Jakarta Tapestry Project Documentation

1. Tapestry

1.1. Jakarta Tapestry - Welcome!

1.1.1. Introduction

Tapestry is a powerful, open-source, all-Java framework for creating leading edge web applications in Java.

Tapestry reconceptualizes web application development in terms of objects, methods and properties instead of URLs and query parameters.

Tapestry is an alternative to scripting environments such as JavaServer Pages or Velocity. Tapestry goes far further, providing a complete framework for creating extremely dynamic applications with minimal amounts of coding.

Tapestry's approach, using a component object model similar to a traditional GUI, provides the following benefits:

- Very high level of reuse, within and between projects
  Everything in Tapestry is a reusable component
- Frees developers from writing boring, buggy code
  Code in terms of objects, methods and properties, not URLs and query parameters
- Allows applications' complexity to scale well
  Framework does all the URL building and message dispatching, transparently
- Easy Internationalization/Localization
  Framework selects localized version of text, templates and images
- Extremely robust applications
  Less code is less bugs
  Sophisticated built-in exception reporting
  Line precise error reporting
- Easy team integration
  Graphic designers and Java developers can work together without having to know each

other's jobs

Tapestry is distributed under the terms of the Apache Software License.

Tapestry exploits the dynamic nature of the Java language, leveraging the JavaBeans API, as well as servlets and other J2EE technology. Tapestry applications are fast, scalable, robust and powerful.

Tapestry components are a combination of a specification file (in XML), an HTML template and a Java class (extending a framework class, with simple additions). Tapestry components are combined together to form larger components or complete Tapestry pages.

1.1.2. Tapestry in Print

Tapestry in Action Cover

Tapestry in Action is now available from Manning Publications. It is the definitive introduction to Tapestry written by Howard Lewis Ship, the creator of Tapestry.

Tapestry has also been described in the print journal *The Java Report* in the September 2001 issue. Other articles includes the on-line journal OnJava, in November 2001.

1.1.3. Tapestry Community

Tapestry has a very active User mailing list, with archives. This is the list for getting help with using the framework.

The Developer mailing list is for Tapestry committers and other power users to discuss enhancements to the framework. It also has archives.

A Wiki has been set up to discuss Tapestry and plan new features.

1.2.

1.3. Jakarta Tapestry - What's New

... or perhap's, "What's Old". These are release notes from earlier versions of Tapestry. More recent changes are in a different format, on the Change Log page.

1.3.1. Release 3.0-beta-1

- Major refurbishment of the Virtual Library example to use all the latest and greatest Tapestry facilities.
- Removed the "rows" and "columns" parameters from TextArea; use informal parameters "rows" and "cols" instead.

- Removed unused/unneeded exceptions `RollbackException` and `PageRecorderSerializationException`.
- The default for the Hidden component is now to encode its parameter.
- Ugraded to OGNL 2.4.2.
- Shell comp;onent now allows multiple stylesheets.
- Added a DataSqueezer adaptor for `Enum` types.
- Switched over to using JBoss 3.0.6 for demos.
- Renamed method `generateAttributes()` to `renderInformalParameters()` in AbstractComponent.
- Added support for the informal parameters of a component to be passed down to a contained component using the inherit-informal-parameters attribute.
- Added IRequestCycle.activate() as a replacement for setPage(). Besides setting the page to be rendered, it also invokes page.validate() and handles the page redirections.
- Added addValidateListener() and removeValidateListener() to IPage.
- DataSqueezer: Squeezed strings are now always prefixed with 'S'
- Added mechanism for checking that super-class implementations of key methods are invoked.
- Added checks that IPage methods `validate()` and `detach` are properly overriden by subclasses.
- Changed Form, Hidden and ListEdit so that all hidden fields are written out with the <form> tag.
- Added new features to the script specification, bumping its DTD to 1.3. It is now possible to generate page-unique ids from within the script (using the new `unique` attribute on `<let>`, as well as to render a block only once per page render (using `<unique>`)
- Integrated Per Norrman's vastly improved DatePicker component

[18340] ApplicationRuntimeException doesn't compile on jdk 1.3. [18336] Tapestry 2.4a5 - LOGGING images missing for Inspector component [18490] compile warning for Workbench [18013] typo in EvenOdd javadoc [18607] Check for unimplemented abstract methods [17904] Ongoing LGPL Issues [19153] Easier way to override validation messages [19263] Change error message for null parameter [18880] DatePicker broken under Mozilla

1.3.2. Release 2.4-alpha-5

- Added some simple optimizations to keep the engine instance from being stored into the session unecessarily often.
- Fix NPE when image parameter of Image component is bound but value is null.
- Create a basic JSP tag library to allow JSPs to access Tapestry pages using the page and external services.
- Added support for primitive arrays, java.lang.Object[] and java.lang.String[] for connected parameters.

