Concept - Action availability and access control

This concept page describes both how to control when an action is shown and how to restrict access to such actions.

Action availability

The availability is based on the selected item.

Both in the actionbar and in the context menu we want to show actions as enabled or disabled. We do not want to replicate the configuration in both places. Therefore we will configure it on the actions themselves.

On the action definition we need to be able to specify

- whether its available when there's no selection (the root node is selected)
- whether its available for properties
- any node types the action should only be available for

If no node types are specified the action is available for all nodes.

Action availability is determined/evaluated by ActionExecutor and implemented in AbstractActionExecutor.

Q: Should ActionExecutor.isAvailable() take a JCR item or Vaadin item? Actions are passed a Vaadin item.

Access control

Determining wether the current user has access to an action should be based on roles. We need to configure the required roles on the action definition.

If the user does not have access we will show the action as disabled in the actionbar.

If no roles are specified on the action definition it is assumed to mean that everybody has access.

This should follow the configuration style implemented in

⚠ Unable to locate Jira server for this macro. It may be due to Application Link configuration.

Mixins can also be used in node types.

Configuration and naming

~~Two proposal, either using the term restrictions or the term availability.~~ Proposal 2 is used.

Default values need to limit the amount of configuration necessary.

~~Proposal 1 - Restrictions (implemented)~~

~~Property~~	~~Meaning~~	~~Default value~~
~~restrictions.root~~	~~Restricts whether the action is available for root, true means not restricted~~	~~false (restricted)~~
~~restrictions.properties~~	~~Restricts whether the action is available for properties, true means not restricted~~	~~false (restricted)~~
~~restriction.nodeTypes~~	~~Action is restricted if the selected node is not one of the node types in list~~	~~empty (not restricted)~~
~~restriction.roles~~	~~Action is restricted if current user does not have one of the roles~~	~~empty (not restricted)~~

~~if (roles.empty || item.role in roles) && ((item is null && root) || (item.isProperty && properties) || (item.isNode && (nodeTypes.empty || item.nodeTypes in nodeTypes))~~

Proposal 2 - Availability

Property	Meaning	Default value
availability.root	Action is available for root if true	false (not available)
availability.properties	Action is available for properties if true	false (not available)
availability.nodes	Action is available for nodes if true	true (available)
availability.nodeTypes	Action is available if empty or selected node is one of the node types in list	empty (available)
availability.access.roles	Action is available if empty or current user has one of the roles	empty (available)
availability.rule	Action is available if empty or rule.isAvailable() returns true	empty (available)

if (roles.empty || item.role in roles) && ((item is null && root) || (item.isProperty && properties) || (item.isNode && nodes) || (item.isNode && (nodeTypes.empty || item.nodeTypes in nodeTypes)) && (rule is null || rule.isAvailable(item))

Proposal commentary

The term gets a bit confusing, restrictions.root sounds like it makes the action available for only root. But its really available for root AND by node types.

Rules

To allow use of advanced logic in the access evaluation without making the configuration too complicated and confusing, a new interface `AvailabilityRule` is introduced. The interface defines just one `boolean isAvailable(javax.jcr.Item item)` method. The availability definition can have a rule subnode, where the implementing class is defined (as the `class` property). If the rule is defined, the result of the `isAvailable` method is used in a logical conjunction with the result of the default availability definition evaluation.

The implementing classes may define their own configuration. The classes are in the `info.magnolia.ui.api.availability` package and in `magnolia-ui-app-pages` package.

Some proposed rules:

- `IsDeletedRule` - returns true only if the item has been marked for deletion (i.e. has the mgnl:deleted mixin node type).
- `IsNotDeletedRule`
- `AllRulesRule` - works like an AND operator, meaning all the other rules in this configuration must be satisfied
- `AllNodeTypesRule` - as the default availability nodeTypes configuration uses disjunction (OR) to evaluate, this rule will return true only if the node has all the node types (e.g. mixins) defined in the rule configuration.
- `ActivationStatusRule` - according to its configuration, the rule will return true only if the item has / has not been yet activated, alternatively if it has been (not) changed since the last activation.

- `PageHasSubPages`

Actionbar appearance

The actionbar should only show one section at a time. The section to show depends on the selected node. Therefore we need to configure:

- whether its to be shown when there's no selection (the root node is selected)
- whether its to be shown for properties
- the node types that it should only be shown for

If no node types are specified the section is not shown for nodes.

If more than one section applies the first one is used. Example use case: in the Pages app, the pageDeletedActions section has a stricter rule (these actions are available only for deleted nodes) than the pageActions section (these actions are available for any nodes). Therefore, the pageDeletedActions section must be first.

Selecting the section to show and evaluating the restrictions/availability is done in BrowserSubApp.updateActionbar(), extending classes can override this method to implement their own behaviour.

