Configuration

Configuring Tapestry

This page discusses all the ways in which Tapestry can be configured. Tapestry applications are configured almost entirely using Java, with very little XML at all.

Contents

Related Articles

- Tapestry IoC Configuration
- Response Compression
- Symbols
- Application Module Class Cheat Sheet
- IoC cookbook - Service Configurations
- Configuration

XML configuration (web.xml)

Tapestry runs on top of the standard Java Servlet API. To the servlet container, such as Tomcat, Tapestry appears as a *servlet filter*. This gives Tapestry great flexibility in matching URLs without requiring lots of XML configuration.

Although most configuration is done with Java, a small but necessary amount of configuration occurs inside the servlet deployment descriptor, WEB-INF /web.xml. Most of the configuration is boilerplate, nearly the same for all applications.

web.xml (partial)

```
<!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
      "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>My Tapestry Application</display-name>
    <context-param>
        <param-name>tapestry.app-package</param-name>
        <param-value>org.example.myapp</param-value>
    </context-param>
    <filter>
        <filter-name>app</filter-name>
        <filter-class>org.apache.tapestry5.TapestryFilter</filter-class>
        <!-- Or org.apache.tapestry5.http.TapestryFilter if you're using tapestry-http without tapestry-core -->
    </filter>
    <filter-mapping>
        <filter-name>app</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

The application-specific part, the `tapestry.app-package` context parameter, provides your application's root package name. Tapestry uses this to locate your page and component classes. It expects page classes in the `pages` sub-package and components in the `components` sub-package. In the example above, page classes will be stored in the `org.example.myapp.pages` package (or in sub-packages below). Likewise, component classes will be stored in the `org.example.myapp.components` package.

By convention, the filter name (`filter-name`) is almost always "app", but you can use any name you want. Tapestry uses this to determine what *module class* name to look for (see below).

Tapestry Requests vs. Container Requests

The Tapestry filter matches all the requests that apply to Tapestry, and passes the rest off to the servlet container. In situations where there would be a naming conflict, actual files inside the web application take precedence over Tapestry pages.

Tapestry recognizes the *root URL*, where the servlet path is simply "/", and renders the application page "Index", if it exists.

Your Application's Module Class

Main Article: Tapestry IoC Configuration

Most other configuration occurs inside your application's module class. The application module class will often define new services, provide overrides of services, or make contributions to service configurations.

Tapestry looks for your application module class in the services package (under the root package) of your application. It capitalizes the <filter-name> and appends "Module". In the previous example, because the filter name was "app" and the application's root package name is "org.example.myapp", the module class would be org.example.myapp.services.AppModule.

If such a class exists, it is added to the IoC Registry. It is not an error for your application to not have a module class, though any non-trivial application will have one.

Your application module class (usually AppModule.java) will typically override some of Tapestry's default, or "factory", symbols, by contributing overrides to the ApplicationDefaults service configuration. For example:

AppModule.java