- Added connected parameter support for missing primitive types byte and char
- Added support for primitive arrays, java.lang.Object[], and java.lang.String[] for declared properties.
- Replaced JFreeChart with JCharts, due to licensing considerations.
- Refactored to use Jakarta Digester to parse specifications.
- Changed specification and template parsers to track locations of specification objects and attach them to runtime objects and exceptions for error reporting purposes.
- Severely refactored exceptions, removing many exception classes and flattening all others under ApplicationRuntimeException.
- Simplified the URL format, merging the "service" and "context" parameters together.
- Removed the "displayWidth" and "maxLength" parameters from TextField and ValidField (HTML attributes "size" and "maxlength", as informal parameters, are sufficient).
- Added two new application extensions to allow page and component specifications and templates to be provided in non-standard ways (when not found using the default rules).
- Changed file upload to work using Jakarta Commons FileUpload (patch provided by Joe Panico).
- Added new parameter direction: `auto`, which creates a synthetic property backed by the binding.

[18249] file upload using Commons FileUpload [17905] Link to mailing list and archives is wrong.

1.3.3. Release 2.4-alpha-4

- All packages have been renamed from `net.sf.tapestry` to `org.apache.tapestry`.
- Several non-ASL libraries (including Jetty) have been removed from both CVS and the distribution. To build Tapestry and run the demos is now more involved; it requires obtaining several external dependencies. The Tapestry distribution is much smaller, however. This was done for licensing reasons. Sorry.
- Expression bindings in HTML templates are now in the format `attribute="ognl:expression"`.
- String bindings (to localized strings) in HTML templates are now in the format `attribute="string:key"`.
- Allow <set-property> element of specification to specify the expression as an attribute or as wrapped character data.
- The interfaces for IValidationDelegate, IValidator and ValidatorException changed incompatibly. This will only be an issue if you have created custom validation delegates or custom validators.
- Added methods to IComponentStrings for formatting localized strings with arguments.
- Remove ejb.jar and any direct dependencies on javax.ejb classes (application servers are

responsible for properly replicating EJBObject and EJBHome instances).
- Added a `createRequestCycle()` method to AbstractEngine.
- Moved the invocation of the IMonitor method `serviceEnd()` to always occur after the invocation of `serviceException()`.
- The Upload component now works with the enclosing Form to automatically set the encoding type to `multipart/form-data`. It is no longer necessary to set the `enctype` attribute of the Form.
- Removed the code related to making copies of persistent properties.
- Removed non-ASL libraries from CVS. These files will need to be downloaded separately.
- Removed some of the old tutorials, leaving just the Workbench and Virtual Library as examples.
- Removed the "Demo" pages from the web site, until we find a stable home.

1.3.4. Release 2.4-alpha-3

- Reorganized the packaging into a binary distribution (which includes documentation) and a second, smaller, source-only distribution.
- Renamed the JARs, stripping off the "net.sf." prefix.
- Updated all examples to use the 1.4 Specification DTD.
- Refactored (severely) the relationship between services and link components, splitting the rendering portion of links into a separate interface.
- Upgrade to McKoi database 0.94h.
- Tapestry will now create properties for connected parameters, if the properties do not already exist (or are abstract).
- Renamed the `java-type` attribute of the `<parameter>` element (in component specifications) to `type` (for the 1.4 DTD).
- Allowed more elements to specify values as character data inside the element as an alternative to using a particular attribute (useful for complex OGNL expressions).
- Continued extending the JUnit test suite.
- Deprecated the `PageCleanupListener` interface and removed support for it.

