DZone Refcardz

CONTENTS INCLUDE:

- Project Layout
- Configuring the Web Application
- About Pages and Components
- Tapestry Markup Templates
- Built-in Components
- Tapestry Annotations and more...

Apache Tapestry 5.0

By Howard M. Lewis Ship

ABOUT TAPESTRY 5

Tapestry 5.0 is a high-productivity, component-based, open source user interface tier for Java EE web applications. It combines simple, concise page templates with minimal Java classes (to contain state and business logic), and embraces convention over configuration, dynamically locating and adapting to your classes. This refcard covers Tapestry 5.0, (released as version 5.0.18 in December 2008), and describes the structure of a Tapestry application, the format of Tapestry markup templates, the standard components, and typical configuration options.

PROJECT LAYOUT

The first decision on a new project is the root package name. Tapestry will automatically locate Tapestry pages and components under the root package. Figure 1 shows a sample project layout for a root package name of com.whizzco.snuz.

Java classes to be compiled are stored under src/main/java. Additional resources to be packaged with the compiled classes are stored under src/main/resources.

The web application is under src/main/webapp, complete with a WEB-INF folder and a web.xml configuration file.

```
src/main/java/com/wizzco/snuz
  pages
    Index.java
  services
    AppModule.java
src/main/resources/com/wizzco/snuz/pages
  Index.properties
src/main/webapp
    Index.tml
    WEB-INF
      app.properties
      web.xml
```

Figure 1: Project layout with root package name of com.whizzco.snuz.

Hot Tip Follow this layout and the instructions on the Tapestry 5 home page and you can use *live class reloading*: changes to your page classes are picked up without a restart or redeploy.

CONFIGURING THE WEB APPLICATION

Tapestry's primary configuration comes from web.xml; this is where the application's root package is defined.

If you are using Maven, it can build a template project for you. Enter the following all on a single line:

```
mvn archetype:create -DarchetypeGroupId=org.apache.tapestry \
  -DarchetypeArtifactId=quickstart \
  -DgroupId=com.wizzco -DartifactId=snuz
  -DpackageName=com.wizzco.snuz
```

You can replace the parts in bold with values specific to your

Configuring the Web Application, continued

application. Maven creates an entire project structure, including a web.xml and starter pages and classes.

Additional configuration is accomplished via Tapestry's built-in Inversion of Control container. This takes the form of a module class that defines services and provides configuration. If an AppModule class exists in the services package (com.wizzco.snuz.services.AppModule), it is loaded automatically. Maven will create this file for you.

Let's turn Tapestry's production mode off, so that any exceptions will be reported in full detail:

```
package com.whizzco.snuz.services;

import org.apache.tapestry5.ioc.*;
import org.apache.tapestry5.SymbolConstants;

public class AppModule {
    public static void
contributeApplicationDefaults(MappedConfiguration<String,
String> configuration) {
    configuration.add(SymbolConstants.PRODUCTION_MODE,
    "false");
    }
}
```

SymbolConstants Field	Description	Default Value
CHARSET	Output and request encoding	UTF-8
COMPRESS_WHITESPACE	"true" to remove excess template whitespace, "false" to leave it in	true
PRODUCTION_MODE	"true" for abbreviated exceptions, "false" for the full exception report	true
SUPPORTED_LOCALES	Comma separated list of locale names. Often overridden to "en"	en, it, es, de, ...

Apache Tapestry 5

www.dzone.com

ABOUT PAGES AND COMPONENTS

In Tapestry, each page is a specific Java class in the pages package and has a page name that is based on the Java class name. Thus com.whizzco.snuz.pages.Index is page "Index" and com.whizzco.snuz.pages.profile.Edit is page "profile/Edit".

Note Tapestry is case-insensitive about page names ("Index", "index" and "iNdEX" are all equivalent). It is also case-insensitive about property names, parameter names, message keys, component types, and more.

Each page has a Tapestry markup template file that defines what components are used by the page. Some components also have templates and their own child components.

Every component has a type and an id. The type identifies what Java class will be instantiated. The id is unique to the component's container (the page, or containing component). Tapestry assigns an id if you don't.

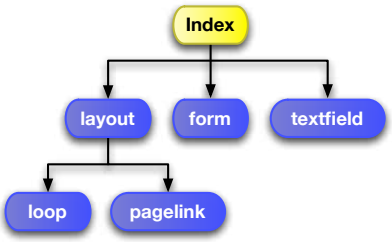


Figure 2: Pages contain components; some components contain other components.

