**Actions**

The three most important terms in *Madvoc* are:

* **action path** - the HTTP request path or the URL;
* **action** - handler method, mapped to action path, processes requests; and
* **result** - handler for whatever action method returns, provides a response (using e.g. JSP). Often results threat returned object as a *result path*.

These three makes the whole cycle of handling request and providing response. Most of the configuration in *Madvoc* is done by convention, so always keep in mind this three dimensions of request processing.

**Action path**

**Action** is action *method* defined in action class, mapped to some URL - the **action path**. *Madvoc* uses naming convention (CoC) and annotations to define action path from action method. By default, action path is built from package, class and method name of an action or its annotations, using the following convention:

action path = /<action\_package>/<action\_class>.<action\_method>.<extension>

* extension - extension value (default: "html"), defined by *Madvoc* configuration or specified in the annotation;
* action\_method - part of the action path that comes from action method, by default it is method name;
* action\_class - part of the action path that comes from action class, by default it is un-capitalized class name with the last word stripped;
* action\_package - optional part of the action path that comes from action's package.

By default, each part of action path is defined from method/class/package name so you don't have to explicitly specify anything. Still, each part of action path can be explicitly defined by corresponding annotation value, so you can easily override defaults.

**Action class & action method**

Bare minimum for creating an action is putting annotations on POJO class and one of its methods that is going to handle the action path request:

@MadvocAction

public class HelloAction {

@Action

public void world() {

}

}

This action class and action method is mapped to the following action path: /hello.world.html. As said, each part of action path may be set explicitly in annotations; the following action:

@MadvocAction("holla")

public class HelloAction {

@Action("mundo")

public void world() {

}

}

is mapped to /holla.mundo.html. Moreover, action paths are built by simple string concatenation, so it is possible to set more complex paths using annotations:

@MadvocAction("foo/boo")

public class HelloAction {

@Action("zoo/hello.exec")

public void world() {

}

}

This action is mapped to: /foo/boo.zoo/hello.exec.html.

One action class may contain more then one actions (action methods). This happens often, especially when you have set of similar requests over some same resource.

**Custom extension**

It sounds reasonable that most of the website's action paths end with the same extension. Therefore, the default extension is defined in global *Madvoc* configuration. However, it is still possible to set custom extension using @Action annotation's element extension:

@MadvocAction

public class HelloAction {

@Action(extension="jpg")

public void world() {

}

@Action(extension=Action.NONE)

public void foo() {

}

}

The first action is mapped to: /hello.world.jpg. The second one is mapped to: /hello.foo.

Action's extension is either default one or one defined by @Action element extension. There is no default convention in setting action's extension.

Since *Madvoc* is very extensible, it is also possible to extend its component dedicated for action path registration and to implement custom behavior. For example, it is easy to develop such feature where any method that starts with store... has extension .do instead of the default one.

**Full action path**

To override *Madvoc* action naming conventions without coding, just specify the full action path in the annotation by using the prefix '**/**':

@MadvocAction

public class HelloAction {

@Action("/bonjour-monde.html")

public void world() {

}

}

Obviously, this action is mapped to /bonjour-monde.html. When action path defined in @Action starts with '**/**' it is considered as **full** action path and all other names (class name, extension) are ignored.

When full action path is specified, *Madvoc* will **not** append default extension nor the custom one (defined by @Actions element extension).

**Action packages**

By default, packages are ignored and not used when building action paths. Nevertheless, it make sense to group several action paths (i.e. action classes) in one folder (i.e. package). *Madvoc* provides way how to include packages when building action paths.

First, this feature must be turned on by setting the root package, one that will be mapped to the web root. This can be set during *Madvoc* initialization. One way of doing this is:

public class MyWebApplication extends WebApplication {

@Override

protected void init(MadvocConfig madvocConfig, ServletContext servletContext) {

madvocConfig.getRootPackages().addRootPackageOf(IndexAction.class);

}

}

Root package is defined by a package name and a path that package is mapped to. For web root you may omit the path. Also, the package name can be defined by passing the action class, like in above example. Therefore, above configuration snippet defines one root package (package where IndexAction class belongs to) mapped to the web root (/).

