tapestry-routing guide

> ⚠ **Version status: 0.0.7 beta**
> 0.0.6 added support live reloading of routes and support for contributing routes from different sources.

Overview

Inspired by sitebricks, resteasy and rails routing, **tapestry-routing** allows you to provide your own custom mapping between Tapestry 5 pages and URLs.

Usage

First (as always), add the *tapestry-routing* dependency to your pom.xml

```
<dependency>
 <groupId>org.tynamo</groupId>
 <artifactId>tapestry-routing</artifactId>
 <version>0.0.7</version>
</dependency>
```

Then annotate your pages with the ***@Route*** annotations (***@At*** annotations are still supported and work the same way)

eg:

Let's say you have a page: pages.projects.Members which have 2 parameters in its activation context: *(Long projectId, Long memberId)* and you want the URL for that page to look like **/projects/1/members/1**

Just add the ***@Route*** annotation to you page, like this:

```
package ...pages.projects;
@Route(" /projects/{0}/members/{1}")
public class Members {
void onActivate(Long projectId, Long memberId)
```

That's it!
The *RouterDispatcher* will take care of recognizing incoming requests and dispatching the proper render request and the *RouterLinkTransformer* will do the rest of the work, it will transform every *Link* for a page render request formatting it according to your route rule.

> ⓘ The *RouterDispatcher* is after the *PageRender* in the chain so Tapestry pages that matches the incoming request will always take precedence over the route rule.

CRUD and REST-like URLs

Here is an example of how *tapestry-model* is using *tapestry-routing*:

path	page	used for
/recipe	`@At("/{0}") public class List`	display a list of all recipies

/recipe/new	`@At("/{0}/new")` `public class Add`	return an HTML form for creating a new recipe
/recipe/*{id}*	`@At("/{0}/{1}")` `public class Show`	display a specific recipe
/recipe/*{id}*/edit	`@At("/{0}/{1}/edit")` `public class Edit`	return an HTML form for editing a recipe

Index Pages

You can't have pages named *Index

The only caveat with the current implementation is that you can't use Index pages. I mean pages named "*Index"
The way Tapestry handles *Index pages prevents the module from working properly.
My workaround (for now) is:

```
@Route("/") public class Home
```

If this "no Index pages" restriction is annoying you and you feel adventurous enough you can try this little module https://gist.github.com/3360101 that allows you to have Index pages as long as they have always an empty activation context.

Contributing Routes

If you like to have all your routes configuration centralized, you don't need to use the ***@Route*** annotation if you don't want to. You can contribute the routes to the *RouteProvider.*

```
@Primary @Contribute(RouteProvider.class)
public static void addRoutes(OrderedConfiguration<Route> configuration, RouteFactory
routeFactory) {
 String canonicalized = "subpackage/UnannotatedPage";
 configuration.add(canonicalized.toLowerCase(),
routeFactory.create("/not/annotated/{0}", canonicalized));
}
```

Avoid scanning ALL the pages.

If you want to prevent tapestry-routing from scanning all the pages packages looking for the ***@Route*** annotation set the ***DISABLE_AUTODISCOVERY*** symbol to "***true***". If you do this then you can either contribute your routes directly to the *RouteProvider,* or tell the *AnnotatedPagesManager* explicitly which pages do you want to be scanned.

```
@Contribute(SymbolProvider.class)
@ApplicationDefaults
public static void provideApplicationDefaults(MappedConfiguration<String, Object>
configuration) {
 configuration.add(RoutingSymbols.DISABLE_AUTODISCOVERY, true);
}
@Contribute(AnnotatedPagesManager.class)
public static void annotatedPagesManager(Configuration<Class> configuration) {
 configuration.add(SimplePage.class);
}
```