[665622] net.sf.tapestry.html.Frame uses old DOCTYPE [675882] option component generates invalid HTML [622691] Full release [679655] Upload component very slow on file uploads

1.3.5. Release 2.4-alpha-2

- Made improvements to how Tapestry handles arrays of objects and scalars
- Upgrade demos to deploy into JBoss 3.0.4
- Merge in changes from Tapestry 2.3
- `<binding>` elements may now specify the expression as the parsed data instead of the expression attribute

- The template extension may now be overriden using the configuration property `net.sf.tapestry.template-extension`
- Added support for declarative transient and persistent properties via `<property-specification>` element in page and component specifications. Tapestry will create (on the fly) a subclass with the necessary accessor methods and fields, as well as any necessary notification and cleanup methods.

[594878] Deploy Tapestry into JBoss 3.0.4 [672743] Pages Implementing Listeners cause NPE

1.3.6. Release 2.4-alpha-1

- Added support for specifying expressions in the component template. Expressions are specified as attributes with the format "`[[ expression ]]`". The brackets and leading and trailing whitespace are removed. Expressions specified this way are the equivalent of the `<binding>` element in a specification.
- Tapestry now supports implicit components. Implicit components are declared exclusively in the containing component's template (instead of in the containing component's specification) using a special `jwcid` syntax: `@type` (for anonymous components) or `id@type` (for named components). Implicit components are especially useful for components that take no parameters, but may also make use of template expressions.
- Added support for the `<listener-binding>` element in page and component specifications. This allows a listener method to be provided, directly within the specification, as a script written in a BSF-supported language.
- A number of non-backwards compatible changes were made to several framework interfaces to support more flexibility on where specifications and templates may be located, but these should not affect the overwhelming majority of Tapestry users. In addition, private assets and context assets may also be relative.
  Private assets are relative to the component specification which declares them, context assets are relative to the application servlet (within the servlet context).
- Moved the Inspector out of the framework and into the contrib library.
- Created smarter checks for stale sessions for ActionLink, DirectLink and Form. The action and direct services used by the components now encode whether the application was stateful into the URL. The stateful check only occurs if the application was stateful when the URL was rendered.
- Changed Form to record the exact ids generated during the render (it used to just store the count). This allows a more useful exception message to be generated for the StaleLinkException.
- Changed the default StaleLink page to have a message property, and to display the message from the StaleLinkException.
- Components (and even pages) can now implement page listener interfaces

PageDetachListener, PageRenderListener or PageCleanupListener) and `finishLoad()` will automatically add them as a listener to the corresponding page events.

- The entire mechanism used to store persistant page properties has been revised.
- Implemented a number of improvements to Pool to support greater flexibility in managing objects stored into and discarded from the pool.

[653358] IPage.getName() == qualified name [608768] Changes saved AFTER IPage.detach()

1.3.7. More ...

Earlier entries

1.4. Jakarta Tapestry - Quotes

This page contains quotes from the Tapestry community, appearing on the Tapestry mailing lists and in online forums such as TheServerSide.com.

These entries have been editted for grammar and readability.

1.4.1. Dorothy Gantenbein

We used Tapestry to implement the monitoring and administration console for a wireless network management product. Initially, we implemented a demonstration using standard JSP but we realized that JSPs presented difficulties for implementation. Some of our requirements were a very aggressive development schedule, integration with EJB 2.0 beans, ability to write modular reusable components, very reliable, easy code maintenance, internationalization, and of course easier debugging than the generated JSP code. Tapestry met all of our requirements and helped us proceed onto a successful release.

After reading the tutorial and reviewing the examples, we started with writing simple components. We started with a StatusImage component. The GIF for the status image should be localized and selected based on standard network status states. Another example of a simple component was a validating IP address text field. The IP address field uses the Tapestry validation framework. After that, we moved onto more complex components like the StatusTable shown in the figure. This table has a varying number of rows depending on the configuration of the product using the Tapestry Foreach component (very cool). This StatusTable makes use of another essential Tapestry component, the Conditional. Looking at the Actions columns, the set of actions is conditional based on the row. All this logic happens in the Java class and is not embedded into our HTML making maintenance much easier.

Finally, we integrated servlets and JSPs into our Tapestry web application. We needed

servlet integration for charting and JSPs for reporting. With all this flexibility, we could use each technology where they worked best. Overall, all of Tapestry's flexibility along with its clean object-oriented architecture made our web interface shine.

Screen shot 1 Screen shot 2

1.4.2. Paul Witt

Of all the web presentation frameworks out there, Tapestry is the most elegant.