Configuration and naming

Proposal 2 (Availability) is used.

~~Proposal 1 - Restrictions (implemented)~~

~~Property~~	~~Meaning~~	~~Default value~~
~~restrictions.root~~	~~Restricts whether the section is shown for root, true means shown~~	~~false (not shown)~~
~~restrictions.properties~~	~~Restricts whether the action is available for properties, true means not restricted~~	~~false (not shown)~~
~~restriction.nodeTypes~~	~~Restricts that the section is shown only for the node types in list~~	~~empty (shown)~~

~~if (item is null && root) || (item.isProperty && properties) || (item.isNode && (nodeTypes.empty || item.nodeTypes in nodeTypes)~~

Proposal 2 - Availability

Property	Meaning	Default value
availability.root	Section is shown for root if true	false (not shown)
availability.properties	Section is shown for properties if true	false (not shown)
availability.nodes	Section is shown for nodes if true	true (shown)
availability.nodeTypes	Section is shown if empty or selected node is one of the node types in list	empty (shown)
availability.access.roles	Section is shown if user has one of the defined roles	empty (shown)
availability.rule	Section is shown if empty or rule.isAvailable() returns true	empty (shown)

if (item is null && root) || (item.isProperty && properties) || (item.isNode && nodes) || (item.isNode && (nodeTypes.empty || item.nodeTypes in nodeTypes) && (rule is null || rule.isAvailable(item))

~~Proposal 3 - Restrictions, meanings inverted~~

~~Property~~	~~Meaning~~	~~Default value~~
~~restrictions.root~~	~~Restricts whether the section is shown for root, false means shown~~	~~true (not shown)~~
~~restrictions.properties~~	~~Restricts whether the action is available for properties~~	~~true (not shown)~~

~~restriction.nodeTypes~~	~~Restricts that the section is shown only for the node types in list~~	~~empty (shown)~~

Proposal commentary

The term restrictions seems to make more sense here since we're only gonna pick one. It's more about excluding the one not to shown.

Appendix 1 - Actionbar behaviour before configuration

We control availability by hiding sections and disabling groups or individual actions.

Remember that the actionbar is organised in sections that contain groups that contain items. Each item is linked to an action. Actionbar -> sections -> groups -> items.

Sections can be shown/hidden.

Groups and items can be enabled/disabled.

Logic charts for current behaviour

Contacts App	S folderActions	G addActions	G editActions	S contactsActions	G addActions	G editActions
no selection or root	x	x		x	x	
mgnl:folder	x	x	x	x	x	
any other node				x		x

Configuration App	G addingActions	G duplicateActions	G activationActions	G importExportActions	A addFolder	A delete
no selection or root					x	
any node	x	x	x	x	x	
property						x

Security Groups/Roles	G addActions	G editActions
no selection or root	x	
any node		x

Security Users	G addActions	G editActions
/admin or /system	x	
any node		x

Assets	S folderActions	S assetsActions	A createVariant
no selection or root		x	
mgnl:folder	x		
any other node		x	only if node is master

Pages	A delete	A preview	A edit	A export	A activate	A deactivate	A activateRecursive
no selection or root							
any selection	x	x	x	x	x	x	only if no sub pages

Appendix 2 - Previous notes

Problem

The action bar can take several states within one sub-app.

- In page editor, some actions are displayed or hidden according to the type of selection (page, area, optional area, component)
- This is currently handled in a dirty way:
    - The action bar provides an API for showing/hiding a whole section
    - Therefore the page editor sub-app defines 7 different sections, which mostly hold similar groups, and sometimes even similar (i.e. duplicated) action definitions.
- Sometimes the **context sensitivity** is only represented by enabling/disabling one or several actions
    - Each sub-app has to reimplement the itemSelected listener to control proper enablement of actions
    - Basic conditional enablement is not ensured, nor is it configurable
    - One cannot easily control which actions are available (visible/enabled) based on parameters out of configuration (roles, instance)
- Conditional enablement of actions has already caused many issues before alpha1

Proposal

- Action bar definition should define two things:
    - the **structure**: sections, groups, and all possible actions in these groups, in one single place. This ensures proper ordering of actions at all time
    - the **states** that describe e.g. the section titles, the set of actions displayed, and potential substates. This mechanism should allow for additive composition of states (think of the use case for an optional AND editable area in page editor)
- Implementing context sensitivity then only means to configure these states properly and toggle between them.
- ⚠ It is likely that context sensitivity is not handled at all at the action bar level, but rather straight on the actions.
    - If so, the action bar definition is reduced to its sole structure.
    - Edge case (page editor): do we provide a way to configure context-sensitive section titles?

Decision

-