When root packages are defined, *Madvoc* will use package name of action classes relative to the build action path. The package name is relative from the root package. The following action:

package org.jodd.madvoc.doc;

@MadvocAction

public class HelloAction {

@Action

public void world() {

}

}

is mapped to /doc/hello.world.html, if the root package is set to: org.jodd.madvoc.

You can specify more then one root package. Be careful not to overlap mapping paths!

Root packages may be defined by putting an empty class named MadvocRootPackage (name is configurable) that is annotated with @MadvocAction. This class serves just as an marker for root packages.

Finally, it is possible to specify custom action package using @MadvocAction annotation on package (in package-info.java).

It is also possible to override package annotation value with @MadvocAction annotation of action class: if its value starts with '**/**' then package value is ignored.

Sometimes developer wants to group some action classes in separate subpackage, but doesn't want to change the action path (e.g. the root path). By specifying @MadvocAction("/") on sub-packages *Madvoc* will map all containing classes to the web root, as they were there and not in the sub-package:

@MadvocAction("/")

package com.....;

import jodd.madvoc.meta.MadvocAction;

**HTTP request methods**

By default, *Madvoc* will ignore value of HTTP request method. No matter if it is POST, GET or other, mapped action method will be invoked. If needed, *Madvoc* offers more control considering HTTP methods: it allows to specify one for action method, using @Action annotation's element method:

@MadvocAction

public class FormAction {

@Action(method = "POST")

public void store() {

}

}

This action method is mapped to /form.store.html and will be invoked only for POST HTTP request methods. GET and others will simply return error 404 (page not found).

When HTTP method is specified, *Madvoc* will register such action path with appended HTTP method information. Action from above example is therefore mapped to: /form.store.html#POST.

When looking up for the action path among registered once, *Madvoc* first tries to find action path with specified HTTP method. If such action path does not exist, *Madvoc* will lookup for action path with no HTTP method information.

Similar as for extensions, it is possible to extend *Madvoc* to programatically specify HTTP method to actions that match some custom criteria.

Nice practice is to specify the extension such as do using @Action annotation to all actions that are mapped to POST request (i.e. form submissions) and then to programatically set POST for those actions, and GET to all others; if it is not explicitly set different.

**Default action methods**

As seen, *Madvoc* uses action method name (or annotation value) for creating action path. Moreover, it is possible to have action path that doesn't include action method name - what is often needed for 'common' pages (such as index.html, about.html, error.html). By default, *Madvoc* will ignore action method name for methods named as execute and view. Such action methods are so-called the **default** ones. So, the following action:

@MadvocAction

public class IndexAction {

@Action

public void view() {

}

}

is mapped simply to: /index.html. If more than one default method name is used, *Madvoc* will either take the last one, or will throw an exception indicating duplicated action paths (depending on configuration). Furthermore, default action names are part of global *Madvoc* configuration and can be customized as needed.

Alternatively, @Action annotation value element may be set to NONE value. Then the method name will be ignored when building action path. Therefore, following action has the same action path mapping:

@MadvocAction

public class IndexAction {

@Action(Action.NONE)

public void foo() {

}

}

**Action mapping cheat-sheet**

Following table summarize default behavior of ActionMethodParser - *Madvoc* component dedicated for building action paths from registered actions.

| **package** | **class** | **method** | **action path** |
| --- | --- | --- | --- |
| \* | \* | /foo | /foo |
| \* | \* | /foo.ext | /foo.ext |
| \* | /boo | foo | /boo.foo.html |
| \* | /boo | foo.ext | /boo.foo.ext.html |
| (none) | boo | foo | /boo.foo.html |
| (none) | boo | view/execute | /boo.html |
| /zoo | boo | foo | /zoo/boo.foo.html |