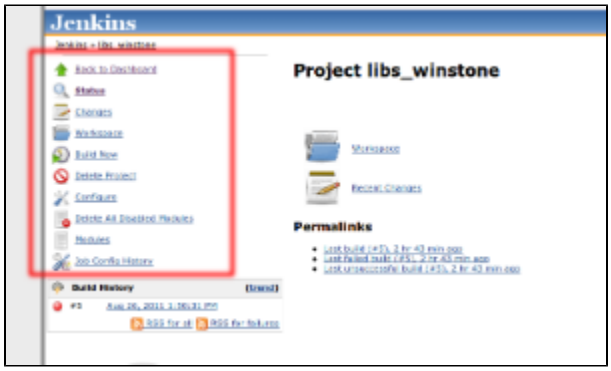
Action and its family of subtypes

Action is one of the primary ways that plugins can use to enrich the build/project pages. This extension point has the longest history, and as such its meaning is somewhat confusing, and its use across places are somewhat inconsistent. This page attempts to clarify that.

What is Action?

The recurring UI pattern in Jenkins is to have a bunch of navigable links and command links to the left of the page. The extension point Action was born as a means to contribute custom entries here.



Three methods defined on Action are for obtaining information needed to render actions (icon, name of the link, and where it takes the user to.)

Action and URL space

Actions often want to define additional URL space into Jenkins to show information. For example, "test reports" action adds a whole bunch of subpages into Jenkins. To facilitate this common use case, Jenkins uses the `Action.getUrlName()` method.

For example, if the host object of `Action` (typically extends from Actionable) is at the URL `/jenkins/foo/bar/`, then `Action` that returns `zot` from the `getUrlName()` would be mapped to `/jenkins/foo/bar/zot` (and thus entire sub-space is available for this action to control. This is how you can have `Action` that shows information.

In some other cases, actions just want to link to existing URL (perhaps even outside Jenkins.) `getUrlName()` javadoc explains how to do it.

Special views for the action host

Some host classes that host `Action` designate a specific view, and when such a view is defined, it gets pulled into the host object's page rendering. For example, AbstractBuild allows actions to define `summary.groovy` (or other languages), and when it is present, those views are merged into the top page of the build. For example, JUnit integration uses this to make the test report prominent.

See the javadoc of respective host classes for details. Similarly, some host classes define a marker interface that extends from `Action` in addition to define additional interaction with actions. See ProminentProjectAction for an example. See the subypes of Action for all such marker interfaces.

Transient and Persistent actions

Depending on the host, `Action` have different persistence semantics. In some hosts, such as AbstractBuild, actions are explicitly added (typically by Build Steps that executed) and persisted along with it. In some other hosts, such as View, actions are assembled entirely from other configurations of the said object whenever it's changed and therefore not persisted. Still others have hybrid approach, such as AbstractProject, which supports some explicitly added actions as well as transient ones contributed from build steps and etc.

Unfortunately, the only way to really figure out which host behaves how is to look at the implementation of the getActions() method.

Transient actions can be discarded any moment, and new ones created to take the place of the old one. Therefore, it is not suitable to use it for synchronization, or for keeping other persisted records.

Persistent actions survive forever, until it's programmatically removed or the host object gets removed itself. So care must be taken to use 'transient' where appropriate. Those objects can be also used for synchronization.

Adding actions

How you insert your implementation of `Action` to Jenkins depends on the type of the host, and whether it's transient or persistent.

Persistent actions are almost always added via `Actionable.addAction()`.

Transient actions are added via callback. You implement other extension points that can contribute transient actions, such as LabelAtomProperty, Builder, and Publisher, then your host will call your methods and expect you to return a list of transient actions. The host may do this every so often when it thinks it needs to update its transient actions.

More recently, we are introducing a series of `Transient***ActionFactory`, such as TransientViewActionFactory, so that transient actions can be contributed without implementing another stateful extension point like Builder. Most generally, TransientActionFactory can be used for *any* Actionable.

RootAction

Most actions are registered in one of the two ways described above, but there's a notable exception, which is RootAction. This is a stand-alone extension point of its own, that you'd put @Extension, and these gets automatically registered as transient actions of the root Jenkins object. There's a variant of this called UnprotectedRootAction that is made accessible even to anonymous users without the read access to Jenkins.

Actions as inputs to build

Actions are also used as a parameter/input (in a broad sense of the word) to the build, as can be seen in `Queue.schedule(...)`, because these actions get added to the newly created AbstractBuild and then it can be accessed from build steps. For example, ParametersAction is one such use of `Action` that's used to implement parameterized builds. Matrix Reloaded Plugin uses this mechanism similarly to remember what subset of the matrix build to execute.

There are several subtypes that allows this kind of actions to interact with the queue, such as FoldableAction and QueueAction.

Actions as the vehicle to keep information about builds

AbstractBuild is Actionable and it takes persisted actions. This is the primary means for plugins to attach any information to the build. This allows other plugins to use your action to understand what has happened (such `Action` are also used to render information, the primary purpose of the `Action` class.)

Even if your plugin doesn't use such record by itself, we highly recommend you do it, because that opens up interesting possibilities to build on top of your work.

Invisible actions

As can be seen in the section above, some use of actions emerge over the time that goes beyond the original purpose of extending the UI, and therefore it became convenient to be able to have actions that aren't visible in the UI at all. InvisibleAction is the convenient base class for such purposes. For example, InterruptedBuildAction records how the build was aborted, but this is currently invisible, so it extends from InvisibleAction

Change the way Action link is rendered

An Action object can have an optional `action.groovy`/`action.jelly` file. When this file is presented, that is used to render the action link instead of the default icon+text.

Use this with caution as it can create inconsistency in the user experience.