FriendlyUrls

NOTE: This is outdated information that applies only to Tapestry 3 and 4. Tapestry 5 includes friendly URLs by default.

Friendly URLs

As we all know, Tapestry uses a single servlet to generate all pages of a web application.

Unfortunately, most web technologies and users still conceptualize web applications as files on a file system. This can be problematic:

- J2EE declarative security relies on url mappings. Since Tapestry URLs are based on a single servlet, the web developer is left with 2 options:
    1. Secure every page using the same set of declared security constraints.
    2. Implement security constraints within the application itself.
- Internet search engines do not index Tapestry-based web applications well, if at all. This one is important to the marketing folks since search engines are vital to promoting public web sites.
- Many internet users expect pages within a web site to be named like files (e.g. index.html, mypage.jsp) and organized into directories. I have too often heard the complaint that Tapestry URLs are *ugly* and less user-friendly.

While converting my company's web site (http://www.identitytheft911.com) from JSP to Tapestry, I attempted to address these by developing a simple scheme for using friendly URLs within the confines of Tapestry 3.0.
In short, it works by overriding all of Tapestry's built-in engine services to use a new ILink implementation. I'm sure my implementation isn't the most efficient one, but perhaps it can serve as a starting point for an ongoing discussion.
Here are the gory details...

Generating Friendly URLs

I chose the following URL pattern to represent Tapestry services *with* an associated page:

/*page-name*.htm

/*page-name*.reset

I chose the following URL pattern to represent Tapestry services *without* an associated page:

/*service-name*.do

Since the file extensions are arbitrary and should therefore be customizable, I define them as properties inside the application specification:

```
<application name="MyApplication" engine-class="com.mycompany.BaseEngine">
    <property name="org.apache.tapestry.page-url-suffix" value=".htm"></property>
    <property name="org.apache.tapestry.reset-url-suffix" value=".reset"></property>
    <property name="org.apache.tapestry.service-url-suffix" value=".do"></property>

    <!-- ... -->

</application>
```

To simulate the directories, I use one or more "/" characters within the page/service name.

e.g.

```
<page name="path/mylisting" specification-path="MyListing.page"/>
<page name="path/mydetail" specification-path="MyDetail.page"/>
```

Generated URL = `/path/mylisting.htm`

Generated URL = `/path/mylisting.reset`

Generated URL = `/path/mydetail.htm`

Generated URL = `/path/mydetail.reset`

e.g.

```
<service name="path/myservice" class-name="com.mycompany.MyService"/>
```

Generated URL = `/path/myservice.do`

Now... Tapestry doesn't actually allow us to use "/"s in the page/service name since this would cause collisions when generating the service context parameter. To circumvent this, I do not squeeze the service context parameters into a single URL parameter, but use multiple parameters with the same name - similar to the way service parameters (i.e. sp) already work.

Unfortunately, this requires the following change to the `org.apache.tapestry.parse.SpecificationParser` class itself:

```
Index: framework/src/org/apache/tapestry/parse/SpecificationParser.java
===================================================================
RCS file: /home/cvspublic/jakarta-tapestry/framework/src/org/apache/tapestry/parse/SpecificationParser.java,v
retrieving revision 1.19
diff -u -r1.19 SpecificationParser.java
--- framework/src/org/apache/tapestry/parse/SpecificationParser.java    19 Feb 2004 17:37:41 -0000    1.19
+++ framework/src/org/apache/tapestry/parse/SpecificationParser.java    12 Aug 2004 21:03:02 -0000
@@ -103,6 +103,16 @@
     public static final String EXTENDED_PROPERTY_NAME_PATTERN = "^_?[a-zA-Z](\\w|-|\\.)*$";

     /**
+     *  Like extended property name, but allows forward slashes in the name as
+     *  well.
+     *
+     *  @since 2.2
+     *
+     **/
+
+    public static final String PATH_PROPERTY_NAME_PATTERN = "^_?[a-zA-Z](\\w|-|\\.|/)*$";
+
+    /**
      *  Perl5 pattern that parameter names must conform to.
      *  Letter, followed by letter, number or underscore.
      *
@@ -132,7 +142,7 @@
      *
      **/

-    public static final String PAGE_NAME_PATTERN = EXTENDED_PROPERTY_NAME_PATTERN;
+    public static final String PAGE_NAME_PATTERN = PATH_PROPERTY_NAME_PATTERN;

     /**
      *  Perl5 pattern for component alias.
@@ -186,7 +196,7 @@
      *
      **/

-    public static final String SERVICE_NAME_PATTERN = EXTENDED_PROPERTY_NAME_PATTERN;
+    public static final String SERVICE_NAME_PATTERN = PATH_PROPERTY_NAME_PATTERN;

     /**
      *  Perl5 pattern for library ids.  Letter followed
```