```
public class AppModule
{
  public static void contributeApplicationDefaults(MappedConfiguration<String,String> configuration)
  {
    configuration.add(SymbolConstants.SUPPORTED_LOCALES, "en,fr,de");
    configuration.add(SymbolConstants.FILE_CHECK_INTERVAL, "10 m");
  }
}
```

Configuration Symbol Names

Main Article: Symbols

Many of Tapestry's built-in services (some of which are not even public) are configured via symbols. These symbols can be overridden by contributing to the ApplicationDefaults service configuration, or by placing a <context-param> element into the application's web.xml, or on the command line by defining JVM System Properties with the -D command line option.

These symbols are always defined in terms of strings, and those strings are coerced to the appropriate type (a number, a boolean, etc.). Of special note are *time intervals*, which are specified in a particular format.

Most of these symbols have a constant defined in the SymbolConstants class, while others are in the IOCSymbols class. Those are noted in **bold** below. Use the symbol name (tapestry.*) for JVM System Properties with the -D option, and use the constant (in bold below) from within your Java classes (e.g. AppModule.java).

tapestry.app-catalog

SymbolConstants.APPLICATION_CATALOG – The location of the global application message catalog, the default is context:WEB-INF/*app-name*.properties.

tapestry.application-version

Added in 5.4

SymbolConstants.APPLICATION_VERSION – The version of the application, which is incorporated into URLs for context and classpath assets in Tapestry versions prior to 5.4. Assets may be compressed, and will have far-future expiration headers; they will be aggressively cached by the client web browser. You should change the application version on each new deployment of the application (that is, any time assets in the context change), to force clients to re-download changed versions of files. If you do not specify an application version, a *random* one will be assigned on every deployment (which is good for development but very bad for production).

tapestry.application-folder

Added in 5.3

SymbolConstants.APPLICATION_FOLDER – The folder, of the context, in which the Tapestry application executes. By default this is blank, meaning the Tapestry application executes in the root of the web application context. Setting this value allows the Tapestry application to be segregated into a folder, which can be useful when Tapestry is executed inside a web application with other servlets or filters.

tapestry.asset-url-fully-qualified

> **Added in 5.3**

SymbolConstants.ASSET_URL_FULL_QUALIFIED – A boolean value to indicate whether asset URLs should be fully qualified in the rendered page. This defaults to `false` (not fully qualified).

tapestry.asset-path-prefix

> **Added in 5.3.1**

SymbolConstants.ASSET_PATH_PREFIX – The prefix to be used for all asset paths. This should start *and* end with a slash ("/"). By default this is "/assets/".

tapestry.blackbird-enabled

SymbolConstants.BLACKBIRD_ENABLED – A flag (true or false). When "false" the Blackbird JavaScript console will be disabled (in Tapestry 5.2 and newer). Defaults to "true".

> **Deprecated since 5.3**
>
> The client-side BlackBird console has been removed.

tapestry.bootstrap-root

> **Added in 5.4**

SymbolConstants.BOOTSTRAP_ROOT – The root asset path for Twitter Bootstrap; if your application uses a modified version of Bootstrap, you can override this symbol to have Tapestry automatically use your version. The value should be a path to a folder (under "classpath:" or "context:") and should not include a trailing slash.

tapestry.font-awesome-root

> **Added in 5.5.0**

SymbolConstants.FONT_AWESOME_ROOT – The root asset path for FontAwesome; if your application uses a modified version of it, you can override this symbol to have Tapestry automatically use your version. The value should be a path to a folder (under "classpath:" or "context:") and should not include a trailing slash.

tapestry.charset

SymbolConstants.CHARSET – The character encoding used when generating output (or parsing input). The default is "UTF-8". See Content Type and Markup for more details.

tapestry.clustered-sessions

Added in 5.3

SymbolConstants.CLUSTERED_SESSIONS – If "true" then at the end of each request the SessionPersistedObjectAnalyzer will be called on each session persisted object that was accessed during the request. The default is "true", to preserve 5.2 behavior. For non-clustered applications (the majority), this value should be overridden to "false".

tapestry.combine-scripts

SymbolConstants.COMBINE_SCRIPTS – If "true", then Tapestry will combine (or "aggregate") the individual JavaScript libraries within a JavaScript stack; this reduces the number of requests from the client to the server, as the client can cache the combined JavaScript files locally (and will not need to re-download them on subsequent pages). The implementation of this changed significantly between Tapestry 5.1 and 5.2.

Defaults to "true" in production mode.

tapestry.compact-json

Added in 5.2

SymbolConstants.COMPACT_JSON – If "true", then JSON page initialization content is compressed; if "false" then extra white space is added (pretty printing). Defaults to "true" in production mode.

tapestry.compatibility.unknown-component-id-check-enabled

Added in 5.3

Deprecated since 5.3

SymbolConstants.UNKNOWN_COMPONENT_ID_CHECK_ENABLED – When enabled, Tapestry will check that component ids referenced in event handler method names (or the @OnEvent annotation) match up against components in the container's template. The default is true, but applications upgraded form Tapestry 5.2 may want to set this to false, to keep pages from failing due to the presence of such dead code.

tapestry.component-render-tracing-enabled