Pages are not like servlets; they are not singletons. There will be many instances of each page. A page instance is only visible to one request at a time, and Tapestry uses a **page pool** to store page instances between requests.

Component classes may extend from other component classes (but not from other non-component classes). There is no required base component class in Tapestry; most components extend from Object.

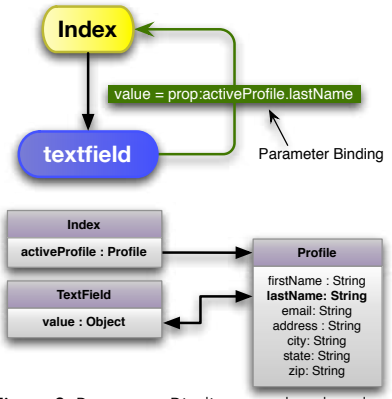


Figure 3: Parameter Bindings read and update properties

A page *binds* the parameters of its child components. For example, a TextField component's value parameter is bound to a property, or property expression, indicating what object property it will read (when rendering) or update (when the containing form is submitted). In Figure 3, the value parameter is bound to the Index page's activeProfile.lastName property.

The prop: prefix indicates a property expression binding; this is often the default and is usually omitted. Other binding prefixes fulfill other purposes, such as accessing localized messages from a message catalog, and are described later.

TAPESTRY MARKUP TEMPLATES

Tapestry template files are well-formed XML documents with a .tml extension. The file name must exactly match the class name (including case). Page templates can be stored on the classpath (under src/main/resources) or directly inside the web application (under src/main/webapp). See Figure 2. Templates are optional, and many components do not have a template.

Hot Tip Tapestry is not case insensitive about file names. You must match the case of the class name to the case of the template file name.

Tapestry Namespace

Tapestry uses the namespace http://tapestry.apache.org/schema/tapestry_5_0_0.xsd for its elements and attributes, usually assigned the prefix "t". A minimal, empty Tapestry page will define the namespace in its outermost ("html") element:

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">
</html>
```

Any elements that are not in the Tapestry namespace will ultimately be sent to the client web browser as-is. Markup templates may contain other namespaces as well; these too are sent to the client web browser as-is.

Using Expansions

Expansions allow you to read properties of the corresponding page (or component). Expansions start with a '${' sequence and end with a '}'. They may appear in ordinary text, including inside attribute values:

```
<dt>Account Balance:</dt>
<dd class="${balanceClass}">${balance}</dd>"
```

Note Expansions are not allowed inside element names or attribute names; just inside blocks of text or inside attribute values.

Hot Tip Expansions are just like parameter bindings: you can use a prefix. Often the message: binding prefix is used to access a localized message from the page's message catalog.

Adding Components

A Tapestry component appears in the template as an element inside the Tapestry namespace.

```
<t:textfield value="activeProfile.lastName"/>

<t:pagelink page="index">back to home page</t:pagelink>
```

The element name is the component type, and matches a component class. Tapestry searches the application's components package first, then the built-in core library if not found.

Component parameters are bound using attributes of the element.

Hot Tip You may assign your own id to a component with the t:id attribute. It's always a good idea to assign an id to any form-related component, or to ActionLink components; your choice will be shorter and more mnemonic than what Tapestry assigns, and component ids sometimes appear inside URLs or client-side JavaScript.

Body Element

The <t:body/> element is a placeholder for a component's

Tapestry Markup Templates, continued

body: the portion of its container's template enclosed by its element.

Component Blocks

A section of a template may be enclosed as a block:
<t:block id="nav"> ... </t:block>

Blocks may contain anything: text, elements, components or expansions. Blocks do not render in the normal flow; instead they can be rendered on demand (discussed in the Component Rendering section). Blocks can be injected into a field of type Block:

```
@Inject
private Block nav;
```

Block Parameters

Some components take a parameter of type Block. You can specify these using the <t:parameter> element:

```
<t:grid source="users">
  <t:parameter name="empty">
    <p>No users match the filter.</p>
  </t:parameter>
</t:grid>
```

BUILT-IN COMPONENTS

Tapestry includes a large suite of built-in components; essential components are described in the following tables. Full details of all components and component parameters are available at:
http://tapestry.apache.org/tapestry5/tapestry-core/ref/.

Link Component	Description
ActionLink	Triggers an "action" event on the component
EventLink	Triggers an arbitrary event.
PageLink	Creates a link to render another page in the application.

Dynamic Output Component	Description
If	Renders its body if a condition is met.
Loop	Renders its body multiple times, iterating over a collection.
Output	Formats and outputs an object using a Formatter.

Form Control Component	Description
Checkbox	Toggle Button
Errors	Displays input validation errors for the enclosing Form.
DateField	Text field with JavaScript popup calendar
Form	Container of form control components.
Label	Label for related form control component.
LinkSubmit	A JavaScript-enabled link to submit a form.
Palette	JavaScrript multiple selection and reordering.
PasswordField	Single line input field with input obscured.
Radio	Exclusive toggle button.
RadioGroup	Invisible component that organizes Radio components into an exclusive group.
Select	Drop down list.
Submit	Clicking form-submit button
TextArea	Multiple-line text input field.
TextField	Single line input field.

Tapestry's scaffolding components allow for quick user interfaces for ordinary JavaBeans to be assembled quickly and easily.

Built-in Components, continued

Scaffolding Component	Description
BeanDisplay	Displays all the properties of a JavaBean.
BeanEditor	Creates a UI for a JavaBean, providing different types of form controls for each property.
BeanEditForm	Combines a BeanEditor with a Form and submit button.
Grid	Tabular output of a set of JavaBeans, with paging and sorting.

Tapestry has built-in support for many common Ajax operations, built on top of Prototype and Scriptaculous. Many of the components have Ajax related parameters, and the following table lists components that exist just for Ajax.

Ajax Component	Description
AjaxFormLoop	Special looping component for use inside Forms to allow detail rows to be added or removed.
FormFragment	A portion of a Form that can be made visible or invisible.
FormInjector	Allows an existing Form to be extended in place.
Zone	A receiver of dynamic content from the server; used for in=place updates

TAPESTRY ANNOTATIONS

Tapestry uses annotations to change how fields and methods of your component classes are used.

Field Annotation	Description
InjectPage	Injects the page that ultiately contains this component as a read-only field.
Parameter	Defines the field as a formal parameter of the component. Fields may be optional or required, may allow or forbid null, and may have a default value.
Persist	Identifies fields whose value should persist between requests (stored in the session).
Property	Tapestry should generate a getter and setter method for the field.

Hot Tip Tapestry resets fields to their default values after each request. If you have data that needs to last longer than a single request, use the @Persist annotation.

Class Annotation	Description
IncludJavaScriptLibrary	Ensures that the specified JavaScript libraries are linked to in the output markup.
IncludeStylsheet	Ensures that the specified Cascading Stylesheet file is linked to in the output markup.
SuportsInformalParameters	Marks the component as supporting additional, non-formal parameters.

Method Annotation	Description
Log	Tapestry should log method entry and exit (at debug level).
OnEvent	Method is an event handler method.
Cached	Return value of method should be cached against later invocations.
CommitAfter	The Hibernate transaction should be committed after invoking the method.

Many additional method annotations are discussed later, in component rendering. All those annotations have a naming convention alternative.

PROPERTY EXPRESSIONS

Expressions are always evaluated in the context of a page or component. Using the . and ?. operators allows expressions to navigate a graph of objects and properties.

Expression Form	Description
true	Boolean.TRUE
false	Boolean.False
null	null
this	The current page or component
1234	A number as a java.lang.Long.
-1234.56	A number as a java.lang.Double.
foo	The name of a property.
bar()	The name of a method to invoke.
foo.bar	Nested property: Evaluate property foo, then property bar (can be repeated). May be used with method names.
foo?.bar	Safe dereference: Evaluate foo then bar, unless foo is null, in which case the expression's value is null.
'err'	A string literal, inside single quotes.

Property expressions are updatable only if the final form is a property, and there is a setter method for that property. The case of property and method names is ignored. The . and ?. operators can be called as many times as you need.

Hot Tip The safe dereference operator, ?., keeps you from having to nest multiple If components to safely access a nested property where some of the intermediate properties may be null.

BINDING PREFIXES

The default binding prefix for most component parameters is prop:, meaning a property expression. In certain cases, a particular parameter will have a different default binding prefix, often literal:.

Essential Binding Prefixes	Meaning
block:	The id of a block within the template.
component:	The id of a child component. Used to connect two components together (such as Label and TextField).
literal:	A literal string.
message:	A key from the component's message catalog.
prop:	A property expression.

COMPONENT EVENTS

Tapestry interacts with your application code by reading and updating properties, and by invoking event handler methods. Event handler methods may have any visibility. Component events may be triggered by a request from the client (such as a form submission) or may exist only on the server, or some mix thereof.

Events have a name: in many cases, the event name is "action". Some components trigger other events (see their documentation). Class EventConstants defines string constants for all events provided as part of Tapestry.

Component Events, continued

Events may have a context: one or more values that are passed as strings in the URL. Context values are converted back to appropriate types and appear as method parameters to event handler methods.

Naming Convention

Event handler methods are of the form "on*EventName*" or "on*EventName*From*ComponentId*". Examples:

```
void onSuccess() {
  ...
}

void onActionFromDelete() {
  ...
}
```

@OnEvent annotation

The @OnEvent annotation can be used instead of the naming convention:

```
@OnEvent(EventConstants.SUCCESS)
void storeOrderInDatabase() {
  ...
}

@OnEvent(component="delete")
void deleteLineItem() {
  ...
}
```

Tip: Use the @OnEvent annotation to support more descriptive method names.

Return values

Return values from event handler methods are used for page navigation:

Type	Meaning
String	Name of page to render.
Class	Class of page to render.
Object	Page instance to render (often via @InjectPage).
StreamResponse	Send byte stream direct to client.
java.net.URL	External URL to redirect to.
boolean	Return true to cancel event bubbling

Event bubbling

When an event is triggered on a component, the first step is to look in the component itself for an event handler method, then its container, then its container's container, and so forth. Event bubbling occurs when there is no event handler, or an event handler exists but returns null (or void). The event is re-triggered in the container. As the event bubbles, it always appears to be from the component last checked.

Form events

The Form component fires a series of events when rendering and when processing a submission:

Event Name	When	Usage
prepareForRender	render	Prepare page state to render the form.
prepareForSubmit	submit	Prepare page state to process the form submission.
prepare	render/submit	Prepare the page state to render and submit.
validateForm	submit	Perform final cross-field validations after all fields have processed.

Component Events, continued

success	submit	Invoked if no input field validation errors.
failure	submit	Invoked if there are input field validation errors.
submit	submit	Invoked last, regardless of validation errors.

COMPONENT RENDERING

Tapestry breaks the rendering of a component down into phases, shown in Figure 4. Each phase corresponds to a method of the component class (possibly a method inherited from a base class). The method names match render phase names (case insensitive). Alternately, you may attach a @SetupRender, @BeginRender, etc. annotation to a method.

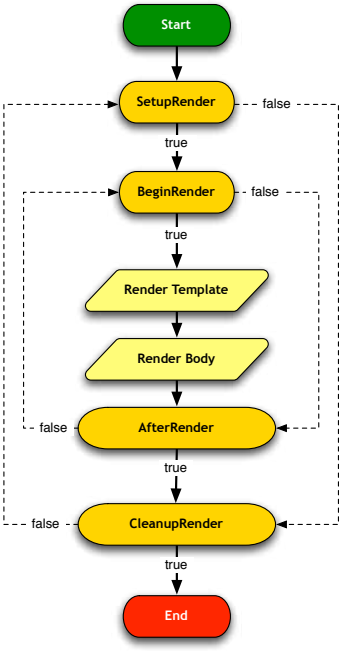


Figure 4: Component Rendering Phases

Normally, each phase follows the previous (the bold lines marked "true"). A render phase method may return **false** to jump backwards (the dotted lines). This is how conditional components (such as If) and looping components (such as Loop and Grid) operate.

Render phase methods may be any visibility. They may either take no parameters, or take a parameter of type **MarkupWriter**. The **MarkupWriter.element()** method is used to create a new output element (along with attributes) and should *always* be balanced by a call to **MarkupWriter.end()**.

Components will often start an element in the BeginRender phase, and end it in the AfterRender phase.

Components that support informal parameters should be marked with the **@SupportsInformalParameters** class annotation, and include a call to **ComponentResources.renderInformalParameters()**:

```
@Inject
private ComponentResources resources;

void beginRender(MarkupWriter writer) {
  writer.element("div", "class", "popup");
  resources.renderInformalParameters(writer);
}