In order for nested directories to work properly with Tables, a modification needs to be made to `org.apache.tapestry.util.io.ComponentAddressAdaptor`:

```
public Object unsqueeze(DataSqueezer squeezer, String string) throws IOException
    {
-       int separator = string.indexOf(SEPARATOR);
+        //FriendlyURL patch
+        int separator = string.lastIndexOf(SEPARATOR);

        if (separator < 0)
            throw new IOException(Tapestry.getMessage("ComponentAddressAdaptor.no-separator"));

        String pageName = string.substring(1, separator);
        String idPath = string.substring(separator + 1);
        if (idPath.equals(""))
                idPath = null;

        return new ComponentAddress(pageName, idPath);
    }
```

Here are some examples of standard Tapestry URLs side by side with their friendly URL counterparts:

Service	Standard Tapestry URL	Friendly URL
Page	{{ http://localhost/app?service=page/mylisting }}	{{ http://localhost/path/mylisting.htm }}
External	{{ http://localhost/app?service=external/mydetail&sp=21 }}	{{ http://localhost/path/mydetail.htm?sp=21 }}
Direct	{{ http://localhost/app?service=direct/1/mydetail/$DirectLink&sp=21 }}	{{ http://localhost/path/mydetail.htm?service=direct&service=1&service=$DirectLink&sp=21 }}
Action	{{ http://localhost/app?service=action/1/mydetail/10 }}	{{ http://localhost/path/mydetail.htm?service=action&service=1&service=10 }}
Home	{{ http://localhost/app?service=home }}	{{ http://localhost/home.do }}
Restart	{{ http://localhost/app?service=restart }}	{{ http://localhost/restart.do }}
Reset	{{ http://localhost/app?service=reset/mydetail }}	{{ http://localhost/path/mydetail.reset }}
Custom	{{ http://localhost/app?service=myservice&sp=21 }}	{{ http://localhost/path/myservice.do?sp=21 }}

