Understanding the Ad hoc Workflow Engine

This article covers the theory behind the Ad hoc Workflows engine. This article is not intended as a tutorial per se, but rather as a backgrounder to help workflow designers understand how the engine works. For detail examples see Tutorials.

- What are Ad hoc Workflows
- Workflow as a State Machine
- What is a State Machine?
- Types of Defined Workflows
- How to Define a Workflow
  - Step 1 – Define the Needed States
  - Step 2 – Define the Needed Tasks for Each State
  - Step 3 – Define How the Article Transitions between States
  - Step 4 – Define How to Initiate the Workflow
  - Step 5 – Determine How Articles will be Published
  - Step 6 – Define How the Workflow Recycles
- Basic Workflow Structure
- Using Metadata
- Workflow versus Action Macros
- Approval versus Assignment versus Task
- Limitations

What are Ad hoc Workflows

The Ad hoc Workflows plug-in delivers a collection of macros that can be used to generate custom workflows for use with Confluence pages. Many business processes require some sort of formalized workflow for creating, approving and publishing documents, whether for external documentation, internal specifications or for policies and procedures. The Ad hoc Workflows macros help automate these processes without the need of complex programming languages.

Workflow as a State Machine

The macros used in Ad hoc Workflows are basically a state machine description language (see sidebar). An Ad hoc workflow first defines each state a page must pass through in a process (using a state). For example, a simple process for a sales plan might include: draft, review, VP approval, final. For each of these states, a workflow designer can define the criteria (events) to move from each state to the next. For example, in the draft state, the article sales person may be assigned tasks to complete (using a task), such as “list each customer opportunity,” “briefly describe each opportunity,” and “define the revenue potential.”

**Note**

A page which has an Ad hoc workflow applied to it must always be in a defined state. There is no default state — a page cannot be in limbo.
Once the sales person has checked off as having completed each of the items, the page can then progress to the Review state, possibly by a sales manager. In this case, the review is viewed as a type of approval (using the approval). The sales manager can either approve the sales plan (sending it for VP Approval) or reject the sales plan with a comment (sending it back to draft). VP Approval is handled similarly, using another instance of the approval macro. If the VP rejects, the workflow can be designed to either return the sales plan back to either review or draft. If approved, it can advance to the last state, Final.

The Ad how Workflow macros allow for more complex definitions of state transition events than described here. For example, a state can transition because it has exceeded a due date, a user has updated a page, or added an attachment. These events can also be used to trigger/execute (using the trigger) a number of other operations. Similarly, approvals can be either simple, or more complex, for example, requiring all members of a group to approve, requiring a special sequence of approvals, etc. The macro language even allows a workflow to publish an article automatically (see Understanding Publishing).
What is a State Machine?

A traffic light can be used as a simple example of a state machine. A traffic light (in the US) has to be in one of three states:

- State 1 - red
- State 2 - green
- State 3 - amber

The state diagram below illustrates each state and defines the triggers for the traffic light to transition to the next color or state. Each state has a timer to determine how long the traffic light should remain in each color. For example, this traffic light remains red for 20 seconds. When the timer reaches 20, the light transitions from red to green and resets the timer. While green, once the timer reaches 15 secs, the light transpositions to amber.

In real life, the state machine for a modern traffic light is more complex as it must control the lights in both directions, plus must add a few additional conditions:

- Left turns
- Pedestrian crossings
- Flashing amber/red for late hours
- Emergency vehicles

Types of Defined Workflows

There are two basic types of workflows, which differ not in how they are coded, but in how they are applied to pages:

- Page-level workflows – this type of workflow is applied by the user be selecting a workflow saved as a template in that space (see Add a workflow)
- Space-level workflows -- this type of workflow is automatically applied to all pages within a space.

How to Define a Workflow

As an example workflow, lets examine the steps to design a simple documentation process. The complete code for this sample workflow can be found in Basic Workflow Structure.

