Struts Dialogs: CRUDAction

1. Overview

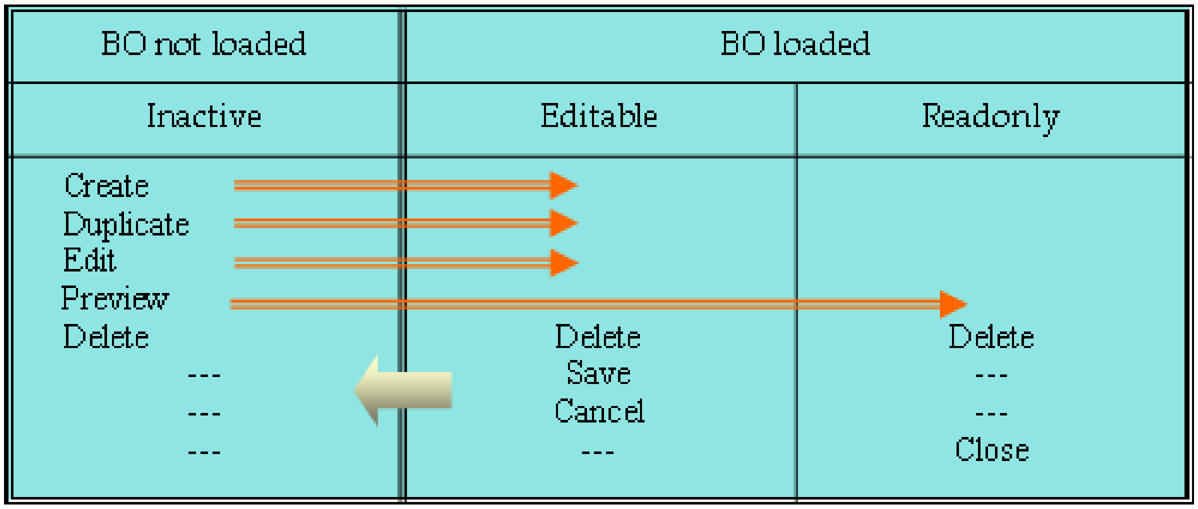
The purpose of `CRUDAction` is creating web controls for the common use case of an interactive application: **cr**eation, **u**pdate and **d**eletion of a *business object* (BO).

`CRUDAction` inherits from DialogAction and adds the following features:

- handler methods for CRUD events;
- states and state transitions for BO;
- mappings for result pages;

2. Events

The following table displays possible states of BO, and events, permissible for each state. A very common implementation is browsing a list of BOs (also called items), then selecting one item for preview or for editing. Viewing and editing is usually performed in a separate form. The same or similar form is usually used to create a new item.



CrudAction states

Let us call an existing item, which is being viewed or edited, or a new item, which just has been created and is being filled out, a *current item*. In other words, current item is the item we currently work with.

If current item does not exist, `CRUDAction` is set to `inactive` state. It is possible to switch to one of the active states, by doing one of the following:

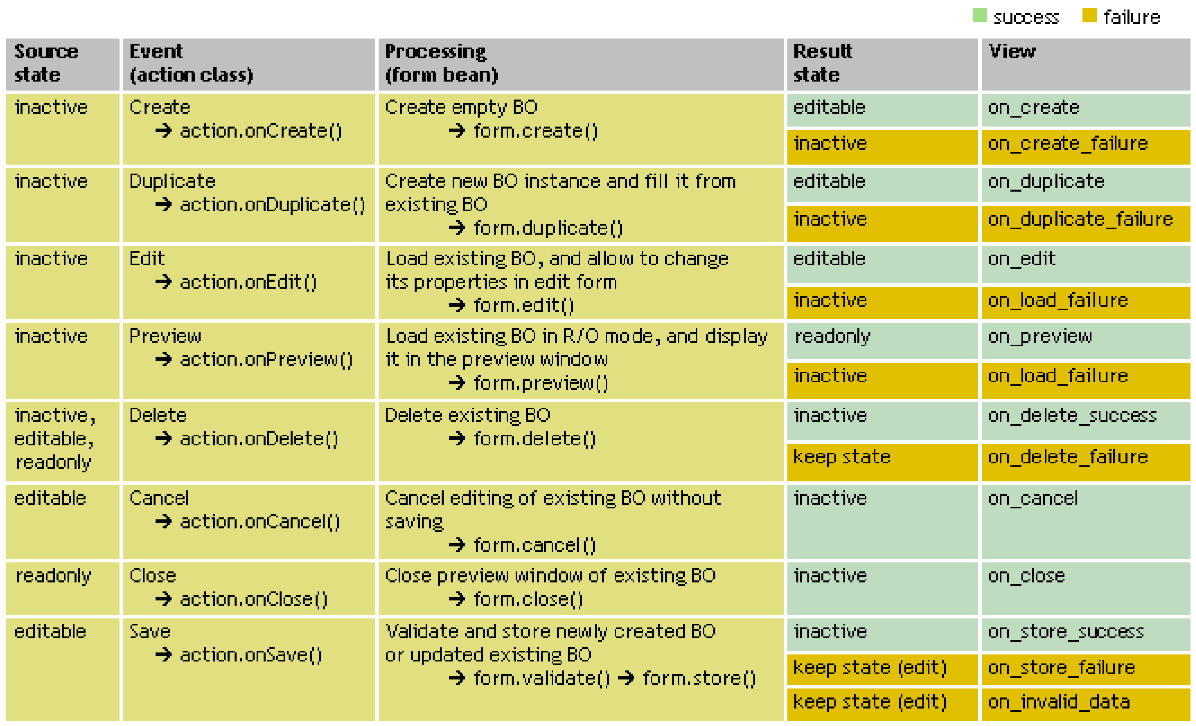
- Create new item from scratch and start populating its properties.
- Create new item duplicating its data from existing one, and change some of its properties.
- Edit existing item, updating some or all of its properties.
- View existing item in read-only mode.
- Delete existing item.