After deciding on the URL scheme, the next step was to create an implementation of ILink that can generate the new URLs:

```
public class Link implements ILink
{
    protected IRequestCycle requestCycle;
    private String servletPath;
    private String[] serviceContext;
    private String[] parameters;

    public Link(IRequestCycle requestCycle, String servletPath, String[] serviceContext, Object[] parameters)
```

```
    {
        this.requestCycle = requestCycle;
        this.servletPath = servletPath;
        this.serviceContext = serviceContext;

        if (parameters != null)
        {
            DataSqueezer squeezer = requestCycle.getEngine().getDataSqueezer();

            try
            {
                this.parameters = squeezer.squeeze(parameters);
            }
            catch (IOException e)
            {
                throw new ApplicationRuntimeException(e);
            }
        }
    }

    /**
     * @see org.apache.tapestry.engine.ILink#getURL()
     */
    public String getURL()
    {
        return getURL(null, true);
    }

    /**
     * @see org.apache.tapestry.engine.ILink#getURL(java.lang.String, boolean)
     */
    public String getURL(String anchor, boolean includeParameters)
    {
        return constructURL(new StringBuffer(), anchor, includeParameters);
    }

    /**
     * @see org.apache.tapestry.engine.ILink#getAbsoluteURL()
     */
    public String getAbsoluteURL()
    {
        return getAbsoluteURL(null, null, 0, null, true);
    }

    /**
     * @see org.apache.tapestry.engine.ILink#getAbsoluteURL(java.lang.String, java.lang.String, int, java.lang.
String, boolean)
     */
    public String getAbsoluteURL(String scheme, String server, int port, String anchor, boolean
includeParameters)
    {
        RequestContext context = this.requestCycle.getRequestContext();
        StringBuffer buffer = new StringBuffer();

        buffer.append((scheme != null) ? scheme : context.getScheme());
        buffer.append("://");
        buffer.append((server != null) ? server : context.getServerName());
        buffer.append(':');
        buffer.append((port != 0) ? port : context.getServerPort());

        return constructURL(buffer, anchor, includeParameters);
    }

    private String constructURL(StringBuffer buffer, String anchor, boolean includeParameters)
    {
        RequestContext context = this.requestCycle.getRequestContext();

        buffer.append(context.getRequest().getContextPath()).append('/').append(this.servletPath);

        if (includeParameters)
        {
```

```
        String encoding = this.requestCycle.getEngine().getOutputEncoding();

        buffer.append('?');

        try
        {
            if (this.serviceContext != null)
            {
                for (int i = 0; i < this.serviceContext.length; i++)
                {
                    buffer.append(Tapestry.SERVICE_QUERY_PARAMETER_NAME);
                    buffer.append('=');
                    buffer.append(URLEncoder.encode(this.serviceContext[i], encoding));
                    buffer.append('&');
                }
            }

            if (this.parameters != null)
            {
                for (int i = 0; i < this.parameters.length; i++)
                {
                    buffer.append(Tapestry.PARAMETERS_QUERY_PARAMETER_NAME);
                    buffer.append('=');
                    buffer.append(URLEncoder.encode(this.parameters[i], encoding));
                    buffer.append('&');
                }
            }

            // Delete trailing & or ?
            buffer.deleteCharAt(buffer.length() - 1);
        }
        catch (UnsupportedEncodingException e)
        {
            throw new ApplicationRuntimeException(e);
        }
    }

    if (anchor != null)
    {
        buffer.append('#').append(anchor);
    }

    return this.requestCycle.encodeURL(buffer.toString());
}

/**
 * @see org.apache.tapestry.engine.ILink#getParameterNames()
 */
public String[] getParameterNames()
{
    List parameterList = new ArrayList(2);

    if ((this.serviceContext != null) && (this.serviceContext.length > 0))
    {
        parameterList.add(Tapestry.SERVICE_QUERY_PARAMETER_NAME);
    }

    if ((this.parameters != null) && (this.parameters.length > 0))
    {
        parameterList.add(Tapestry.PARAMETERS_QUERY_PARAMETER_NAME);
    }

    return (String[]) parameterList.toArray(new String[parameterList.size()]);
}

/**
 * @see org.apache.tapestry.engine.ILink#getParameterValues(java.lang.String)
 */
public String[] getParameterValues(String name)
{
    if (name.equals(Tapestry.PARAMETERS_QUERY_PARAMETER_NAME))
```

```
        {
            return this.parameters;
        }

        if (name.equals(Tapestry.SERVICE_QUERY_PARAMETER_NAME))
        {
            return this.serviceContext;
        }

        throw new IllegalArgumentException(Tapestry.format("EngineServiceLink.unknown-parameter-name", name));
    }
}
```

Next, I overloaded all of the built-in Tapestry services to use the new ILink implementation.

```
<application name="MyApplication" engine-class="com.mycompany.BaseEngine">

    <!-- ... -->

    <service name="home" class="com.mycompany.HomeService"/>
    <service name="restart" class="com.mycompany.RestartService"/>
    <service name="reset" class="com.mycompany.ResetService"/>
    <service name="page" class="com.mycompany.PageService"/>
    <service name="external" class="com.mycompany.ExternalService"/>
    <service name="direct" class="com.mycompany.DirectService"/>
    <service name="action" class="com.mycompany.ActionService"/>
    <service name="asset" class="com.mycompany.AssetService"/>

</application>
```

