

Struts Dialogs

1. Overview

Struts Dialogs is a library for Struts 1.2.x framework, which improves development process and makes applications more robust and user-friendly. Struts Dialogs combines best features of Struts Front Controller pattern with Page Controller features of ASP.NET. It implements event handling, basic state management, provides simplified control flow and facilitates component development.

- **Simplified control flow** - cleaner separation of concerns between actions, action forms and JSP pages.
- **Event handling** - uniform processing of command links and form submission events.
- **State management** - using session-scoped form bean as first-class stateful input/output object.
- **Easier configuration** - improved request/response cycle is controlled with less XML markup and fewer Java classes.
- **Web Wizards** - controlled flow of web pages for a given web resource, similar to traditional desktop wizard dialogs.

2. Front Controller pattern

Struts is a controller framework that adheres closely to the principles of Front Controller Pattern ([J2EE Front Controller pattern](#), [.NET Front Controller pattern](#)):

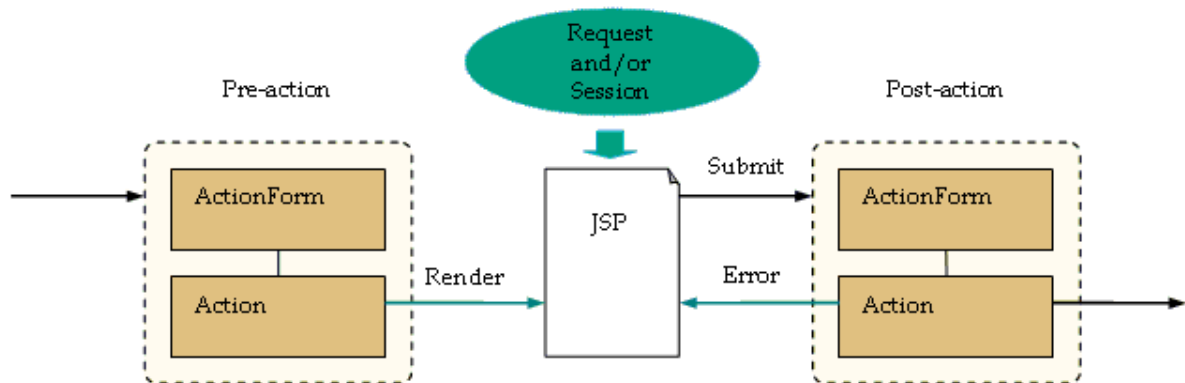
- **ActionServlet, RequestProcessor**: Controller (J2EE); Handler (.NET)
- **Action**: Dispatcher (J2EE); part of Command (.NET)
- **ActionForm**: part of Command (.NET)
- **JSP page**: view

The Front Controller pattern does not specify the exact details of how the state should be managed, or how a response should be handled after a view is rendered.

3. Struts: traditional request/response cycle

Class `ActionForm` was initially designed as convenience object for input data. Struts guidelines recommend using `ActionForm` in request scope. This precludes from storing state information in `ActionForm`. It is up to developer to decide where to queue output data to, and where to store information between requests.

Struts users came up with idea of pre-action (output action, setup action) and post-action (input action) dispatchers, so actions and JSP pages are interlaced, while action-relevant business data (green oval on the picture below) is located outside of an action:



Pre- and Post- actions

Each interactive JSP page is handled by at least two actions, and each action can render different pages. Because of many-to-many relationships between actions and JSP pages, web application becomes hard to maintain right from the start.

4. Struts Dialogs: code-behind and event handling

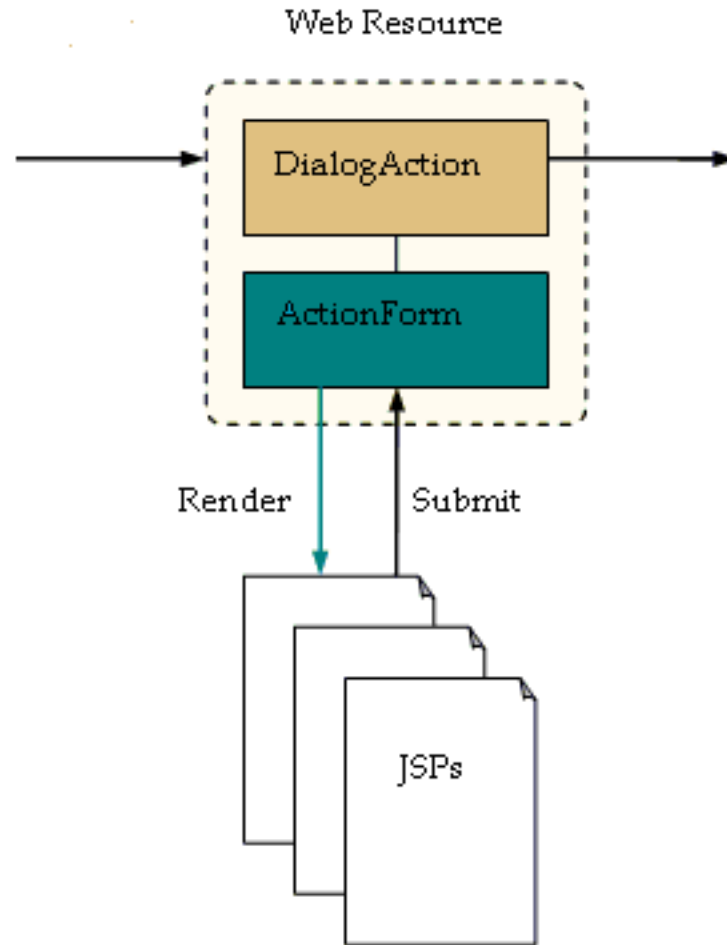
Struts Dialogs makes development simpler by employing code-behind pattern similar to one used in ASP.NET framework, while keeping your investments into Struts.