For me ELEGANT is the goal of a programmer/developer (I've been around awhile). When, in your MVC framework, VIEW is pure HTML, and your CONTROL - controls (wow - nice control; exception/debugger/session handlers they are -- well, elegant) and the link to your model is - often as simple (elegant) as get/set, then the philosophy of web applications has moved to a new and cleaner level.

But elegance is a moving target, and Tapestry, with this latest release 3.0, puts me not only on board (I already am) but I've fastened my seatbeat, and I'm ready for takeoff.

1.4.3. Miles Egan

The learning curve for Tapestry is still somewhat steep ... but I'm really enjoying it now that I'm beginning to get it and, most importantly, I don't feel like I'm writing much, if any, superfluous code. It really seems like all the code directly contributes to functionality and that there's very little boilerplate. One of the hallmarks of a good framework, in my opinion.

1.4.4. Andrus Adamchik

You know, Tapestry in fact is different from your average JSP taglib. Anyone who had a chance to use component-based web frameworks (like WebObjects, I don't know any other in this category) would know what I am talking about. This is a *different* and highly powerful and convenient way of doing things.

Surprisingly enough with so many web application layers out there, Tapestry seems to be the only one (OpenSource, that is) that allows you to create a set of reusable components and build the design around this. The level of reuse is unmatched with anything JSP. In many ways it is even stronger than commercial implementation of the same design idea (WebObjects). In particular it makes it extremely easy for graphics designers to modify HTML, since dynamic content does not use *any* custom tags. Also the amount of code (and time) it takes to hook up an average model layer to the HTML presentation is so small that at first you may think you've missed something along the way :-).

1.4.5. Chris Wilson

I've said it before and I'll say it again... You owe it to yourself to take a serious look at Tapestry. It's the most exciting model in Java web development I've seen in a long

time. It really allows you to build applications--not web sites--that "execute on the web." Very cool stuff.

1.4.6. David Solis

Tapestry is the best solution to the presentation tier and it provides a clean separation of content and developer code. First, I extended it for WAP support, then I developed several WAP components and finally I developed a complete email application for WAP in record time.

1.4.7. Jim Birchfield

When using Struts or Velocity, in order to see what the finished page will look like, our graphic designers must assemble the application, deploy it ... they basically lose all the luxuries of WYSIWYG development ... I spent 30 minutes or so with one of our graphic designers here, and he was giddy at what I showed him as I redesigned one of our smaller apps using Tapestry.

Bottom line for us is this: We have graphic designers and we have developers, and I don't want them to necessarily have to know each others' job to get things done. Tapestry is the best framework I have seen that offers this sort of separation.

1.4.8. Jiri Lundak

As a 'lay developer' using Tapestry I can only recommend it. We have built a content management system based on Tapestry, plus multiple sites that are now powered by it. Thanks to its component architecture, we are able to publish any kind of our metadata-driven business objects with just a handful of generic, highly dynamic pages, using customizable components. All business logic is encapsulated in business objects and services, so there is not really much logic in the presentation layer besides the navigation parts.

The learning curve is not flat, but is really worth the effort. If you need to bring a dynamic web-app to the browser Tapestry offers many predefined and stable components to build on. We have gone with Tapestry and we will continue to use it in the next couple of projects needing a web-interface, too.

1.4.9. Malcom Edgar

I just wanted to send you a quick note of appreciation for your Tapestry framework. It is a truly original and innovative approach.

While the learning curve is steep compared to thinner webapp frameworks, I have found it is well worth the effort. The amount of code I am having to write is greatly reduced and I am

achieving a much greater level of reuse.

1.4.10. Mind Bridge

I just want to second the positive comments about Tapestry. We have evaluated a lot of frameworks in the past couple of years, and in our opinion Tapestry is the only open source Java framework at the moment that provides everything necessary for Web-centric component oriented development.

The most important feature of Tapestry is indisputably the ability to package common Web functionality into components that can then be effortlessly reused again and again. Building a Tapestry application is pretty much like playing with Lego bricks -- all you have to do is just fit the different pieces together and not worry about how each brick would work internally or how the different pieces would interract with one another (an OO approach at its best). As a result, after gaining some experience and accumulating a set of components, one can build incredibly complex Web applications quickly and easily, and yet be assured that those applications are of a very high quality.

