Component Cheat Sheet

This is a summary of the more common annotations and methods you can add to Tapestry pages and component classes.

For an exhaustive list, see the annotations list.

Related Articles

- Component Parameters
- Component Libraries
- Component Reference
- Page And Component Classes FAQ
- Component Classes
- Templating and Markup FAQ
- Component Templates
- Component Cheat Sheet

Field Injection Annotations

Main articles: Component Classes, Injection, Annotations

@Inject

@Inject is the Swiss Army knife of annotations; it's designed to connect your component to services, resources, and other objects. See Injection.

Service Injection

In most cases, the injected value is a service; the service is located by type. If there are ambiguities, caused by multiple services implementing the same interface, you'll see injection exceptions. You can resolve those exceptions by adding marker annotations to select a specific service, or by adding @Service to specify the specific service ID you want.

> ⚠ Use of @Service is discouraged. If marker annotations are available, that is preferred.

@InjectComponent

Injects a component from this component's template into this component's class. Injecting a component is based on the component's ID, which should match the field name. However, the value attribute of the @InjectComponent annotation can be specified as well, this takes precedence over the field name.

It is common to inject a component in order to obtain its client-side ID (used when generating client-side JavaScript).

@InjectContainer

Injects the container of a component or, when used in a mixin, injects the component the mixin is attached to.

@InjectPage

Injects a page of the application. Normally, the page to inject is identified based on the field type. The value attribute can be specified, in which case the page to be injected is identified by name.

@Environmental

Injects an environmental object; such objects are request scoped but may be overridden at any time using the methods of the Environment service. Environmental objects are used to allow outer components to communicate with components they enclose.

Most often, @Environmental is used with type JavaScriptSupport, which is used to add JavaScript code and libraries to the rendered page.

Field Behavior Annotations

Main articles: Component Classes, Annotations

@PageActivationContext

This annotation is allowed on a *single* field; the value of the field will be included in URLs for the page as the page's activation context. This is an alternative to implementing event handler methods
for the activate and passivate events directly.

@Parameter

Marks the field as a component parameter. Attributes of the annotation allow the parameter to be marked as required or optional. If the parameter value will typically be a literal string (for example, the title parameter to a Layout component), you should add `defaultPrefix=BindingConstants.LITERAL` to the annotation so that users of the component won't have to use the "literal:" binding prefix with the parameter. See Component Parameters.

@Persist

Marks the field as a persistent value, one that maintains its value between requests. The default *strategy* is to simply store the value in the session (which is created as needed). Other strategies can be specified by name as the value attribute. See Persistent Page Data.

@Property

Directs Tapestry to automatically generate a getter and a setter for the field, converting it to a JavaBeans property than can be referenced from the template.

@SessionState

Marks the field as a Session State Object (SSO). SSOs store global data, and can be injected into any page or component. The SSOs are stored in the session, using a key based on the Java type. SSOs are usually created on demand, but the `create` attribute can turn this off. See Component Cheat Sheet

@SessionAttribute

In Tapestry 5.2 and later, marks the field as a Session Attribute. Like Session State Objects (SSO), a Session Attribute is stored in the session, however Session Attributes are stored by using a name you choose rather than based on the Java type. See Session Storage.

@ActivationRequestParameter

Fields with this annotation will be encoded into URLs as query parameters, in much the same way as data is encoded into the URL path. The query parameter name matches the field name, unless the value attribute is specified.

Method Annotations

Main articles: Component Classes, Annotations

@OnEvent

Marks a method as an event handler method. Such methods may have any visibility, and typically use package private visibility (that is, no visibility keyword at all). By default, the method will handle the action event from any component; the value attribute controls the matched event, and the component annotation is used to limit the event source.

An event handler method may take parameters, corresponding the event context associated with the event, such as the page activation context for the activate event. The method will not be invoked if it defines more parameters than there are values in the context.

The @RequestParameter annotation can be used on parameters, in which case the parameters value comes from a request query parameter, and not from the event context.

Events fired on a component bubble up the component's container. Return a non-null value to cancel event bubbling. What values may be returned from an event handler method is dependent on the type of event.

You may also return true to indicate that the event is handled and bubbling should cancel (even for events that do not permit a return value).

An alternative to @OnEvent is the naming convention on*EventName* or on*EventName*`From`*ComponentId*.

@Log

Marks the method to be logged for debugging purposes: method entry (with parameters) and exit (with return value) will be logged at debug level, as will any thrown exception. This is primarily for debugging purposes. The Logger name will match the component classes' fully qualified class name.

@CommitAfter

The support for this annotation comes from the tapestry-hibernate module or tapestry-jpa module.

@Cached

Used on methods that perform expensive operations, such as database queries. The first time such a method is invoked, the return value is cached. Future invocations of the same method return the cached value.

