

The examples used so far have focused on separation of content and style in static pages; that is, HTML pages with predefined text and images. This type of document lends itself to an XML framework perfectly due to its inherent structure, stability, and standardization. A publication such as the **Gateways Guide** adheres to a hierarchy with chapters and sections and paragraphs. For that type of Web site, XML is a great match.

However, few Web sites contain only that type of content. Most Web sites today have some kind of dynamic and interactive content involving external files or databases and transactions. Even simple forms involve some level of interaction.

So now we have to consider how to handle this situation where content is not restricted to text in an XML file, but includes data in a relational database, and content selection involves not just XSL transformations, but programming logic. Can we incorporate different data structures and processing algorithms into our XML framework?

As Figure 12 illustrates, Cocoon does offer a structure for handling this situation by not only separating content from style, but from logic as well. Within Cocoon, the separation of logic is handled in a special type of file called an "XSP" file.

### Figure 12. Logic, Content, and Style Separation in Cocoon Publishing Framework

This structure allows you to manage the process efficiently by segregating duties. An application programmer, for example, can work on the processing logic without worrying about or, more importantly, interfering with the style or content elements. In fact, it enables parallel development because logic, content, and style are worked on simultaneously and independently and then integrated in the final output.

Figure 13 shows an XSP file which contains only logic (in this case, Java and SQL code), no content or style information. The XSP file is connected to its XML and XSL files in exactly the same way that the XML file was connected to the XSL file: through processing instructions at the start of the file that link it to the other files. In fact, an XSP file has a .xsl file extension so the XML file links to the XSP file which links to the XSL stylesheets.

**Figure 13. XSL File Containing Logic Code (Cocoon XSP File, simplified for example)**

```
<?xml
version="1.0"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsp="http://www.apache.org/1999/XSP/Core" xmlns:esql="http://apache.org/cocoon/SQL/v2">
<!--COMMENT Code example simplified for example --> <xsl:template match="page"> <xsl:processing-instruction
name="xml-stylesheet" href="registershow.xsl" type="text/xsl" /> </xsl:processing-instruction> <!-- COMMENT
Defines the query string for the database --> <xsp:logic>&lt;![CDATA[ String username,programe,password,t;
private String assemblyQuery() { t="select * from XXXXXXXX where YYYYYY='"+username+"'"; return t; } ]&gt;
</xsp:logic> <!--COMMENT verifies valid entry, redirects or allows in--> <xsp:logic>&lt;![CDATA[ if
(request.getParameter("username")==null) { response.sendRedirect("signon.xml"); }&gt; else {&lt;![CDATA[
username=request.getParameter("username").replace("&quot;",""); }&gt; <!-- COMMENT Database connection and
query --> <esql:connection> <esql:driver>org.gjt.mm.mysql.Driver</esql:driver>
<esql:dburl>jdbc:mysql://XXXXXXXX/XXXXXXXX</esql:dburl> <esql:username>XXXXXX</esql:username>
<esql:password>XXXXXX</esql:password> <esql:execute-query>
<esql:query><xsp:expr>assemblyQuery()</xsp:expr></esql:query> <esql:results> <esql:row-results> <user>
<username><esql:get-string column="username"/></username> <programe><esql:get-string
column="programe"/></programe> </user> </esql:row-results> </esql:results> <esql:no-results> <norecords/>
</esql:no-results> <esql:error-results> <dbproblem/> </esql:error-results> </esql:execute-query>
</esql:connection> <!--COMMENT Closes out big else condition from the redirect if--> } </xsp:logic> <!--
COMMENT Closes out the template and the stylesheet --> </xsl:template> </xsl:stylesheet>
```

This content/logic/style structure enabled us to develop an interactive component of the **Gateways Guide** in which users using online forms could register and create online worksheets that employ the principles and practices described in the guide. Most importantly, the logic for this system (involving a MySQL database, SQL, Java, and JavaScript coding) was developed independently by application programmers using XSP files that were integrated into the existing XML/XSL framework established for the guide. The result as seen in Figure 14 was an interactive, dynamic application that not only had the same look and feel as the **Gateways Guide** but actually used the same XML source documents and XSL stylesheets. In other words, content and style was not duplicated or re-created for this new application; it was simply re-used.

### Figure 14. Sample Dynamic HTML Page

