Tapestry5HowToXhtml

Tapestry by default generates HTML or XML depending on the content type of your application.

Before you try and make your application XHTML compliant you may want to look at HTML 5

The method varies by version

- Tapestry 5.0 %3C 13
- Tapestry 5.0 %3E 13
- Tapestry 5.1

Tapestry 5.0 < 13

Step 1:

create your own implementation of MarkupWriterFactory:

```
public class XhtmlMarkupWriterFactoryImpl implements MarkupWriterFactory {

    private final MarkupModel xmlModel = new XMLMarkupModel();

    public MarkupWriter newMarkupWriter(ContentType contentType) {
        return new MarkupWriterImpl(xmlModel, contentType.getParameter("charset"));
    }
```

Step 2:
Add an alias override to your module:

```
    public static void contributeAlias(Configuration<AliasContribution> configuration) {
        configuration.add(AliasContribution.create(MarkupWriterFactory.class, new
XhtmlMarkupWriterFactoryImpl()));
    }
```

That's it. Now: validate!

Demo/test snippet:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"  xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">
<head>
        <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
        <title>xhtml test</title>
</head>
        <body>
                valid xhtml?
        </body>
</html>
```

Optional: To prevent weird scripting problems in Firefox et al., replace the xmlMarkupModel with the version below:

```
import static org.apache.tapestry.ioc.internal.util.CollectionFactory.newSet;

import java.util.Set;

import org.apache.tapestry.dom.DefaultMarkupModel;
import org.apache.tapestry.dom.EndTagStyle;

public class ImprovedXhtmlMarkupModel extends DefaultMarkupModel {

    private static final Set<String> DONT_ABRV = newSet("script", "div", "span", "p");

    @Override
    public EndTagStyle getEndTagStyle(String element) {
        boolean isDontAbr = DONT_ABRV.contains(element);
        return isDontAbr ? EndTagStyle.REQUIRE : EndTagStyle.ABBREVIATE;

    }
    @Override
    public boolean isXML() {
        return true;
    }
}
```

Tapestry 5.0 > 13

The version above does not work for last version of T5 (something like above 5.0.13). This is an adaptation of the code working with 5.0.14 :

1/ Replace the XhtmlMarkupWriterFactoryImpl by this one :

```
package org.linagora.linra.core.tapestry.services.impl;

import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

import org.apache.tapestry5.ContentType;
import org.apache.tapestry5.MarkupWriter;
import org.apache.tapestry5.dom.DefaultMarkupModel;
import org.apache.tapestry5.dom.EndTagStyle;
import org.apache.tapestry5.dom.MarkupModel;
import org.apache.tapestry5.internal.services.MarkupWriterImpl;
import org.apache.tapestry5.services.MarkupWriterFactory;

public class XhtmlMarkupWriterFactoryImpl implements MarkupWriterFactory {

        private final String applicationCharset;

        private final MarkupModel xmlModel = new DefaultMarkupModel() {
            private final Set<String> DONT_ABRV = new HashSet<String>(Arrays.asList("script", "div", "span",
"p","textarea", "select"));

            @Override
            public EndTagStyle getEndTagStyle(String element) {
                boolean isDontAbr = DONT_ABRV.contains(element);
                return isDontAbr ? EndTagStyle.REQUIRE : EndTagStyle.ABBREVIATE;

            }

            @Override
            public boolean isXML() {
                return true;
            }
        };

    public XhtmlMarkupWriterFactoryImpl(String applicationCharset) {
        this.applicationCharset = applicationCharset;
    }

    public MarkupWriter newMarkupWriter(ContentType contentType) {
        return new MarkupWriterImpl(xmlModel, contentType.getParameter("charset"));
    }

    public MarkupWriter newMarkupWriter(String pageName) {
        return new MarkupWriterImpl(xmlModel, applicationCharset);
    }

}
```

2/In AppModule, add the injection of default charset :