Page Service

```
public class PageService extends org.apache.tapestry.engine.PageService
{
    /**
     * @see org.apache.tapestry.engine.IEngineService#getLink(org.apache.tapestry.IRequestCycle, org.apache.
tapestry.IComponent, java.lang.Object[])
     */
    public ILink getLink(IRequestCycle requestCycle, IComponent component, Object[] parameters)
    {
        if (Tapestry.size(parameters) != 1)
        {
            throw new IllegalArgumentException(Tapestry.format("service-single-parameter", Tapestry.
PAGE_SERVICE));
        }

        String page = (String) parameters[0];

        String suffix = requestCycle.getEngine().getSpecification().getProperty(BaseEngine.PAGE_URL_SUFFIX);

        return new Link(requestCycle, page + suffix, null, null);
    }

    /**
     * @see org.apache.tapestry.engine.AbstractService#getServiceContext(org.apache.tapestry.request.
RequestContext)
     */
    protected String[] getServiceContext(RequestContext requestContext)
    {
        IApplicationSpecification specification = requestContext.getServlet().getApplicationSpecification();
        String urlSuffix = specification.getProperty(BaseEngine.PAGE_URL_SUFFIX);
        String servletPath = requestContext.getRequest().getServletPath();

        return new String[] { servletPath.substring(1, servletPath.indexOf(urlSuffix)) };
    }
}
```

External Service

```
public class ExternalService extends org.apache.tapestry.engine.ExternalService
{
    /**
     * @see org.apache.tapestry.engine.IEngineService#getLink(org.apache.tapestry.IRequestCycle, org.apache.
tapestry.IComponent, java.lang.Object[])
     */
    public ILink getLink(IRequestCycle requestCycle, IComponent component, Object[] parameters)
    {
        if (parameters == null || parameters.length == 0)
        {
            throw new ApplicationRuntimeException(Tapestry.format("service-requires-parameters", Tapestry.
EXTERNAL_SERVICE));
        }

        String page = (String) parameters[0];

        Object[] pageParameters = new Object[parameters.length - 1];
        System.arraycopy(parameters, 1, pageParameters, 0, parameters.length - 1);

        String suffix = requestCycle.getEngine().getSpecification().getProperty(BaseEngine.PAGE_URL_SUFFIX);

        return new Link(requestCycle, page + suffix, null, pageParameters);
    }

    /**
     * @see org.apache.tapestry.engine.AbstractService#getServiceContext(org.apache.tapestry.request.
RequestContext)
     */
    protected String[] getServiceContext(RequestContext requestContext)
    {
        IApplicationSpecification specification = requestContext.getServlet().getApplicationSpecification();
        String urlSuffix = specification.getProperty(BaseEngine.PAGE_URL_SUFFIX);
        String servletPath = requestContext.getRequest().getServletPath();

        return new String[] { servletPath.substring(1, servletPath.indexOf(urlSuffix)) };
    }
}
```

Direct Service

```
public class DirectService extends org.apache.tapestry.engine.DirectService
{
    private static final String STATEFUL_ON = "1";
    private static final String STATEFUL_OFF = "0";

    /**
     * @see org.apache.tapestry.engine.IEngineService#getLink(org.apache.tapestry.IRequestCycle, org.apache.
tapestry.IComponent, java.lang.Object[])
     */
    public ILink getLink(IRequestCycle requestCycle, IComponent component, Object[] parameters)
    {
        IPage renderPage = requestCycle.getPage();
        IPage componentPage = component.getPage();

        boolean complex = renderPage != componentPage;

        String stateful = requestCycle.getEngine().isStateful() ? STATEFUL_ON : STATEFUL_OFF;

        String[] serviceContext = new String[complex ? 4 : 3];

        int i = 0;

        serviceContext[i++] = Tapestry.DIRECT_SERVICE;
        serviceContext[i++] = stateful;

        if (complex)
        {
            serviceContext[i++] = componentPage.getPageName();
        }

        serviceContext[i++] = component.getIdPath();

        String suffix = requestCycle.getEngine().getSpecification().getProperty(BaseEngine.PAGE_URL_SUFFIX);

        return new Link(requestCycle, renderPage.getPageName() + suffix, serviceContext, parameters);
    }

    /**
     * @see org.apache.tapestry.engine.AbstractService#getServiceContext(org.apache.tapestry.request.
RequestContext)
     */
    protected String[] getServiceContext(RequestContext requestContext)
    {
        IApplicationSpecification specification = requestContext.getServlet().getApplicationSpecification();
        String urlSuffix = specification.getProperty(BaseEngine.PAGE_URL_SUFFIX);
        String[] serviceContext = requestContext.getParameters(Tapestry.SERVICE_QUERY_PARAMETER_NAME);

        String servletPath = requestContext.getRequest().getServletPath();

        // Remove service name from service context
        serviceContext[0] = serviceContext[1];
        // Insert page name as second service context element
        serviceContext[1] = servletPath.substring(1, servletPath.indexOf(urlSuffix));

        return serviceContext;
    }
}
```