SymbolConstants.COMPONENT_RENDER_TRACING_ENABLED – Starting with version 5.3, if "true" then Tapestry will emit rendering comments for all requests; these are comments (such as <!--BEGIN Index:loop (context:Index.tml, line 15)-->) that can assist you in debugging markup output on the client-side. This will significantly increase the size of the rendered markup, but can be very helpful with complex layouts to determine which component was responsible for which portion of the rendered page. (To turn on rendering comments only for a particular request, add the query parameter t:component-trace=true to the URL.)

tapestry.compress-whitespace

SymbolConstants.COMPRESS_WHITESPACE – A flag (true or false). When true (the default) whitespace in component templates is compressed by default (this can be fine-tuned using the standard xml:space attribute on an element in the template). When this flag is false, then whitespace is retained by default (but can still be overridden). See Component Templates for details.

tapestry.module-path-prefix

Added in 5.4

SymbolConstants.MODULE_PATH_PREFIX – Prefix used for all module resources. This may contain slashes, but should not begin or end with one. Tapestry will create two Dispatchers from this: one for normal modules, the other for GZip compressed modules (by appending ".gz" to this value).

tapestry.context-path

Added in 5.4

SymbolConstants.CONTEXT_PATH – Identifies the context path of the application, as determined from ServletContext.getContextPath() method. This is either a blank string or a string that starts with a slash but does not end with one.

tapestry.datepicker

Added in 5.2

SymbolConstants.DATEPICKER – The path to the assets of the embedded DatePicker component

tapestry.default-cookie-max-age

SymbolConstants.COOKIE_MAX_AGE – The default time interval that cookies created by Tapestry will be kept in the client web browser. Primarily, this is used with a cookie that exists to track the preferred user locale. The default value is "7 d" (7 days; see Time Interval Formats).

tapestry.default-stylesheet

Added in 5.3.6

SymbolConstants.DEFAULT_STYLESHEET – In 5.3, this is the default stylesheet automatically injected into every rendered HTML page. Many Tapestry components assume that this stylesheet is available. All the classes defined in the stylesheet are prefixed with "t-". The exact contents of the stylesheet are subject to change at any time (they are considered internal), so replacing the stylesheet, rather than overriding selected rules within it, entails some risk.

The default is org/apache/tapestry5/default.css, stored on the classpath.

Deprecated in 5.4 with no replacement. The stylesheet is now associated with the core JavaScriptStack.

Undeprecated in 5.5.0. The stylesheet defined by this symbol is needed when Tapestry is configured to not include Bootstrap at all.

tapestry.error-css-class

Added in 5..5

SymbolConstants.ERROR_CSS_CLASS – Defines the CSS class that will be given to the HTML element generated by the Error component. If the value isn't `help-block`, the class attribute
will be `help-block [symbol value]`. The default value is `help-block`.

tapestry.enable-html5-support

Added in 5.4

SymbolConstants.ENABLE_HTML5_SUPPORT – If "true", then certain HTML5 features are invoked by built-in Tapestry components. Mostly this controls whether the TextField component will emit HTML5 "type" attributes automatically when certain validators are used. See Forms and Validation for details. The default is false.

tapestry.enable-minification

Added in 5.3.6

SymbolConstants.MINIFICATION_ENABLED – If "true", then resources (individually or when aggregated into stacks) will be minimized via the ResourceMinimizer service. If "false", then minification is disabled. The default is "true" in production mode, "false" otherwise.

Note that Tapestry's default implementation of ResourceMinimizer does nothing; minification is provided by add-on libraries. See Assets for details.

tapestry.enable-pageloading-mask

Added in 5.4

SymbolConstantsaENABLE_PAGELOADING_MASK – If true, then when a page includes any JavaScript, a `<script>` block is added to insert a pageloader mask into the page to ensure that the user can't interact with the page until the page is fully initialized. The default is true.

tapestry.encode-locale-into-path

SymbolConstants.ENCODE_LOCALE_INTO_PATH – If "true" (the default), then the PersistentLocale will be encoded into URLs by the ComponentEventLinkEncoder service. If overridden to "false" this does not occur, but you should provide a LinkCreationListener2 (registered with the LinkCreationHub) in order to add the locale as a query parameter (or provide some alternate means of persisting the locale between requests). See Localization for more details on localization.

tapestry.exception-report-page

SymbolConstants.EXCEPTION_REPORT_PAGE – The name of the page used to report exceptions. This defaults to "ExceptionReport", a page that Tapestry provides. See Overriding Exception Reporting for details.

tapestry.exception-reports-dir

> **Added in 5.4**

SymbolConstants.EXCEPTION_REPORTS_DIR – The root directory where Tapestry's built-in OperationTracker will create dated folders into which it writes exception report files. This is `build/exceptions` by default but should be overridden for production. See the related `tapestry.restrictive-environment` symbol below.

tapestry.execution-mode

SymbolConstants.EXECUTION_MODE – The execution mode. See Setting Execution Modes below.

tapestry.file-check-interval

SymbolConstants.FILE_CHECK_INTERVAL – Time interval between file system checks. During a file system check, only a single thread is active (all others are blocked) and any files loaded are checked for changes (this is part of Tapestry's Class Reloading mechanism).

The default is "1 s" (one second; see Time Interval Formats), and is usually overridden with a higher value in production (say, between one and five minutes).