The result cache is per-request and is discarded at the end of the request.

@Cached only works on methods that take no parameters.

Parameter Annotations

Main article: Component Parameters

@RequestParameter

Used with event handler methods to get the value for the parameter from a request query parameter.

Type Annotations

@Events

Lists the names of events that may be fired from within this component; used for documentation purposes only.

@Import

Allows JavaScript libraries and CSS stylesheet files to be included in the rendered page. Each such file is added to the page only once, in the order in which the page renders.

It is allowed to use symbol expansions (with the `${...}` syntax) inside a library or stylesheet path.

@Import may also be applied to individual methods, in which case the import operation only occurs when the method is invoked.

When specifying a file to import, you'll often use the prefix `context:` to indicate that the file is stored in the web application context, and not on the classpath. Relative paths will be on the classpath, relative to the Java class.

@SupportsInformalParameters

Marks the component as allowing informal parameters (extra attributes in the template that do not match formally declared parameters). Normally, informal parameters are simply discarded.

The method ComponentResources.renderInformalParameters() can be used to include the informal parameters within the element rendered by your component.

@Secure

Main Article: Security

Marks the page as accessible only via secure (HTTPs). Any attempt to access the page via standard HTTP will be redirected to the HTTPs version.

By default, the @Secure annotation is ignored in development mode and only active in production mode.

Render Phase Methods

Main article: Component Rendering

Render phase methods are close cousins to event handler methods; they are how Tapestry integrates your code into the overall rendering of the page. For each render phase, there's an annotation and corresponding naming convention to define a render phase method:

Annotation	Method Name	General Use
@SetupRender	setupRender()	Initializes the component before rendering
@BeginRender	beginRender()	Renders the element and primary attributes of the component
@AfterRender	afterRender()	Closes the element started in beginRender()
@CleanupRender	cleanupRender()	Performs cleanup after all rendering of the component finishes

Render phase methods either take no parameters, or take a single parameter of type MarkupWriter.

Render phase methods may return `void`, a `boolean`, or a *renderable object*.

⚠ Generally, a `renderable object` is a Block or a component. The object is pushed onto the stack of rendering operations, temporarily replacing the current component as the object to be rendered.

Returning true is the same as returning void; it means that the component should follow the typical flow:

- @SetupRender
- @BeginRender
- Render the component's template, if any
- Render the component's body
- @AfterRender
- @CleanupRender

If a component has a template, the component's body will only render if the template contains a <t:body> element. If a component has no template, then it will always render its body (between @BeginRender and @AfterRender).

A render phase method may also return false, in which case the flow continues to an alternate render phase, as per the chart in the Component Cheat Sheet reference page.

The most common cases:

- return `false` from @BeginRender to skip the rendering of the component's template and/or body, and continue with @AfterRender
- return `false` from @AfterRender to return to @BeginRender (this is used in component, such as `Loop`, that render themselves multiple times)

Page Life Cycle Methods

Main article: Page Life Cycle

Pages have a life cycle and this is represented by a *third* set of annotations or method naming conventions. Life cycle methods may appear on a page or any component of a page.

Annotation	Method Name	Description
@PageLoaded	pageLoaded()	The page instance has been loaded but not yet attached for the first time.
@PageAttached	pageAttached()	The page is being used within a particular request. This occurs before the activate event.
@PageReset	pageReset()	See notes below.
@PageDetached	pageDetached()	End of request notification.

Page life cycle methods may be any visibility. They must take no parameters and return void.

Page life cycle methods are of lower importance starting in Tapestry 5.2, since page instances are now shared across threads, rather than pooled.

The @PageReset life cycle is new in Tapestry 5.2. It will be invoked on a page render request when linked to from some other page of the application. This is to allow the page to reset its state, if any, when a user returns to the page from some other part of the application.

Configuring Annotations

The SymbolProvider service has two interfaces : FactoryDefaults and ApplicationDefaults. Tapestry provides 2 annotations in order to define which implementation you want to override in your AppModule :

- @FactoryDefaults

  AppModule.java (partial) with @FactoryDefaults

  ```
  @Contribute(SymbolProvider.class)
  @FactoryDefaults
  public void setParam(MappedConfiguration< String, String> configuration){
    configuration.add(SymbolConstants.PRODUCTION_MODE, "false");
  }
  ```

- @ApplicationDefaults

AppModule.java (partial) with @ApplicationDefaults

```
@Contribute(SymbolProvider.class)
@ApplicationDefaults
public void setParam(MappedConfiguration< String, String> configuration){
  configuration.add(SymbolConstants.PRODUCTION_MODE, "false");
}
```

Application Module Class Cheat Sheet | Cheat Sheets | Tapestry for JSF Users