Action Service

```
public class ActionService extends org.apache.tapestry.engine.ActionService
{
    private static final String STATEFUL_ON = "1";
    private static final String STATEFUL_OFF = "0";

    /**
     * @see org.apache.tapestry.engine.IEngineService#getLink(org.apache.tapestry.IRequestCycle, org.apache.
tapestry.IComponent, java.lang.Object[])
     */
    public ILink getLink(IRequestCycle requestCycle, IComponent component, Object[] parameters)
    {
        if ((parameters == null) || (parameters.length != 1))
        {
            throw new IllegalArgumentException(Tapestry.format("service-single-parameter", Tapestry.
ACTION_SERVICE));
        }

        String stateful = requestCycle.getEngine().isStateful() ? STATEFUL_ON : STATEFUL_OFF;
        IPage renderPage = requestCycle.getPage();
        IPage componentPage = component.getPage();

        boolean complex = (componentPage != renderPage);

        String[] serviceContext = new String[complex ? 5 : 4];

        int i = 0;

        serviceContext[i++] = Tapestry.ACTION_SERVICE;
        serviceContext[i++] = stateful;
        serviceContext[i++] = (String) parameters[0];

        if (complex)
        {
            serviceContext[i++] = componentPage.getPageName();
        }

        serviceContext[i++] = component.getIdPath();

        String suffix = requestCycle.getEngine().getSpecification().getProperty(BaseEngine.PAGE_URL_SUFFIX);

        return new Link(requestCycle, renderPage.getPageName() + suffix, serviceContext, null);
    }

    /**
     * @see org.apache.tapestry.engine.AbstractService#getServiceContext(org.apache.tapestry.request.
RequestContext)
     */
    protected String[] getServiceContext(RequestContext requestContext)
    {
        IApplicationSpecification specification = requestContext.getServlet().getApplicationSpecification();
        String urlSuffix = specification.getProperty(BaseEngine.PAGE_URL_SUFFIX);
        String[] serviceContext = requestContext.getParameters(Tapestry.SERVICE_QUERY_PARAMETER_NAME);
        String servletPath = requestContext.getRequest().getServletPath();

        // Remove service name from service context
        serviceContext[0] = serviceContext[1];
        // Insert page name as second service context element
        serviceContext[1] = servletPath.substring(1, servletPath.indexOf(urlSuffix));

        return serviceContext;
    }
}
```

Home Service

```
public class HomeService extends org.apache.tapestry.engine.HomeService
{
    public ILink getLink(IRequestCycle requestCycle, IComponent component, Object[] parameters)
    {
        String suffix = requestCycle.getEngine().getSpecification().getProperty(BaseEngine.SERVICE_URL_SUFFIX);

        return new Link(requestCycle, Tapestry.HOME_SERVICE + suffix, null, null);
    }
}
```

Asset Service

```
public class AssetService extends org.apache.tapestry.asset.AssetService
{
    /**
     * @see org.apache.tapestry.engine.IEngineService#getLink(org.apache.tapestry.IRequestCycle, org.apache.
tapestry.IComponent, java.lang.Object[])
     */
    public ILink getLink(IRequestCycle requestCycle, IComponent component, Object[] parameters)
    {
        if (Tapestry.size(parameters) != 2)
        {
            throw new ApplicationRuntimeException(Tapestry.format("service-incorrect-parameter-count", Tapestry.
ASSET_SERVICE, new Integer(2)));
        }

        String suffix = requestCycle.getEngine().getSpecification().getProperty(BaseEngine.SERVICE_URL_SUFFIX);

        return new Link(requestCycle, Tapestry.ASSET_SERVICE + suffix, null, parameters);
    }
}
```