One of the ASP.NET concepts is the unity of page markup (ASPX) and business-related code (C# or VB). A page layout and widgets are defined in the markup, while page lifecycle and incoming events are handled by corresponding class file. This concept is especially easy to grasp for programmers of desktop applications:

Before page is displayed, the code behind it initializes page data.
 After page is rendered, activating a widget in the browser window generates event, which is dispatched to a handler. defined in the class.

This pattern is implemented by Struts Dialogs with no changes to core Struts classes or tag libraries. Moreover, it is improved, because Struts allows to define several markup pages corresponding to one Java class.

To use event-dispatching actions one should think in terms of *web resources*. Internet is about web resources, not about mere pages. A page is just a visualization of a resource in its current state at a given time. An address identifies a resource, not a particular representation of it. A resource can be rendered differently depending on its state.



Dialog action

In Struts Dialogs each resource is represented by one action (which consists of an action class and a form bean). Depending on resource state, an action can render one view or another. Views, corresponding to an action, are defined as JSP pages. Each JSP page belongs to one action only, which is called a *parent action*.

Submission of a form from JSP page generates input event, which is handled by the page's parent action. Action class defines handler methods for every input event. Input events can be generated not only by submitting a form (POST), but also by clicking on a command link (GET). Think of enhanced `DispatchAction`.

Direct linking from JSP page to another resource is possible, but discouraged. Instead, a command link should generate an event, which would be handled by parent action class. It is up to an action class to decide where to navigate next. This approach allows to define all navigation targets in the `struts-config.xml` file, having a clear representation of web application structure.

5. State management

Struts Dialogs does not introduce new classes to manage application state. Instead, it uses existing `ActionForm` class as a first-class stateful input/output object. In JSF terms, `ActionForm` now acts as a backing bean for JSP page. There is nothing groundbreaking in using session scope for `ActionForm`, or in storing output data in it. Online poll shows that about 60% of respondents use `ActionForm` for queueing output data.

With changing `ActionForm` scope to session, it is possible to initialize form bean only once, and to reuse data between requests. This is convenient for form resubmissions, for page reloading or for navigating back to previous resource. Having all resource data in a single `ActionForm` simplifies JSP page. Struts automatically populates `ActionForm` with submitted data on input phase, the same data can be used for presentation on render phase without additional efforts.

Session scope justifies the usage of nested properties within `ActionForm`. It is easy and convenient to use business objects or DTOs as nested properties instead of copying their data to `ActionForm` and from `ActionForm`.

6. Actions

Most features of the Struts Dialogs library are available through use of different action classes. Most often you will be using `DialogAction` class, extending your specific action class from it.

6.1. `SelectAction`: dispatches submit events

[SelectAction](#) is an enhancement of standard `DispatchAction`. It handles submit events and provides improved dispatching functionality. You can use this action class if you do not need full power of `DialogAction`, and all you want is just to dispatch submit events to handler methods. `SelectAction` has the following features:

- works uniformly with pushbuttons, image buttons and regular links;
- allows to set arbitrary button caption and to change it at runtime.

6.2. `DialogAction`: Jack of all trades

Struts Dialogs

[DialogAction](#) is the main asset of Struts Dialogs library.

DialogAction allows creating robust web components and web controls, which exhibit friendly user experience, react properly to Refresh, Back and Forward buttons, and do not cause implicit double submits or annoying POSTDATA messages.

DialogAction adds the following capabilities to event dispatching features of SelectAction:

- processing of initialization event;
- handling of error messages;
- view rendering;
- state handling (via corresponding action form);
- two-phase input processing (using POST-redirect-GET pattern)

6.3. WizardAction: creates robust page flows

[WizardAction](#) allows to create *web wizards*, similar to traditional desktop wizard dialogs. A wizard has predefined sequence of states, and is rendered with HTML forms, containing Back, Forward, Cancel and Done pushbuttons.

6.4. CRUDAction: simplifies CrUD operations

[CRUDAction](#) implements all operations needed to manipulate business data, nested business object (BO), or nested value object (VO) also called an `item`. This action allows to create new item, duplicate existing item, edit, view, clear and delete item.

A very common use case is browsing a list of items, then selecting one item and performing different operations on it. CRUDAction can handle this use case with grace. It is possible to implement handling of both item list and CRUD operations as one web component.

7. Mail Reader demo application

The Struts Dialogs package now includes a version of "MailReader Demonstration Application", originally bundled with core Struts distribution.

MailReader is rewritten using component technology of Struts Dialogs to show the possibility and benefits of component approach for a Struts application. See the [Mail Reader Walking Tour](#) for details.

See [MailReader live demo](#) to compare Struts Dialogs implementation with original Struts version.

8. Live Demos

Each action class from Struts Dialogs library is illustrated with sample code and [live demos](#).

9. Download

[Download Struts Dialogs](#).