Step 1 – Define the Needed States

The first step is to determine what states are needed for the workflow. For the example documentation process, these states could be:
• Assign Author - in this state, someone responsible for the workflow (for example, a tech writer), would assign who should author the article.
• Planning - in this state the author would write and abstract and an outline to be reviewed in the next state.
• Review - an editor/manager would review the abstract and outline in this state.
• Draft - the author writes the article in this state.
• TW Review - in the state, the tech writer edits/updates
• Final Review - article is submitted to the editor/manager for approval
• Published - article is now ready for customer use.

How to Define States

Within Ad how Workflows, states are defined using the state. The macro is used to define the name of the state, as well as some of the exit/transition criteria (see How to Transition Between States below). For example, to define the Assign Author state, the syntax would be:

```
{state:Assign Author|assignable=true}
{state}
```

Clearly there is more that needs to happen in this state. How to transition between states is covered in Step 3

Step 2 – Define the Needed Tasks for Each State

For some of our states, tasks need to be assigned. Tasks are useful for when:

• More than a single action is required for a state
• Actions are required from multiple users within that state
• Providing tracking --- assigned tasks appear on a users dashboard

In our example workflow, tasks are useful for the Planning state where the page author is tasked with writing an abstract and an outline. The syntax for assign these tasks in the Planning state is:

```
{state:Planning|completed=Review}
{task:Write abstract|assignee=@pageauthor@}
{task:Create outline|assignee=@pageauthor@}
{state}
```

The syntax above defines the two tasks to a user specified by the variable pageauthor and places notifications on the user dashboard under “My Ad hoc Workflow Tasks.” Once the user marks both tasks as complete, the workflow transitions to the next state automatically.

**Note**

The variable pageauthor represents metadata set during the Assign Author state (not shown above). See Using Metadata below for a discussion variables and metadata.

Step 3 – Define How the Article Transitions between States

Next step is define how a page moves from one state to the next. There are many events that can be used to cause a state transition, among these are (this list is not exhaustive):

• A user submits a page
• A user completes all the tasks assigned in the state
• Selected users give or reject an approval
• Elapsed time
• A user manually selects a new state from a (possibly restrictive) list
• Page is created or updated
• A label is added

There are two ways to define these transitions:

• Via parameter settings used with the state macro, for example define what the next state should be when an approval is given.
• Using the set-state macro in conjunction with the trigger macro.

Defining a State Transition with the {state} Macro

For a certain class of events, how a state transitions (and to what state) is defined using a parameter setting with the macro. For example, in the sample workflow above, we need to define what happens in the TW Review. In this state, the tech writer can either accept the page, complete his/her edits and send the page for final review, or reject the page to send it back to the author for more work. This scenario can be handled with as an approval. Defining an approval requires two parts, first is using parameters of the state macro to define what the next state is after approval or rejection:
Next, the approval and who the approver is (defined by the variable tw) must be defined:

Next, the approval and who the approver is (defined by the variable tw) must be defined:

**Defining a State Transition with the {trigger} Macro**

In the sample workflow, an author needs to be assigned to the page and then have the page transition to the Planning state. Assigning the page is simple, and in the state macro:

In the sample workflow, an author needs to be assigned to the page and then have the page transition to the Planning state. Assigning the page is simple, and in the state macro:

We want the user to be presented with a dialog box to set the page author. This must be done with a {trigger} macro. We can also set a transition for a next state:

We want the user to be presented with a dialog box to set the page author. This must be done with a {trigger} macro. We can also set a transition for a next state:

The above trigger fires when the page is assigned to the author while in the Assign Author state. It uses the (set-metadata) macro to save the user name of the author to the variable pageauthor for later use. Then the code sets (transitions) the state of the page to Planning.

**Step 4 – Define How to Initiate the Workflow**

Workflows can be applied to a page in one of three ways:

- Manually, one page at a time. A user can select and apply the workflow via the Tools menu.
- Automatically at the space level, triggering on page creation.
- Selectively at the space level by triggering off a label. A user can apply a predetermined label to trigger the workflow.

In the sample workflow, assuming that the workflow has been activated at the space level (see Manage workflows), the workflow automatically initializes the page when it is created and sets the state to Assign Author, but the page has not been assigned to any user (meaning no email notification or dashboard notice). To have a newly created page be assigned to the tech writer regardless of who created the page and set an instructional message on the page, the following {trigger} code can be added.

