Actions Support User Guide

Release 2.0.4: Early Preview Edition

Table of Contents

Introduction

The ActionsSupport NBM is a set of classes which can be used by other NetBeans plugins, to allow the creation of Actions, with the ability to run external programs.

This document is designed for developers of NetBeans Modules who wish to integrate these features into their code.

These Action can execute:

- NetBeans methods
- Methods coded within the plugin
- External programs which can be executed in a CLI-like manner, allowing the use of such programs from the NetBeans UI

DynamicActions and DynamicAsyncActions

A Dynamic Action is an Action which can be enabled or disabled and is not displayed when disabled.

These are two classes of Dynamic Actions:

- The base class (DynamicAction) which mirrors the Action class.
- The DynamicAsyncAction class which mirrors the DynamicAction class, running the onClick method in a seperate thread, so being suitable for any action which takes a period of time (which would otherwise block the processing of UI events).

DynamicActions can be associated with a particular node, either being for:

- All projects of a particular type
- A specific project, as defined by a properties file using DynamicAyncActions.

NodeActions

NodeActions supports the creation of Actions for a node, by enabling the creation of DynamicCLIActions using a properties file.

NodeActions observes the node folder containing the property file, ensuring the actions are updated whenever changes occur to the properties file.

Additional files within the node folder can be observed, so that changes can trigger updates to any associated objects.

NodeActions provides a method for assembling node actions, combining various sources of actions to create the final dynamic actions array required by a node definition.

ActionsSupport in action

NodeActions example

NodeActions creation:

- accepts the filename of a properties file
- enables the creation of DynamicAyncActions from the information in a properties file
- monitors the properties file for changes and refreshes the actions as necessary
- enables support for assembling a node’s actions

In the node constructor, add:

```
XXXXProject(FileObject dir, ProjectState state) {
    .....
    dynamicactions = new NodeActions(dir, "projectactions"); ①
}
```

① create a DynamicActions object, which will observe a file *projectactions.properties* within folder *dir* for changes. It will use the content of that file to define DynamicAsyncAction objects, updating these actions whenever the file changes.

In order to be able to create the node’s actions, NodeActions must be made aware of all other actions required. It will create the list containing three sections, with separators. Any of the sections may be empty.

The sections are:

- Basic node actions, typically the standard NetBeans actions which a node would want to include
- Node actions, typically DynamicAsyncActions which are node type specific (ie not defined in the node’s properties file)
- DynamicAsyncActions which are created using the node’s property file

The third group are defined by the NodeActions constructor, while the others must be passed to NodeActions (using setNodeBasicActions and setNodeActions methods).

```
dynamicactions.setNodeBasicActions(
                CommonProjectActions.renameProjectAction(),
                CommonProjectActions.copyProjectAction(),
                CommonProjectActions.closeProjectAction() ①
        );
dynamicactions.setNodeActions(
                new DynamicAsyncAction("Bake")
                        .onAction(
                            () -> new CLIExec(projectDir,"jbake -b")
                                    .stderrToOutputWindow()
                                    .stdoutToOutputWinow()
                                    .ioTabName("Bake " + projectDir.getName())
                                    .execute("Baking")) ②
                );
```

① define 3 actions, in this case standard Netbeans Project Actions.

② define 1 DynamicAsyncAction, which will appear in all projects of this type.

The Node should use the NodeActions getAllNodeActions method to obtain it's Action array. This will combine all valid actions which have been made known to NodeActions.

```
public Action[] getActions(boolean arg0) {
    return dynamicactions.getAllNodeActions(); ①
}
```

① Call NodeActions getAllNodeActions method to get a merged list of all actions.

Use of the DynamicAction/DynamicAsyncAction classes

While the majority of uses of this module will be via the NodeActions API, it is possible to use the DynamicAction classes independently.

DynamicAction examples

Add a DynamicAction to a node's actions.

```
@Override
public Action[] getActions(boolean arg0) {
    return  new Action[]{
                ...
                null,
                new DynamicAction("do something")
                    .onAction(()->do_something()) ① ②
            });
}
```

① Create the DynamicAction

② ...and add it to the nodes's actions. The action is enabled so needs no further initialisation.

DynamicAsyncAction example

Add a DynamicAsyncAction to a node's actions.

```
@Override
public Action[] getActions(boolean arg0) {
    return  new Action[]{
                ...
                null,
                new DynamicAsyncAction("do something")
                    .onAction(()->do_something()) ① ②
            });
}
```

① Create the DynamicAsyncAction

② ...and add it to the nodes's actions. The action is enabled so needs no further initialisation.

Utilising the ActionsSupport NBM

The following provides a quick reference to key information needed to use the ActionsSupport NBM.

Using the Library in a Maven build

The NBM must be added as a dependency in your POM. To ensure Maven can find it in the GitHub Maven repository, you will have add the following in your POM:

```
<repositories>
    <repository>
        <id>github</id>
        <url>https://maven.pkg.github.com/The-Retired-Programmer/NetBeansNBMs</url>
    </repository>
</repositories>
```

and then add a dependency in your POM

```
    <dependency>
        <groupId>uk.theretiredprogrammer</groupId>
        <artifactId>actionssupport</artifactId>
        <version>{revnumber}</version>
    </dependency>
```

Installing the NBM into Netbeans

To be able to use the Plugin it must be downloaded and then installed in Netbeans.

There is two ways to download the NBM:

- Releases of this product are stored in a Maven repository on GitHub. It is possible to download an NBM from the Package available .
- However assuming you have added the dependency to you POM and have compiled your code, a copy of the NBM will have been down loaded to your local Maven repository (by default ~/.m2).

Once you have a downloaded copy on your machine, you follow the standard Netbeans process to install the downloaded file (Tools>Plugins)

Links

- Releases are stored in a Packages Maven repository on GitHub
- Source Code is stored in a Git repository on GitHub as part of a monorepro containing NBMs.

- Documentation (included JavaDoc API documentation) can be found on the TRP website.
- Documentation source is stored in a Git repository on GitHub as part of a monorepro containing documentation.
- Development Process Guide for The-Retired-Programmer products can be found on the TRP website.