Tapestry has another major advantage as well -- it provides very clean separation between presentation and logic (cleaner than that of the most commonly used frameworks). The design of its templates is also very mindful of the actual web development process, accomodates very nicely the work of the web designers from the start, and allows them to play along even in later stages of the application development. This, again, is something rarely found in the other frameworks.

I will mention just one more superb feature -- the Tapestry design and APIs provide a huge number of extension points that allow you to customize the behaviour of the framework almost to no end. In order to achieve a specific goal for one customer we twisted the behaviour of the system so much, that Howard would faint if he sees it. In any case, being able to easily modify and extend specific elements of the framework is a major advantage.

This may sound a bit like an advertisement, but it is not -- we are just very satisfied users. We've got a significant increase in productivity, we've got increased quality, we've got happy web designers, and we've put an end to the repetitive operations -- it would be hard to think of a better scenario.

If you are experienced in OO and/or components and care about Web development, I would suggest that you have a look at Tapestry. I would be very surprised if you do not find it a far more viable solution for your needs than JSP or Struts, for example.

1.4.11. Luis Neves

Actually what drove me to Tapestry wasn't the saved development time, it was the sheer beauty of it :-)

I come from an ASP/VB background and I was getting sick and tired of that mess, designers constantly asking me "Is it OK to move this part of code to the bottom of the page?" , "Could you take a look and see if I didn't mess anything when I added the picture?", "Ohh, I'm sorry. that was an include file, I thought it was just a comment" ... well you get the picture, and tools like Dreamweaver only help to some extent. The perfect separation of roles was what drew me to Tapestry ... the rest of the things (and there are a lot of them) were the iceing on the cake.

1.4.12. Adam Greene

The company that I work for did a research project with the National Research Council of Canada, and we spent several months reviewing web services, J2EE, and Database systems for a technical risk project that we were doing. Out of that we learned 3 things:

1. Web services were immature.
2. J2EE was overrated.
3. That JSP and anything else that stuck anything other than HTML into an HTML file was not only counter-productive but dangerous to the life of the project.

I will give you an example. We have a "legacy" project that is done in JSP (all of our development efforts have moved to Tapestry). We had to perfect the logic of the project while the text and look of the project was still in flux. Everytime our graphics guys fixed the pages, code would get damaged, or vice versa. Everytime our graphics guys created new pages, our programmers would have to go mark up the pages with code and could not work on the logic of the given page until after our graphics guy had finished it. On top of that the whole system has to be translated into French. Which means that we not only have doubled our work, we have actually ended up tripling it, pushing back deadlines, etc. If it had not been for the fact that we "inherited" the code, it would have been done in Tapestry to start with, because:

1. Our graphics designers could do their thing without disturbing the work of our programmers.
2. The programmer wouldn't have to wait for the designers to finish a page to start work on it. Many times our programmers work with pages that are simple white with text and form fields and after it works, our designers integrate the pages together.
3. Localizing for French simply involves copying the existing layout, replacing English text with French, and saving it with a new file extension ... no need to recreate the code of a JSP.
4. If our client doesn't like the look, it is a simple matter of changing it, no code is disturbed, no code is broken, it will work when it is done being modified without our programmers

even looking at it. (And our designers don't even know Java....)

If you are using a language or framework that puts proprietary (non-HTML) code into your templates or HTML files, I seriously suggest that you take a look at the approach that Tapestry takes as you will probably find that it is quite revolutionary in its approach to web development (as stated above).

So take it from someone who just researched existing and upcoming technology for three months, Tapestry frags the snot out of frameworks like JSF, Struts, and the like.

I will give Apache one thing though, they have built a lot of production quality software that beats even Microsoft. We added Torque to Tapestry to fill a void in the database support (Tapestry by itself has no database support) because Tapestry's model fit so perfectly that I can actually create a web page to display all the records in a table using only 14 lines of Java code, of which less than half (only 6 lines of code) are actually hand written (the rest are auto-generated by Eclipse). So I guess that is another point for Tapestry: Integration of other packages / APIs that support a Java Beans style is a no-brainer. No taglibs need to be written, no new scripting needs to be created, it simply works.

1.5. Jakarta Tapestry - Documentation

1.5.1. Tapestry API Documentation

Full documentation for the Tapestry framework, the `contrib` framework (which contains additional components and classes), and all the example code.

[HTML]

1.5.2. Tapestry Component Reference

A handy refrence to the built-in Tapestry components, with example specifications, HTML templates and code.