tapestry.file-check-update-timeout

SymbolConstants.FILE_CHECK_UPDATE_TIMEOUT – Time interval that Tapestry will wait to obtain the exclusive lock needed for a file check. If the exclusive lock can't be obtained in that amount of time, the request will proceed normally (without the check), but each successive request will attempt to get the lock and perform the check until successful.

The default is "50 ms" (50 milliseconds; see Time Interval Formats).

tapestry.force-absolute-uris

> **Deprecated since 5.2**
>
> Starting in Tapestry 5.2, the optimization to generate relative URIs has been removed, and all URIs are always absolute. *Removed in 5.3*.

SymbolConstants.FORCE_ABSOLUTE_URIS *– For Tapestry 5.0 and 5.1 only*: when false (the default), Tapestry will attempt to optimize URIs that it generates, using relative URIs when such URIs are shorter than absolute URIs. When true, all URIs will be absolute URIs (including the context path, and the complete path for the request).

tapestry.gzip-compression-enabled

SymbolConstants.GZIP_COMPRESSION_ENABLED – Override to "false" to disable GZIP compression of dynamic Tapestry pages and static assets.

tapestry.hostname

> **Added in 5.3**

SymbolConstants.HOSTNAME – The hostname that application should use when constructing an absolute URL. The default is "", i.e. an empty string, in which case system will use request.getServerName(). Not the same as environment variable HOSTNAME (but you could contribute "$HOSTNAME" as the value to make it the same).

tapestry.hostport

Added in 5.3

SymbolConstants.HOSTPORT – The port that application should use when constructing an absolute URL. The default is "0", which means to use the port value from the request.

tapestry.hostport-secure

Added in 5.3

SymbolConstants.HOSTPORT_SECURE – The secure (https) port that application should use when constructing an absolute URL. The default is "0", i.e. use the value from the request.

tapestry.hmac-passphrase

Added in 5.3.6

SymbolConstants.HMAC_PASSPHRASE – The plaintext phrase used to set the key for HMAC securing of serialized object data. The default is blank, which causes a runtime alert and console error. You should set this to a reasonably unique, private value, and ensure (in a cluster) that all servers use the same value – typically by making a contribution in your applications module class (normally AppModule.java). See Security for details.

tapestry.include-core-stack

Added in 5.4

SymbolConstants.INCLUDE_CORE_STACK – Whether to include Tapestry's "core" stack of JavaScript libraries. The default is "true".

tapestry.javascript-infrastructure-provider

Added in 5.4

SymbolConstants.JAVASCRIPT_INFRASTRUCTURE_PROVIDER – Tapestry relies on an underlying client-side JavaScript infrastructure framework to handle DOM manipulation, event handling, and Ajax requests. Prior to Tapestry 5.4, the foundation was Prototype. In 5.4 and later, support for jQuery has been added, and it is possible to add others. This symbol defines a value that is used to select a resource that is provided to the ModuleManager service as a JavaScriptModuleConfiguration to provide a specific implementation of the `t5/core/dom` module. Tapestry 5.4 directly supports "prototype" or "jquery". To support other foundation frameworks, override this symbol value and supply your own module configuration.

In Tapestry 5.4, this defaults to "prototype" for compatibility with 5.3. This will likely change in 5.5 to default to "jquery". At some point in the future, Prototype support may no longer be present.

tapestry.lenient-date-format

> **Added in 5.4**

SymbolConstants.LENIENT_DATE_FORMAT – When set to true, the DateField component will be lenient about date calculations, for example allowing a January 32 date as input and automatically converting it to February 1. When false (the default), only valid dates may be entered.

tapestry.min-gzip-size

SymbolConstants.MIN_GZIP_SIZE – The minimum stream size necessary for Tapestry to use GZIP compression on the response stream. See Response Compression for more details.

tapestry.omit-generator-meta

SymbolConstants.OMIT_GENERATOR_META – If "true", then the <meta> tag that Tapestry normally writes into the <head>, identifying the Tapestry version, will be omitted. Use this when you do not wish to advertise your application's use of Tapestry.

tapestry.page-pool.active-window

> **Deprecated since 5.2**
>
> Starting in 5.2, this is only used if tapestry.page-pool-enabled is "true". *Removed in 5.3*

The time interval that an instantiated page instance may be cached before being removed. As pages are returned to the pool, they are time stamped. Periodically (as per the file check interval), the pool is scanned for page instances that have not been used recently; those that are outside the active window are discarded. This is used to free up unnecessary page instances after a request surge. Starting in 5.2, this is only effective if tapestry.page-pool-enabled is true.

The default is "10 m" (10 minutes; see Time Interval Formats).

tapestry.page-pool-enabled

Starting with Tapestry 5.2, page pooling has been turned off by default. This symbol lets you re-enable page pooling. Under most circumstances this symbol should not be set. The disabling of page pooling starting in 5.2 significantly reduces heap memory usage and improves performance for most web applications.

The default is "false".

> **Deprecated since 5.2**
>
> Removed in 5.3.

tapestry.page-pool.hard-limit

> **Deprecated since 5.2**

Starting in 5.2, this is only used if tapestry.page-pool-enabled is "true". *Removed in 5.3*

The absolute maximum number of page instances (for a particular page name / locale combination) that Tapestry will create at any time. If this number is reached, then requests will fail because a page instance is not available ... this can happen as part of a denial of service attack. For this value to have any meaning, it should be lower than the number of threads that the servlet container is configured to use when processing requests.

