in Second state, and the page has the "test" label
      {set-message}
   {trigger}
{workflow}
```

Special filters

There are some special filters which go beyond simple comparison checks. They are:

- **`initial`** – unique to the **`statechanged` event**, this filter can be used to make the trigger act only on the first occasion that a state is entered for a given piece of content
- **`partial`** – unique to **content review events**, this filter can tell the trigger to act on each individual Approve or Reject, rather than waiting for the review to be completed
- **`success`** – unique to **custom events**, this filter looks at whether the actions of the parent trigger encountered any errors (see **Error handling** section below)

Custom events

Triggers can be configured to create their own events, which can then be processed by other triggers.

This is achieved using the **`newevent`** parameter, for example:

```
{workflow:name=Custom Events}
   {state:First}
   {state}
   {state:Second}
   {state}
   {trigger:statechanged|state=First|newevent=FirstAndSecond}
   {trigger}
   {trigger:statechanged|state=Second|newevent=FirstAndSecond}
   {trigger}
   {trigger:FirstAndSecond}
      {set-message:duration=PT1M}
         Current state is: @state@
      {set-message}
   {trigger}
{workflow}
```

In the example above, the first two triggers both listen to the **`statechanged`** event. The first trigger is filtered to the **`First`** state, the second to the **`Second`** state. Both of them will create a **`newevent`** of the same name: **FirstAndSecond**. The third trigger listens for that **`FirstAndSecond`** event and then displays a message containing the name of the current workflow state (using the pre-defined **`@state@`** value reference). In essence, we are routing the **`statechanged`** events for two states, **`First`** and **`Second`**, in to a single **FirstAndSecond** trigger, and then using that trigger to perform the action.

Note: Event-specific Event references are not available in custom event triggers. For example, the **`@comment@`** reference, which would be available in the first two triggers, would not be available in the third trigger.

Queuing actions

When an event is received by a trigger, it will check it's filters and only then perform it's actions. It will perform each action in turn, one after the other. In most cases, this won't be a problem - because most actions happen very quickly, so the user won't have to wait before continuing their work.

However, there are some actions which may take longer to complete, for example:

- Publishing actions, particularly for Remote-space publishing actions **`{remotepublish-page}`** and **`{remoteremove-page}`** macros.
- Sending lots of emails to a user group via the **`{send-email}`** macro
- Approving or rejecting large numbers of child pages using the **`{approve-children}`** or **`{reject-children}`** macros

In these cases, the user would have to wait for the actions to complete, before they could continue their work. To overcome this issue, you can **`queue`** the actions so the user can get on with their work whilst the actions are performed:

```
{workflow:name=Queued actions}
  {state:Editing|submit=Published}
  {state}
  {state:Published}
  {state}
  {trigger:statechanged|state=Published|queue=true}
     {remotepublish-page:ToCloud}
  {trigger}
{workflow}
```

Error handling

Triggers which handle custom events can use the **`success`** parameter to filter depending on whether the actions of the parent trigger (that created the custom event) encountered errors.

Actions are very reliable, so errors are extremely rare, however there is an increased chance of error when using Remote-space publishing – for example, the remote Confluence instance might be down for maintenance, or there might be a problem with the internet connection.

For an example of error handling, see: Advanced remote-space publishing

Trigger inspector

If the workflow is used in a space that runs in Page Mode, you can view a summary of its triggers (and the actions they will perform) in the Workflow inspector.

Macros

- **`{trigger}`**
- For a list of action macros, see: Actions

Examples

- Add, remove and set page restrictions
- Adding page activity to email — Include page activity report, or a link to it, in email notifications
- Advanced different-space publishing
- Advanced remote-space publishing
- Attachment events — Triggering events when attachments are created, updated or removed.
- Blog Post Events — Workflow events associated with blog posts
- Fast-tracked Rejections — How to require everyone to Approve, but only need one person to Reject
- Message notification styles — Test the **`style`** parameter of the **`{set-message}`** macro
- Require Parameters on State Transitions — Require workflow parameter values to be set before moving into a workflow state.
- Space mode workflow application
- State expiry dates — Using state expiry dates, defining them with metadata, and making them editable
- StiltSoft Talk app
- Template Comments — Examples of where the **`{comment}`** macro can be used in your template markup.
- Transitions — Transitions create the routes between states

See also

Workflow Authoring Guide:

- Events
- Actions
- Notifications