void afterRender(MarkupWriter writer) {
  writer.end();
}
```

A render phase method may also return a component instance or a **Block** instance; the component or **Block** will take over and be rendered completely before the current component

Component Rendering, continued

advances to the next phase.

If a component has a template, it is rendered after the BeginRender phase. The template may include a **<t:body/>** element to render the component's body. A component without a template is treated as if its template consists of just a **<t:body/>** element.

COMPONENT LIFE CYCLE

Components, as part of pages, have a specific life cycle. Components are instantiated and added to pages and containing components. Once the page is fully constructed, components receive a one-time notification that the page has loaded. For the duration of a request, a page is attached to the request: there is a notification for the page being attached and later, for the page being detached (before it is returned to the page pool).

There are three method annotations for this purpose: **@PageLoaded**, **@PageAttached** and **@PageDetached**. Alternately, you can also use the method naming convention: **pageLoaded()**, **pageAttached()** and **pageDetached()**.

LOCALIZATION

Tapestry localization support is pervasive. Tapestry determines the client's desired locale based on standard HTTP headers, or the presence of a HTTP Cookie.

Pages and components may have their own individual message catalogs.

A message catalog is a set of files with similar names: **MyComponent.properties** for the default message catalog, or with locale names appended: **MyComponent_fr.properties** or **MyComponent_de.properties**.

Component subclasses may have message catalogs; subclass keys override super-class keys.

Applications may have a message catalog: **WEB-INF/app.properties**. Page and component message keys override application messages keys.

Messages are accessible inside templates:

${message:greeting}, ${user.firstName}!

You may inject your component's message catalog as a Messages object:

```
@Inject
private Messages messages;
```

Using the **@Inject** annotation on a field of type **java.util.Locale** will inject the page's locale. This can be used to format dates and currency values.

ENVIRONMENTAL OBJECTS

Environmental objects are request-scoped objects that are of use to your components. The **@Environmental** annotation dynamically binds a component field to an environmental object. Environmental objects don't have a name; the type of the Environmental is used to locate the object.

Environmental Objects, continued

Environmental Type	Usage
FormSupport	Details about the current form render, or form submission. Generation of unique control names within the form.
RenderSupport	Generation of unique element ids, and inclusion of JavaScript libraries and initialization snippets.

TAPESTRY SERVICES

Tapestry includes a number of useful services that can be injected into components or other services. Other services are useful not for injection, but to be configured.

Service	Description
ApplicationDefaults	Configuration overrides FactoryDefaults symbols used to configure other services.

Tapestry Services, continued

BeanModelSource	Source for BeandEditor that can be customized and provided to Grid and BeandEditor components.
ComponentSource	Provides access to pages or components.
Request	Current request object, used to obtain parameter values directly.

For More Information

Tapestry 5 Home Page:
http://tapestry.apache.org/tapestry5/

Detailed component reference (with examples):
http://tapestry.apache.org/tapestry5/tapestry-core/ref/

Tapestry Wiki:
http://wiki.apache.org/tapestry/

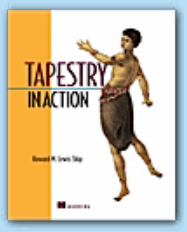
ABOUT THE AUTHOR

Howard M. Lewis Ship

Howard M. Lewis Ship is the creator of the Apache Tapestry project. He is a frequent speaker at JavaOne, ApacheCon, No Fluff Just Stuff, and other software conferences. He is the author of "Tapestry in Action" for Manning Publications, which covers Tapestry 3. He has trained dozens of developers in Tapestry over the last five years, is currently the Director of Open Source Technology at Formos, a Vancouver, Washington based consulting company.

Web site
http://howardlewisship.com
http://www.formos.com

RECOMMENDED BOOK



Tapestry in Action

The definitive guide to the Tapestry approach: creating full-featured web apps by connecting framework components to economical amounts of application code. Many simple examples show you how to tackle common tasks such as form validation, application localization, client-side scripting, and synchronization between browser and app server.

BUY NOW
books.dzone.com/books/tapestryinaction

Get More FREE Refcardz. Visit refcardz.com now!

Upcoming Refcardz:

Core Seam
Core CSS: Part III
Hibernate Search
Equinox
EMF
XML
JSP Expression Language
ALM Best Practices
HTML and XHTML

Available:

Essential Ruby
Essential MySQL
JUnit and EasyMock
Getting Started with MyEclipse
Spring Annotations
Core Java
Core CSS: Part II
PHP
Getting Started with JPA
JavaServer Faces
Core CSS: Part I
Struts2
Core .NET
Very First Steps in Flex
C#
Groovy
NetBeans IDE 6.1 Java Editor
RSS and Atom
GlassFish Application Server
Silverlight 2

Visit refcardz.com for a complete listing of available Refcardz.

Design Patterns
Published June 2008

DZone

DZone communities deliver over 4 million pages each month to more than 1.7 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more.
"DZone is a developer's dream," says PC Magazine.

DZone, Inc.
1251 NW Maynard
Cary, NC 27513
888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-43-1
ISBN-10: 1-934238-43-0
50795
9 781934 238431
$7.95