The default is 20 page instances.

tapestry.page-pool.soft-limit

Deprecated since 5.2

Starting in 5.2, this is only used if tapestry.page-pool-enabled is "true". *Removed in 5.3*

The number of pages in the page pool (for a given page name / locale combination) before which Tapestry will start to wait for existing pages to be made available. Under this limit of pages, Tapestry will simply create a new page instance if no existing instance is readily available. Once the soft limit is reached, Tapestry will wait a short period of time (the soft wait interval) to see if an existing page instance is made available. It will then create a new page instance (unless the hard limit has been reached).

The default is 5 page instances. Remember that page pooling is done separately for each page (and localization of the page).

tapestry.page-pool.soft-wait

Deprecated since 5.2

Starting in 5.2, this is only used if tapestry.page-pool-enabled is "true". *Removed in 5.3*

The time interval that Tapestry will wait for a page instance to become available before deciding whether to create an entirely new page instance.

The default is "10 ms" (10 milliseconds; see Time Interval Formats).

tapestry.page-preload-mode

Added in 5.4

SymbolConstants.PRELOADER_MODE – Controls in what environment page preloading should occur. By default, preloading only occurs in production. Possible values are "ALWAYS", "DEVELOPMENT", "NEVER", or "PRODUCTION" (the default is PRODUCTION when in production mode, or DEVELOPMENT otherwise). See PreloaderMode.

tapestry.persistence-strategy

SymbolConstants.PERSISTENCE_STRATEGY – Identifies the default persistence strategy for all pages that do not provide an override. The default is "session" (PersistenceConstants.SESSION).

tapestry.production-mode

SymbolConstants.PRODUCTION_MODE – A flag (true or false) indicating whether the application is running in production or in development. The default is true, which means that runtime exceptions are not reported with full detail (only the root exception message is displayed, not the entire stack of exceptions, properties and other information shown in development mode).

tapestry.restrictive-environment

Added in 5.4

SymbolConstants.RESTRICTIVE_ENVIRONMENT – A flag (true or false) that, if true, changes some default Tapestry behavior to make it work better in restrictive environments such as Google App Engine (GAE). Specifically, if true, then OperationsTracker writes its exception report files into a single folder (specified by the tapestry.exception-reports-dir symbol, above) rather than creating dated sub-folders under that path, and ResourceTransformerFactory avoids creating a cache folder for resources.

tapestry.secure-enabled

SymbolConstants.SECURE_ENABLED – If true, then @Secure annotations are honored; if false, no security checks or redirects take place. This defaults to tapestry.production-mode, meaning that in development mode it will (by default) be disabled. However, sites that are intended to be served *only* under HTTPS should set this to **`false`**. See HTTPS for details.

tapestry.secure-page

MetaDataConstants.SECURE_PAGE – If true, then the page may only be accessed via HTTPS. The @Secure annotation will set this value to true. This symbol is the default for all pages; set it to "true" to force the entire application to be secure. See HTTPS for details.

tapestry.service-reloading-enabled

Added in 5.2

If true (the default), then Tapestry IoC will attempt to reload service implementations when they change. This only applies to classes that Tapestry IoC instantiates itself, and have a known service interface (the container creates a proxy that, internally, can reload the implementation). Service reloading only works when the underlying class files are on the filesystem ... it is intended for development, not as an option in production.

This must be specified as a JVM system property. *You may not set it in your module class.*

tapestry.scriptaculous

Added in 5.2

SymbolConstants.SCRIPTACULOUS – The path to the embedded copy of script.aculo.us packaged with Tapestry. This value may be overridden to use a different version of the script.aculo.us library. See Legacy JavaScript for the default version.

tapestry.session-locking-enabled

Added in 5.4

SymbolConstants.SESSION_LOCKING_ENABLED – If true (the default), then Tapestry will use a lock when reading/updating HttpSession attributes, to avoid simultaneous access by multiple threads when using AJAX. See TAP5-2049. Set to false to deactivate the session locking logic.

Prior to version 5.4 session locking was not performed.

tapestry.start-page-name

SymbolConstants.START_PAGE_NAME – The logical name of the start page, the page that is rendered for the *root URL*. This is normally "start". This functionality is vestigial: it has been superseded by the use of Index pages.

tapestry.strict-css-url-rewriting

SymbolConstants.STRICT_CSS_URL_REWRITING – Controls whether to throw an exception (true) or log a warning (false) when Tapestry encounters a URL reference to a non-existing file within a CSS file. The default is false.

tapestry.supported-locales

SymbolConstants.SUPPORTED_LOCALES – A comma-separated list of supported locales. Incoming requests as "narrowed" to one of these locales, based on closest match. If no match can be found, the first locale in the list is treated as the default.

The default is (currently) "en,it,es,zh_CN,pt_PT,de,ru,hr,fi_FI,sv_SE,fr_FR,da,pt_BR,ja,el". As the community contributes new localizations of the necessary messages files, this list will expand. Note that the Tapestry quickstart archetype overrides the factory default, forcing the application to be localized only for "en".