With this code, when a page is added (created) in a space where this workflow is active, it will automatically be initialized to the Assign Author state and assigned to the tech writer, sending an email notification and adding a dashboard notice under the "My Ad hoc Workflow Tasks."

**Step 5 – Determine How Articles will be Published**

Probably the most powerful aspect of the Ad hoc Workflows engine is how it handles publishing content. The engine handles separating draft from published versions of a page by restricting the view of any user without editing privileges to only being able to see the published version of the page. The workflow engine has added flexibility in that it allows for three different scenarios for handling content creation and the display of published content to the public:
• Same space publishing – with this scenario both draft versions as well as published version of pages are held within the same Confluence space. Only users with editing privileges can view (and edit) draft version of the pages. All other users can view only approved, published versions of the same page.
• Segregated spaces – for added security, the content development space can be kept separate from the public space which holds published versions of pages. The workflow engine automatically handles pushing copies of pages and their attachments once the publishing state is reached. Content is published using the {publish-page} macro and requires the Ad hoc Publishing Plugin.
• Publishing between Confluence servers – in this scenario, content is developed on an internal Confluence server, with public content published to an external public Confluence server. This scenario offers the highest security as only public versions of articles are accessible outside the firewall. Remote publishing uses the {remotepublish-page} macro and requires the Remote Publishing Plugin.

Note
In all of these scenarios, editors (anyone with editing rights for the page) normally see the most recent version of the page. However, editors can click the Published button at the top of the page to view the published version of that page.

For the sample workflow, we will use a simple, same-space publishing scenario. Once the final approval is given, the article will be automatically published (moves to the Published state, marked as the final state of the workflow).

```confluence
{state:Final Review|approved=Published|rejected=TW Review}
  {approval:Final Approval|user=@editor@}
{state}
{state:Published|final=true}
{state}
```

### Step 6 – Define How the Workflow Recycles

To be useful to the end user, content must be maintained. Currently, the sample workflow ends at the Published state, with no provisions for how to handle an update. We could start a new page copied from the published version and start the flow again, or a better approach is to update the workflow to handle the article refresh process for us. The simplest approach is to detect when anyone with edit permissions edits that page to return the article to Draft and restart the process from there. For the workflow, a simple modification is required, adding the updated parameter:

```confluence
{state:Published|final=true|updated=Draft}
{state}
```

Now when anyone edits the page, the editable article returns to the Draft state, leaving the Published (final) version still visible to users.

### Detecting Attachment or Label changes

Another common need is to handle what happens when an attachment is updated. We'd like our workflow to detect the update and return the article back to TW Review even when the change does not involve the content on the page. By adding another trigger macro we can program this additional functionality. Ad hoc Workflows has similar functionality for detecting the change of labels on a page.

```confluence
{trigger:attachmentupdated|state=Published}
  {set-state:TW Review}
{trigger}
```

Now the sample workflow will cycle back to Draft if the page content is updated, or back to TW Review if an attachment is updated.

### Basic Workflow Structure

The structure of a workflow has four basic parts

• Workflow definition – the {workflow} macro defines a workflow, setting the name of the workflow. All code defining the workflow must be wrapped in a {workflow} macro.
• Workflow parameters (optional) – any needed workflow parameters, for example, to define users or groups of users are set using one or more {workflowparameter} macro. These parameters can be used to add configurability at the space-level for a workflow. For example, the workflow could set the parameter tw to refer to the tech writer responsible for that space. These parameters can be overridden from the space administrator's console.
• State definitions - this section defines each of the workflow states using a series of one or more {state} macros.
• Trigger definitions (optional) – to handle more complex workflow requirements, one or more {trigger} macros are used to perform a number of other functions, such as, setting messages, changing states, setting metadata, etc., based upon various events.