After we created a new item or started to edit an existing one, CRUDAction switches to one of the active states: `editable` or `readonly`. `Editable` state permits to update item data. `Readonly` state is used for preview mode and may help to optimize item caching.

After changes are stored or canceled, current item is deallocated, and `CRUDAction` is switched back to `inactive` state. Any further attempt to view or edit non-existing item will cause error.

3. States and transitions

The table below describes `CRUDAction` states, operations permissible in each state, and outcomes of each operation. Events are processed in action class by handler methods. These methods perform functions common to every event, like handling error messages and deciding which mapping to return. The actual data processing is done in the form bean.



🟩 success 🟨 failure

Source state	Event (action class)	Processing (form bean)	Result state	View
inactive	Create → action.onCreate()	Create empty BO → form.create()	editable	on_create
			inactive	on_create_failure
inactive	Duplicate → action.onDuplicate()	Create new BO instance and fill it from existing BO → form.duplicate()	editable	on_duplicate
			inactive	on_duplicate_failure
inactive	Edit → action.onEdit()	Load existing BO, and allow to change its properties in edit form → form.edit()	editable	on_edit
			inactive	on_load_failure
inactive	Preview → action.onPreview()	Load existing BO in R/O mode, and display it in the preview window → form.preview()	readonly	on_preview
			inactive	on_load_failure
inactive, editable, readonly	Delete → action.onDelete()	Delete existing BO → form.delete()	inactive	on_delete_success
			keep state	on_delete_failure
editable	Cancel → action.onCancel()	Cancel editing of existing BO without saving → form.cancel()	inactive	on_cancel
readonly	Close → action.onClose()	Close preview window of existing BO → form.close()	inactive	on_close
editable	Save → action.onSave()	Validate and store newly created BO or updated existing BO → form.validate() → form.store()	inactive	on_store_success
			keep state (edit)	on_store_failure
			keep state (edit)	on_invalid_data

CrudAction transitions

Why business object is handled in the form bean? Should not all buseness operations be defined in an action class? Not really, because these methods logically belong to form bean.

- `CRUDAction` uses form bean to store both input and output data. This data is either updated from request, or is used to render output page. Online poll shows that about 60% of Struts developers use form beans for both input and output, so `CRUDAction` does not invent the wheel.
- `CRUDAction` implements two-phase I/O processing, where each data submission is split in two requests. Thus, a form bean must store intermediate data between requests.

Because of above, the purpose of form bean is elevated from simple transfer object populated by request data, to stateful interactive object. This object can be manipulated with methods, which are conveniently defined right in it. Each session receives its own copy of a form bean, and it is logical to process business events right where business data is stored.

A form bean must implement `ICRUDForm` interface. Event handlers in the `CRUDAction` correspond to business methods in `ICRUDForm` one to one. "Save" event is an exception, it consists of two steps: validate item data, and then persist item. Standard `validate` method of a form bean is used for validation.

In the same manner as `validate` method, all business methods of `ICRUDForm` interface return `ActionMessages` object. Non-empty object signifies an error. Upon completion, whether with errors or not, business methods update the state of BO and return to event handler. Then event handler decides where to go next.

4. Error Messages

`CRUDAction` employs two-phase I/O processing, so error messages are temporarily saved between requests by event handlers. Messages are removed from session automatically.

5. Business data

Because the form bean must implement `ICRUDForm` interface, it is unlikely that you will use dynabeans. You have to define a good old form bean extending `ActionForm` class. Recall that intermediate data should be stored in the form bean. This allows to populate it automatically from the request during input phase. To make this work, there are two main choices.

First approach is to define getters and setters for all properties of your business object right in the form bean. The example provided below uses this approach. Creating setters and getters is a tedious process, but it has its benefits.

- You can define all form bean properties as strings. This helps to retain entered values, even if they are incorrect, because no conversion to native type is made.
- You can use `validate` method provided by a form bean to check input data, and you do not need to worry about consistency of a real business object.

Another choice is to set up a BO as a property of the form bean. Struts will access it as nested property and will be able to set and read its internal fields. You will need to use qualifiers to access properties of a BO from JSP page. This choice makes sense in the following situation:

- Your business object accepts string values, and has built-in validation mechanism.
- The validation mechanism returns error messages, which can be converted to Struts `ActionMessages`.

6. Input and Output

`CRUDAction` uses the same two-phase processing model, as `DialogAction` which it inherits from.

Handler methods process input data, sent to the action in the first (POST) phase. They perform update of business object, update its state, generate error messages if needed, and then return to Struts with "reload mapping". The mappings are shown in the transition table in the two right columnts. They identify an action class, which would build and display a page. For all but very view applications all these mappings point back to the same action, which process CRUD events. See the sample source code.

Second phase (GET) is serviced by `getView` method. This method can be overriden, but in its standard form it should suffice for most applications. It does not do much except reading current state of business object:

- If item is `editable`, it forwards to edit page.
- If item is `readonly`, it forwards to preview page.
- If item is `inactive`, that is, it does not exist, then `getView` method forwards to error page, which should notify a user that item has expired. This may happen when a user leaves edit or preview form, and then clicks Back button. `CRUDAction` tries to prevent a user from viewing stale data.

The pages themselves should be designed by a developer of specific application. Edit page usually contains input fields and two buttons: "Save" and "Cancel". Preview page usually displays business data as text, and has "Close" button. Of course, these buttons should generate events which are defined in the transition table. You do not need to implement all handler methods or events. On the other hand, you can add new events or outcome mappings.