Restart Service

```
public class RestartService extends org.apache.tapestry.engine.RestartService
{
    /**
     * @see org.apache.tapestry.engine.IEngineService#getLink(org.apache.tapestry.IRequestCycle, org.apache.
tapestry.IComponent, java.lang.Object[])
     */
    public ILink getLink(IRequestCycle requestCycle, IComponent component, Object[] parameters)
    {
        String suffix = requestCycle.getEngine().getSpecification().getProperty(BaseEngine.SERVICE_URL_SUFFIX);

        return new Link(requestCycle, Tapestry.RESTART_SERVICE + suffix, null, null);
    }
}
```

Reset Service

```
public class ResetService extends org.apache.tapestry.engine.ResetService
{
    /**
     * @see org.apache.tapestry.engine.IEngineService#getLink(org.apache.tapestry.IRequestCycle, org.apache.
tapestry.IComponent, java.lang.Object[])
     */
    public ILink getLink(IRequestCycle requestCycle, IComponent component, Object[] parameters)
    {
        if (Tapestry.size(parameters) != 1)
        {
            throw new IllegalArgumentException(Tapestry.format("service-single-parameter", Tapestry.
RESET_SERVICE));
        }

        String page = (String) parameters[0];

        String suffix = requestCycle.getEngine().getSpecification().getProperty(BaseEngine.RESET_URL_SUFFIX);

        return new Link(requestCycle, page + suffix, null, null);
    }

    /**
     * @see org.apache.tapestry.engine.AbstractService#getServiceContext(org.apache.tapestry.request.
RequestContext)
     */
    protected String[] getServiceContext(RequestContext requestContext)
    {
        IApplicationSpecification specification = requestContext.getServlet().getApplicationSpecification();
        String urlSuffix = specification.getProperty(BaseEngine.RESET_URL_SUFFIX);
        String servletPath = requestContext.getRequest().getServletPath();

        return new String[] { servletPath.substring(1, servletPath.indexOf(urlSuffix)) };
    }
}
```

Custom Services

Custom services that utilize the service context should extend the following abstract class:

```
public abstract class AbstractService extends org.apache.tapestry.engine.AbstractService
{
    protected String[] getServiceContext(RequestContext requestContext)
    {
        return requestContext.getParameters(Tapestry.SERVICE_QUERY_PARAMETER_NAME);
    }
}
```

Interpreting Friendly URLs

Each new service implementation already knows how to both generate and interpret the new URLs. Next I needed to override the default Tapestry engine to properly determine which engine service to use with a given URL.

```
public class BaseEngine extends org.apache.tapestry.engine.BaseEngine
{
    public static final String SERVICE_URL_SUFFIX = "org.apache.tapestry.service-url-suffix";
    public static final String PAGE_URL_SUFFIX = "org.apache.tapestry.page-url-suffix";
    public static final String RESET_URL_SUFFIX = "org.apache.tapestry.reset-url-suffix";

    /**
     * @see org.apache.tapestry.engine.AbstractEngine#extractServiceName(org.apache.tapestry.request.
RequestContext)
     */
    protected String extractServiceName(RequestContext requestContext)
    {
        String[] serviceContext = requestContext.getParameters(Tapestry.SERVICE_QUERY_PARAMETER_NAME);

        if (requestContext.getRequest().getParameter(Tapestry.TAG_SUPPORT_SERVICE_ATTRIBUTE) != null)
        {
            return Tapestry.TAGSUPPORT_SERVICE;
        }

        String servletPath = requestContext.getRequest().getServletPath();
        IApplicationSpecification specification = this.getSpecification();
        String pageUrlSuffix = specification.getProperty(PAGE_URL_SUFFIX);

        if (servletPath.endsWith(pageUrlSuffix))
        {
            if ((serviceContext == null) || (serviceContext.length == 0))
            {
                if (requestContext.getRequest().getParameter(Tapestry.PARAMETERS_QUERY_PARAMETER_NAME) == null)
                {
                    return Tapestry.PAGE_SERVICE;
                }

                return Tapestry.EXTERNAL_SERVICE;
            }

            return serviceContext[0];
        }

        String serviceUrlSuffix = specification.getProperty(SERVICE_URL_SUFFIX);

        if (servletPath.endsWith(serviceUrlSuffix))
        {
            return servletPath.substring(1, servletPath.indexOf(serviceUrlSuffix));
        }

        String resetUrlSuffix = specification.getProperty(RESET_URL_SUFFIX);

        if (servletPath.endsWith(resetUrlSuffix))
        {
            return Tapestry.RESET_SERVICE;
        }

        throw new ApplicationRuntimeException("Invalid request: " + requestContext.getRequest().
getRequestURL());
    }
}
```