tapestry.cors-enabled

SymbolConstants.CORS_ENABLED – Defines whether the CORS (Cross-Origing Resource Sharing) support should be enabled or not. Default value is `false`. If you set this to `true`,
you should also set at least `Symbol.CORS_ALLOWED_ORIGINS` too.

Added in 5.8.2

tapestry.cors-allowed-origins

SymbolConstants.CORS_ALLOWED_ORIGINS – Comma-delimited of origins allowed for CORS. The special value * means allowing all origins. This is used by the default implementation of `CorsHandlerHelper.getAllowedOrigin(HttpServletRequest)`. Default value is the empty string (i.e. no origins allowed and CORS actually disabled).

Added in 5.8.2

tapestry.cors-allow-credentials

SymbolConstants.CORS_ALLOW_CREDENTIALS – Boolean value defining whether the `Access-Control-Allow-Credentials` HTTP header should be set automatically in the response for CORS requests. Default value is `false`. This is used by the default implementation of `CorsHandlerHelper.configureCredentials(HttpServletResponse)`.

Added in 5.8.2

tapestry.cors-allow-methods

SymbolConstants.CORS_ALLOW_METHODS – Value to be used in the `Access-Control-Allow-Methods` in CORS preflight request responses. This is used by the default implementation of `CorsHandlerHelper.configureMethods(HttpServletResponse)`. Default value is `GET,HEAD,PUT,PATCH,POST,DELETE`.

Added in 5.8.2

tapestry.cors-allowed-headers

SymbolConstants.CORS_ALLOWED_HEADERS – Value to be used in the `Access-Control-Allow-Headers` in CORS preflight request responses. This is used by the default implementation of `CorsHandlerHelper.configureAllowedHeaders(HttpServletResponse)`, which only sets the header if the value isn't empty. Default value is the empty string.

Added in 5.8.2

tapestry.cors-expose-headers

SymbolConstants.CORS_EXPOSE_HEADERS – Value to be used in the `Access-Control-Expose-Headers` in CORS preflight request responses. This is used by the default implementation of `CorsHandlerHelper.configureExposeHeaders(HttpServletResponse)`, which only sets the header if the value isn't empty. Default value is the empty string.

Added in 5.8.2

tapestry.cors-max-age

SymbolConstants.CORS_MAX_AGE – Value to be used in the `Access-Control-Max-Age` in CORS preflight request responses. This is used by the default implementation of `CorsHandlerHelper.configureMaxAge(HttpServletResponse)`, which only sets the header if the value isn't empty. Default value is the empty string.

Added in 5.8.2

tapestry.suppress-redirect-from-action-requests

Deprecated since 5.2

Removed in 5.3

SymbolConstants.SUPPRESS_REDIRECT_FROM_ACTION_REQUESTS – Normally, Tapestry responds to action requests (such as form submissions) by sending a client-side redirect to the rendering page. This has a lot of benefits in terms of improving browser navigation, making sure URLs are bookmarkable, and so forth. However, it has a cost: more data stored persistently in the session, and a double-request for each user action (one action request, one render request).

Setting this symbol to "true" changes the Tapestry behavior to make it more like Tapestry 4: a markup response is sent directly for the action request, with no redirect in the middle. This option should be used with care, and only in cases where you are certain that the benefits outweigh the disadvantages.

tapestry.thread-pool.core-pool-size

IOCSymbols.THREAD_POOL_CORE_SIZE – Nominal size of the thread pool Tapestry uses to execute tasks in parallel. Under sufficient load, the thread pool may grow larger than this core size. Defaults to 3.

tapestry.thread-pool.max-pool-size

IOCSymbols.THREAD_POOL_MAX_SIZE – Maximum size of the thread pool Tapestry uses to execute tasks in parallel. Defaults to 10.

tapestry.thread-pool.queue-size

> **Added in 5.3**
>
> Previously, the queue size was unbounded, which meant that max-pool-size was ignored.

IOCSymbols.THREAD_POOL_QUEUE_SIZE – Size of the task queue for the thread pool. Once the core pool size is reached, new threads are not created until the queue is full. The default queue size is 100.

tapestry.thread-pool.keep-alive

IOCSymbols.THREAD_POOL_KEEP_ALIVE – The time to keep a created but unused thread in the pool alive. Defaults to one minute.

tapestry.thread-pool-enabled

IOCSymbols.THREAD_POOL_ENABLED – If set to false, then parallel task execution does not occur. This is useful in environments where creating new threads is not allowed, such as Google App Engine.

Setting Component Parameter Defaults

> **Added in 5.3**
>
> Some components, notably Grid, Pallete and Zone, have default parameter values specified in terms of symbols. This means you can use these symbols to modify the defaults for all instances of such components in your application. For example, you can set the default rows per page for all Grid instances by adding this to the `contributeApplicationDefaults` method in your application's module class (typically AppModule.java): `configuration.add(ComponentParameterConstants.GRID_ROWS_PER_PAGE, "15");`

See the complete list of such constants at ComponentParameterConstants.

Configuring Ignored Paths

You may sometimes need to use Tapestry in concert with other servlets. This can cause problems, since Tapestry (being a servlet filter) may see URLs intended for another servlet and attempt to process them.

The Servlet API does not provide Tapestry with any clues about what other servlets are available in the web application. Instead, you must configure Tapestry to ignore paths intended for other servlets.

The IgnoredPathsFilter service is the method for this kind of configuration. Its configuration is an unordered collection of regular expression patterns. A request whose path matches any of these patterns is **not** processed by Tapestry.

For example, say you are using Direct Web Remoting. You'll likely have the servlet path /dwr mapped to the Direct Web Remoting servlet.

You contribution would look like:

```
public static void contributeIgnoredPathsFilter(Configuration<String> configuration)
{
  configuration.add("/dwr/.*");
}
```

The regular expression matches any path that begins with "/dwr/".

The regular expressions provided in the configuration are always compiled with case insensitivity enabled.

Also note that actual files in your web application (images, stylesheets, etc.) are always ignored by Tapestry.

Configuring Content Type Mapping

The mapping from file type (by extension) to content type is typically done as part of your servlet-containers configuration. Alternately, you may contribute to the ResourceStreamer service's configuration. This is a mapped configuration; it maps file extensions (such as "css" or "js") to content types ("text/css" or "text/javascript") respectively.

Setting Execution Modes

Starting with Tapestry 5.2.4, we can specify an *execution mode* by loading specific Tapestry Modules through a JVM System property. All modules declared in this way will be loaded after the AppModule of your application. This feature is very useful for defining a different environment for Production and Development modes, for example.

This JVM System property, named tapestry.execution-mode, is a comma-separated list of mode names. You can declare this property in a number of different ways:

1. Add the parameter to your JVM command line:

```
-Dtapestry.execution-mode=uat jetty:run
```

2. Add the parameter to the Jetty plugin:

pom.xml

```
<plugin>
    <groupId>org.mortbay.jetty</groupId>
    <artifactId>maven-jetty-plugin</artifactId>
    <version>6.1.16</version>
    <configuration>
        <requestLog implementation="org.mortbay.jetty.NCSARequestLog">
            <append>true</append>
        </requestLog>
        <systemProperties>
            <systemProperty>
                <name>tapestry.execution-mode</name>
                <value>uat</value>
            </systemProperty>
        </systemProperties>
    </configuration>
</plugin>
```

3. Add the parameter to the Surefire plugin for your test:

pom.xml

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.8.1</version>
    <configuration>
       <systemPropertyVariables>
         <tapestry.execution-mode>uat</tapestry.execution-mode>
       </systemPropertyVariables>
    </configuration>
</plugin>
```