[HTML]

1.5.3. Tutorial

The best way to begin learning about the Tapestry framework; this document eases the reader into basic Tapestry concepts.

[HTML] [PDF]

Out of date
This Tutorial is out of date, and work to replace it is currently taking place.

1.5.4. Developer's Guide

Exhaustive reference for Tapestry, in extreme detail.

[HTML] [PDF]

Out of date

This document is out of date, and is being replaced with a new User's Guide.

1.5.5. User's Guide

Complete reference to the Tapestry framework.

[HTML] [PDF]

Incomplete

This document is currently being constructed and is incomplete. It is a replacement for the Developer's Guide.

1.5.6. Contributor's Guide

Guide for users and prospective developers who wish to contribute code and patches to the Tapestry project.

[HTML] [PDF]

1.6.

1.7. Jakarta Tapestry - FAQs

1.7.1. Questions

1. **General Tapestry Information**
   - How does Tapestry compare to other frameworks?
   - How is the performance of Tapestry?
   - Is Tapestry a JSP tag library?
   - What does it cost?
   - Is there a WYSIWYG editor for Tapestry, or an IDE plugin?
   - Does Tapestry work with other other application servers besides JBoss?
2. **Technical Questions**
   - I have to restart my application to pick up changes to specifications and templates, how can I avoid this?
   - What is "line precise error reporting"?

3. **Other Frameworks**
    - How do I intergrate Tapestry with Spring?

1.7.2. Answers

1.7.2.1. 1. General Tapestry Information

1.1. How does Tapestry compare to other frameworks?

Tapestry is very much unlike most other frameworks in that it doesn't use code generation; instead it uses a true component object model based on JavaBeans properties and strong specifications. This gives Tapestry a huge amount of flexibility and enables dynamic runtime inspection of the application with the Tapestry Inspector (a mini-application that can be built into any Tapestry application).

In addition, Tapestry applications require far less Java coding and are far more robust than equivalent applications developed with other popular frameworks. This is because the Tapestry framework takes responsibility for many important tasks, such as maintaining server-side state and dispatching incoming requests to appropriate objects and methods.

The many new features of release 3.0 mean that Tapestry is not only the most powerful web application framework available, it is also the fastest and easiest to adopt, regardless of whether your background is Java, Perl, XML or PHP!

1.2. How is the performance of Tapestry?

My own testing, documented in the Sept. 2001 issue of the Java Report, agrees with other testing (documented in the Tapestry discussion forums): Although straight JSPs have a slight edge in demo applications, in real applications with a database or application server backend, the performance curves for equivalent Tapestry and JSP applications are identical.

Don't think about the performance of Tapestry; think about the performance of your Java developers.

1.3. Is Tapestry a JSP tag library?

Tapestry is *not* a JSP tag library; Tapestry builds on the servlet API, but doesn't use JSPs in any way. It uses it own HTML template format and its own rendering engine.

Starting with release 3.0, Tapestry includes a simple JSP tag library to allow JSP pages to create links to Tapestry pages.

1.4. What does it cost?

Tapestry is open source and free. It is licensed under the Apache Software License, which allows it to be used even inside proprietary software.

1.5. Is there a WYSIWYG editor for Tapestry, or an IDE plugin?

Currently, no WYSIWYG editor is available for Tapestry; however, the design of Tapestry allows existing editors to work reasonably well (Tapestry additions to the HTML markup are virtually invisible to a WYSIWYG editor).

Spindle is a Tapestry plugin for the excellent open-source Eclipse IDE. It adds wizards and editors for creating Tapestry applications, pages and components.

1.6. Does Tapestry work with other other application servers besides JBoss?

Of course! JBoss is free and convienient for the turn-key demonstrations. You can download Tapestry and JBoss and have a real J2EE application running in about a minute! The scripts that configure JBoss are sensitive to the particular release of JBoss, it must be release 3.0.6.

However, Tapestry applications are 100% container agnostic ... Tapestry doesn't care what servlet container it is used with and does not even require an EJB container.

1.7.2.2. 2. Technical Questions

2.1. I have to restart my application to pick up changes to specifications and templates, how can I avoid this?

Start your servlet container with the JVM system parameter `org.apache.tapestry.disable-caching` set to true, i.e., `-Dorg.apache.tapestry.disable-caching=true`.