Using Friendly URLs

To use the new URLs, I just to map the Tapestry servlet to the url patterns defined in my application specification.

```
<web-app>

    <!-- ... -->

    <servlet>
        <servlet-name>MyApplication</servlet-name>
        <servlet-class>org.apache.tapestry.ApplicationServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>MyApplication</servlet-name>
        <url-pattern>*.htm</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>MyApplication</servlet-name>
        <url-pattern>*.reset</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>MyApplication</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>

    <!-- ... -->

</web-app>
```

Done !!!

Integrating J2EE Declarative Security

Friendly URLs allow us to access a web application using a virtual directory structure from which to secure sections of a web site using standard J2EE declarative security.

e.g.

```
<web-app>

    <!-- ... -->

    <!-- Sample security constraint for an entire directory -->
    <!-- User will be must belong to "privileged" role and SSL is required. -->
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Secured pages</web-resource-name>
            <url-pattern>/path/*</url-pattern>
            <auth-constraint>
                <role-name>privileged</role-name>
            </auth-constraint>
            <user-data-constraint>
                <transport-guarantee>CONFIDENTIAL</transport-guarantee>
            </user-data-constaint>
        </web-resource-collection>
    </security-constraint>

    <!-- ... -->

    <!-- Sample JAAS login configuration using "users" realm. -->
    <!-- Login page should implement IExternalPage to handle login failed condition. -->
    <login-config>
        <auth-method>FORM</auth-method>
        <realm-name>users</realm-name>
        <form-login-config>
            <form-login-page>/login.htm</form-login-page>
            <form-error-page>/login.htm?sp=T</form-error-page>
        </form-login-config>
    </login-config>

    <!-- ... -->

</web-app>
```

Tapestry 4.0

HowardLewisShip: I've implemented something similar to this for Tapestry 4.0, that results in prettier URLs similar to what Paul's accomplished with his patch to Tapestry 3.0. However, Paul has some good ideas beyond what I've accomplished so far, such as mapping /app?service=foo to /foo.do ... and these, too, will appear in 4.0. There's a couple of new levels of abstraction between the service (which has information needed to be encoded into the URL) and the actual link (the combination of URL and query parameters), with new objects, ServiceEncoders, responsible for converting query parameter information into path information, and vice versa. In other words, slick, extensible and adaptable. For example, the page and direct services can be utilized with two modifications:

web.xml:

```
  <servlet-mapping>
    <servlet-name>workbench</servlet-name>
    <url-pattern>*.page</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>workbench</servlet-name>
    <url-pattern>*.direct</url-pattern>
  </servlet-mapping>
```

hivemodule.xml:

```
  <contribution configuration-id="tapestry.url.ServiceEncoders">
    <path-encoder id="direct" extension="direct" service="direct"/>
    <path-encoder id="page" extension="page" service="page"/>
  </contribution>
```

At this point, the application will generate URLs like /context/Home.page (equivalent to /context/app?service=page/Home) and /context/Home.direct?component=foo (equivalent to /app?service=direct/0/Home/foo).