For each mode declared in your JVM System Property, TapestryFilter checks for a parameter in your web.xml, named tapestry.TheModeName-modules, with TheModeName being the name of the desired mode. Its value will be a comma-separated list of modules.

If the tapestry.execution-mode is not declared, Tapestry will automatically look for the tapestry.production-modules parameter, because “production” is the default tapestry.execution-mode value.

The example below defines two different execution modes in your web.xml file: production (the default value) and uat (for "user acceptance testing"). For each mode, we list the modules we want to load. If we use JVM System property declared in the example above, the UatModeModule module will be loaded.

web.xml

```
<context-param>
    <param-name>tapestry.production-modules</param-name>
    <param-value>com.example.myapp.services.ProductionModeModule</param-value>
</context-param>
<context-param>
    <param-name>tapestry.uat-modules</param-name>
    <param-value>com.example.myapp.services.UatModeModule</param-value>
</context-param>

<context-param>
    <param-name>tapestry.integration-modules</param-name>
    <param-value>com.example.myapp.services.IntegrationModeModule</param-value>
</context-param>
```

Execution mode itself may be a comma separated list:

```
-Dtapestry.execution-mode=uat,integration jetty:run
```

Segregating Applications Into Folders

In many cases where Tapestry is being adopted into an existing web application (possibly written in Tapestry 4 or some other framework), it is nice to segregate the Tapestry application into its own folder, to avoid conflicts with the existing application or servlets.

> **Added in 5.3**
>
> Support for application folders was added in release 5.3.

Setting this up is in two parts:

- Modifying the configuration of the `<url-pattern>` for the Tapestry filter to match the specified folder.
- Identifying the folder name using a Tapestry symbol value contribution.

So, if you wanted to run the Tapestry application inside folder `t5app`, you would modify your `web.xml` indicate the use of the folder:

```
  <filter-mapping>
    <filter-name>app</filter-name>
    <url-pattern>/t5app/*</url-pattern>
  </filter-mapping>
```

... and in your AppModule, you would inform Tapestry about the mapping change:

```
public class AppModule
{
    @Contribute(SymbolProvider.class)
    @ApplicationDefaults
    public static void applicationDefaults(MappedConfiguration<String, String> configuration)
    {
        configuration.add(SymbolConstants.APPLICATION_FOLDER, "t5app")
    }
}
```

This extra mapping is unfortunately necessary, because the Servlet API does not provide a way for a servlet filter, such as the one used by Tapestry, to know about its mapping.

This changes the servlet container to *only* forward requests inside the `t5app` folder to Tapestry; requests for other folders (or the root folder) will not be passed to Tapestry at all. The symbol contribution informs Tapestry to change the URLs it generates to include the necessary folder name; it also affects the logic in Tapestry that recognizes and handles requests.

In addition, if you choose to place page template files in the context, rather than on the classpath (as with component templates), then you will place those template files inside the `t5app` folder.

At this time, it is still not possible to run multiple Tapestry 5 applications within the same web application.