Tapestry will discard cached specifications and templates after each request. You application will run a bit slower, but changes to templates and specifications will show up immediately. This also tests that you are persisting server-side state correctly.

2.2. What is "line precise error reporting"?

Tapestry applications are built from templates and specifications. It's natural that when these templates and specifications are read, any syntax errors are reported, and the precise file and location is identified.

Tapestry goes far beyond that! It always relates runtime objects back to the corresponding files so that even runtime errors report the file and location.

Line Precise

Tapestry exception report (click for larger image).

For example; say you bind a parameter of a component that expects a non-null value, but the value ends up being null anyway, due to a bug in your code or your specification. Tapestry can't tell, until runtime, that you made a mistake ... but when it does, part of the exception report will be the line in the template or specification where you bound the component parameter. Zap! You are sent right to the offending file to fix the problem.

Other frameworks may report syntax errors when they parse their specifications, but after that, you are own your own: if you are lucky, you'll get a stack trace. Good luck finding your error in that! Tapestry gives you a wealth of information when unexpected exceptions occur, usually more than enough to pinpoint the problem *without* having to restart the application inside a debugger.

1.7.2.3. 3. Other Frameworks

3.1. How do I intergrate Tapestry with Spring?

Spring is a popular service framework. Colin Sampaleanu has written a integration document on using these two open-source frameworks together.

1.8. Todo List

1.8.1. Release 3.0

- **[code]** Handle change of locale correctly, by reloading new instance of page in proper locale # open
- **[code]** Improved User's Guide to replace existing Developer's Guide # HLS
- **[code]** Replace the current tutorial with Neil Clayton's Tutorial2 # open
- **[code]** Reoganize directory structure to more standard format (compatible with Maven) # open
- **[code]** Get Tapestry compiling under Maven, have nightly builds # open
- **[code]** Fill out the test suite and code coverage, reach 85% or better # open
- **[code]** Fix all the Component Reference pages to use the 3.0 syntax # open
- **[script]** Improve the "include script" element to support relative scripts, scripts in context, script in classpath # open
- **[misc]** Unit testing stategy for the JSP tags and tagsupport service # open
- **[misc]** Allow auto parameters to not be required # open

1.8.2. Release 3.1

- **[code]** Intersertials # open
- **[code]** Extend listener methods to take parameters (matching service parameters to actual method parameters) # open

1.9. History of Changes

RSS

1.9.1. Version 3.0-beta-4 (unreleased)

- Change direction of parameter model for WML components, SelectionField and PropertySelection (DS)
- Fixed the byte[] mapping in org.apache.tapestry.enhance.JavaClassMapping (DS)
- Convert the Tapestry home page to build using Forrest. (HLS) Thanks to Tetsuya Kitahata.
- Updated javadoc build target to link to JDK 1.3 () Thanks to Michael M. Brzycki.
- Added new validator, Pattern Validator (HK)
- Fixed the null pointer exception from FieldLabel when not enclosed by Form or when IValidationDelegate is not provided (HK)
- The key attribute of the script foreach is changed to not be a required attribute (HK)
- Added support for OGNL TypeConverter in expression bindings (EH)
- Updated Component Reference for WML components (DS)
- Made DateValidator's getEffectiveFormat()/getEffectiveDisplayFormat() public (RLS)
- Updated to use DocBook 1.62.4 XSL stylesheets for documentation generation (RLS)
- Moved info priority logging to debug priority (RLS)
- Custom parameter binding properties only enhanced if abstract (RLS)
- Updated the Table components to have an easier interface, similar to that of Foreach. (MB)
- build.properties.sample does not include jython.dir (MB) Fixes 21833.
- Table component doesn't sort fine when there are null values (MB) Fixes 22840.
- Wrong Package in link (MB) Fixes 22640.
- JavaDoc: should IField be IFieldTracking?? (MB) Fixes 22641.
- contrib:PopupLink ignores informal parameters (MB) Fixes 23668.
- Wrong Package in link (MB) Fixes 22635.
- Typo: "In a IListenerBindingSpecification" (MB) Fixes 22634.
- Add url parameter to Shell component (MB) Fixes 22694.
- current OGNL 2.6.3 needs to be used (MB) Fixes 23870.
- Problem with URL encodings at EngineServiceLink.java (MB) Fixes 23511.
- The source parameter of Foreach should be required (MB) Fixes 23227.