In code, the basic structure is as follows:

```confluence
{workflow:name=workflow_name}
{workflowparameter:parameter1_name|type=parameter_trype}
```
Using the sample workflow defined above as an example, the fully coded workflow is as follows:

```
{workflow:name=sample workflow}
{workflowparameter:tw|type=user}
{workflowparameter}
{workflowparameter:editor|type=user}
{workflowparameter}
{state:Assign Author}
    {task:Assign Author|user=@tw@}
    {task:Author}
{state}
{state:Planning|completed=Draft}
    {task:Write abstract|assignee=@pageauthor@}
    {task:Create outline|assignee=@pageauthor@}
{state}
{state:Review|approved=Draft|rejected=Planning}
    {approval:Review Approval|user=@editor@}
{state}
{state:Draft}
    {task:Complete draft|assignee=@pageauthor@}
{state}
{state:TW Review|approved=Final Review|rejected=Draft}
    {approval:TW Review Approval|user=@tw@}
{state}
{state:Final Review|approved=Published|rejected=TW Review}
    {approval:Final Approval|user=@editor@}
{state}
{state:Published|final=true|updated=Draft}
{state}
{trigger:pagecreated}
    {set-message} This page is awaiting author assignment.{set-message}
{trigger}
{trigger:taskassigned|task=Author}
    {set-metadata:pageauthor}@taskassignee{set-metadata}
{set-state:Planning}
{trigger}
{trigger:attachmentupdated|state=Published}
    {set-state:Draft}
{trigger}
{workflow}
```

Using Metadata
Often when designing a workflow, the ability to store page- and workflow-specific data is needed so that it can be retrieved later, adding to the workflow's flexibility. Typical examples for uses of metadata are for storing the names of page authors or reviewers for later use, for example, when a page is rejected in a later state and needs to be sent back to the author for rework. The sample workflow takes advantage of this ability to store the page author’s name with the \{set-metadata\}:

\{trigger\|pageassigned\|state=Assign Author\}
\{set-metadata:pageauthor\}@assignee@\{set-metadata\}
\{set-state:Planning\}
\{trigger\}

In this example, that assignee for the page is saved as the metadata value pageauthor and can me used to display a message, for example:

\{trigger\|pageassigned\|state=Assign Author\}
\{set-metadata:pageauthor\}@assignee@\{set-metadata\}
\{set-message\} The author for this page is @pageauthor@.\{set-message\}
\{set-state:Planning\}
\{trigger\}

And data saved as metadata can also retrieved for workflow reports (see Reporting)

**Workflow versus Action Macros**

There are two classes of macros:

- Workflow – these macros are used to define aspects of the workflow, such as states, tasks, and triggers (see Workflow Macros)
- Action – these are macros, typically used inside of the \{trigger\} macro, take action on pages, attachments, set messages and save metadata, etc. (see Action Macros).

**Approval versus Assignment versus Task**

There is some overlap in how approval, assignment and (a single) task function, and in some cases, these macros could be used interchangeably. Each of these macros causes a notice to be posted on the targeted user’s dashboard under “My Ad hoc Workflow Tasks” but how the user interacts with the page differs:

- Assignment – the user either has to manually reassign the page to another user. They also have the option to advance the page in the workflow, or to select another state.
- Approval – the user has to either approve or reject the approval. This will advance the workflow as defined by the workflow (could be as simple as a state change, or a more complex action defined in a \{trigger\} macro).
- Task – the user has to mark the task as complete to advance the workflow. An approval acts like a single task and can be used interchangeably.

Which method to use depends on the requirements of the workflow and how automated and tightly controlled the workflow designer wants the workflow to be.

**Limitations**

There are some limitations to the capabilities of the workflow engine. Understanding these limitations will help when designing workflows:

- Workflows – only one workflow can be active on a page at any one time.
- Reminders – there is no reminder function currently available. In some cases the duedate parameter can be used to fire trigger to send a message and reset the state.
- Boolean logic and conditionals – there is no support for Boolean operators or conditionial statements.

**Admin Mode**

Sometimes things go wrong, but there is an admin mode available for the workflow! Just add ?admin=true to the end of the page URL. This will let you jump to any state directly if you are a space administrator.