```
/**
 * Override the default MarkupWritterFactory with one that always output XHTML tags.
 * This is because we want to use "text/html" content type (to be understood by IE)
 * _and_ output valid XHTML.
 */
public static void contributeAlias(
        Configuration<AliasContribution<MarkupWriterFactory>> configuration,
        @Inject @Symbol(SymbolConstants.CHARSET) final String applicationCharset) {

        configuration.add(AliasContribution.create(MarkupWriterFactory.class,
                        new XhtmlMarkupWriterFactoryImpl(applicationCharset)));
        }
```

Tapestry 5.1

NOTE : This currently isn't working as overriding the service is failing.

Create a class XhtmlMarkupWriterFactory to replace the marker factory

```
public class XhtmlMarkupWriterFactory implements MarkupWriterFactory {
	private class XhtmlMarkupModel extends AbstractMarkupModel {
		private final Set<String> DONT_ABRV = new HashSet<String>(
				Arrays.asList("script", "div", "span", "p", "textarea", "select"));

		public EndTagStyle getEndTagStyle(String element) {
			boolean isDontAbr = DONT_ABRV.contains(element);
			return isDontAbr ? EndTagStyle.REQUIRE : EndTagStyle.ABBREVIATE;

		}

		public XhtmlMarkupModel() {
			this(false);
		}

		public XhtmlMarkupModel(boolean useApostropheForAttributes) {
			super(useApostropheForAttributes);
		}

		/**
		 * Returns true.
		 */
		public boolean isXML() {
			return true;
		}
	}

	private final PageContentTypeAnalyzer analyzer;

	private final RequestPageCache cache;


	private final MarkupModel xmlModel = new XhtmlMarkupModel();

	private final MarkupModel xmlPartialModel = new XhtmlMarkupModel(true);

	public XhtmlMarkupWriterFactory(PageContentTypeAnalyzer analyzer, RequestPageCache cache) {
		this.analyzer = analyzer;
		this.cache = cache;
	}

	public MarkupWriter newMarkupWriter(ContentType contentType) {
		return newMarkupWriter(contentType, false);
	}

	public MarkupWriter newPartialMarkupWriter(ContentType contentType) {
		return newMarkupWriter(contentType, true);
	}

	@SuppressWarnings({ "UnusedDeclaration" })
	private MarkupWriter newMarkupWriter(ContentType contentType, boolean partial) {
		boolean isHTML = contentType.getMimeType().equalsIgnoreCase("text/html");

		MarkupModel model = partial
				? (xmlPartialModel)
				: (xmlModel);

		// The charset parameter sets the encoding attribute of the XML declaration, if
		// not null and if using the XML model.

		return new MarkupWriterImpl(model, contentType.getCharset());
	}

	public MarkupWriter newMarkupWriter(String pageName) {
		Page page = cache.get(pageName);
```

```
                ContentType contentType = analyzer.findContentType(page);

                return newMarkupWriter(contentType);
        }

}
```

Then add to your AppModule

```
        public static void contributeAlias(
                        Configuration<AliasContribution<MarkupWriterFactory>> configuration,
PageContentTypeAnalyzer analyzer,
                        RequestPageCache cache) {

                configuration.add(AliasContribution.create(MarkupWriterFactory.class,
                                new XhtmlMarkupWriterFactory(analyzer, cache)));
        }
```

In response to the *"NOTE : This currently isn't working as overriding the service is failing"*:

I had the problem of a recursive dependency on Alias service... Though, I was able to get it working by injecting the services explicitly using the @InjectService annotation in the following way.

```
    public static void contributeAlias(
                final Configuration<AliasContribution<MarkupWriterFactory>> configuration,
                @InjectService("PageContentTypeAnalyzer") final PageContentTypeAnalyzer analyzer,
                @InjectService("RequestPageCache") final RequestPageCache cache) {

                    configuration.add(AliasContribution.create(MarkupWriterFactory.class, new
XhtmlMarkupWriterFactory(analyzer, cache)));
    }
```