- tests failing under JDK 1.3 (MB) Fixes 20252.
- Name attribute gets duplicated in TextField (MB) Fixes 23500.
- DateField component throws an ApplicationRuntimeException (MB) Fixes 22835.
- NumericField component throws an ApplicationRuntimeException (MB) Fixes 22836.
- Using NumericField cause a ClassCastException (MB) Fixes 22837.
- Compatibility of the AbstractEngine to servlet-api 2.2 (MB) Fixes 24467.
- patch to allow workbench to work with JDK 1.4.2 and current Jetty (MB) Thanks to Colin Sampaleanu. Fixes 24298.
- race condition in class enhancement (MB) Fixes 24425.
- The Dates tab in the Workbench contains an empty popup link (MB) Fixes 23916.
- ApplicationRuntime Exception loses root cause... (MB) Fixes 24008.
- NumberValidator forces to input a value (MB) Fixes 22958.
- junit.jar needed to build junit subproject (MB) Fixes 21871.
- JUnit XML tests broken under JDK with no XML parser (eg. 1.3) (MB) Fixes 20253.
- No JVM requirements mentioned in docs (MB) Fixes 24393.
- [PATCH]ValidationStrings_zh_TW.properties (MB) Thanks to Zhenbang Wei. Fixes 24874.
- PageService.getLink throws ClassCastException if parameters arg is Object[] (EH) Fixes 25117.
- NumericField does not pass on its type parameter (RLS) Fixes 25462.
- ValidationDelegate throws NPE for some new FieldLabel/ValidField (RLS) Fixes 25585.
- properties cannot be of complex array types (MB) Fixes 25642.
- fixed broken links in doc\src\common\TapestryLinks.xml (EH) Fixes 25766.
- Inherited parameters do not pick up default values (MB) Fixes 26395.
- Changed code to no longer invoke `StringUtils.isEmpty() / isNonEmpty()` (this is because the behavior of the method is changing between jakarta-commons 1.0 and 2.0). (HLS)
- Add an implementation of `toString()` to `RequestCycle`. (HLS)
- Update all copyrights for 2004. (HLS)
- Add download links. (HLS)
- Remove unncessary constructor from test case classes. (HLS)
- Changed mock unit tests to redirect System.out and System.err to log files rather than the console. (HLS)
- Improve the documentation for the Palette component, providing a real example of CSS styles used with the component. (HLS) Thanks to Glen Stampoultzis.
- Component parameters with direction 'form' should not allow static bindings. (HLS) Fixes 26416.
- Fixed NPE in PatternValidator.toString(). (HK) Fixes 26599.
- Fix TestMocks to not use JDK 1.4 API. (HLS)
- Automatically download external dependencies (such as Forrest and McKoi DB). (MB)

1.10. Jakarta Tapestry - Developers

1.10.1. Jakarta Tapestry - Developers

Tapestry is a community project, and now follows the Apache Software Foundation meritocracy rules to guide its future development. Tapestry Developers plan features, discuss and vote on the Tapestry Developers' Mailing List.

The following are the current Tapestry committers.

1.10.1.1. Mind Bridge

Mind Bridge has been a long term contributor to Tapestry.

1.10.1.2. Neil Clayton

Neil has contributed to the Component Reference, and undertaken the daunting task of creating a new (and actually useful) Tapestry Tutorial, which is nearing completion.

1.10.1.3. Erik Hatcher

Erik began by contributing Javadoc fixes (flagged by IDEA) as he was learning Tapestry. He has added the OGNL type converter facility for expression bindings and is active in making the Table component easy to use.

1.10.1.4. Harish Krishnaswamy

Harish is a Tapestry enthusiast and is also becoming a HiveMind enthusiast. Hopefully he will help integrate the two.

1.10.1.5. Richard Lewis-Shell

Richard is one of the real early adopters of Tapestry.

1.10.1.6. Howard Lewis Ship

Howard started the Tapestry project in early 2000.

1.10.1.7. Geoff Longman

Geoff has advanced Tapestry light years with his excellent Eclipse plug-in, Spindle.

1.10.1.8. David Solis

David is another long-term user, and has added a suite of WML components to the framework.

1.11.

2. Project

3. Useful Information

4. Downloads

5. Related Projects

6. Whole Site