XML -- table of contents

Development

Administration

4.2.3: Incorporating XML

IBM WebSphere Application Server provides XML Document Structure Services, which consist of a document parser, a document validator, and a document generator for server-side XML processing.

See article 4.1.1.2 for all of the details about XML supportin the product.If you are just becoming familiar with XML, start with article 0.33, a primer on XML concepts, vocabulary, and uses.

Other related information provides guidance on the following topics:

- Structure -- defining and obeying the syntax for an XML tag set
- Content -- determining the mechanism for filling XML tags with data
- Presentation -- determining the mechanism for formatting and displaying XML content

In addition, some special topics are covered, including DOM objects andmanipulation of Channel Definition Format (CDF) files as illustrated bythe SiteOutliner example.

When you install IBM WebSphere Application Server, the core XML APIs are automatically added to the appropriate class path, enabling you to serve static XML documents as soon as the product is installed.

To serve XML documents that are dynamically generated, use the core APIs to develop servlets or Web applications that generate XML documents (for example, the applications might read the document content from a database) and then deploy those components on your application server.

4.2.3.2: Specifying XML document structure

The structure of an XML document is governed by syntax rules for its tag set. Those tags are defined formally in an XML-based grammar, such as a Document Type Definition (DTD). At the time of this publication, DTD is the most widely-implemented grammar. Therefore, this article discusses options for using DTDs.

Options for XML document structure include:

Do not use a DTD. Not using a DTD enables maximum flexibility in evolving XML document structure, but this flexibility limits the ability to share the documents among users and applications. An XML document can be parsed without a DTD. If the parser does not find an inline DTD or a reference to an external DTD, the parser proceeds using the actual structure of the tags within the document as an implied DTD. The processor evaluates the document to determine whether it meets the rules for well-formedness.

Use a public DTD. Various industry and other interest groups are developing DTDs for categories of documents, such as chemical data and archival documents. Many of these DTDs are in the public domain and are available over the Internet. Using an industry standard DTD maximizes sharing documents among applications that act on the grammar. If the standard DTD does not accomodate the schema the applications need, flexibility is limited.

Several industry and interest groups have developed and proposed DTD grammars for the types of documents they produce and exchange. To make it easier for you to use those grammars, local copies are installed with the product. Use the grammars as examples in developing your own grammars as well as for creating and validating XML documents of those types. The library is located atproduct_installation_root\web\xml\grammar\

Develop a DTD. If none of the public DTDs meet an enterprise's needs and enforcing document validity is a requirement, the XML implementers can develop a DTD. Developing a DTD requires careful analysis of the information (data) that the documents will contain.

For DTD updates,visit the XML Industry Portal.For details about the DTD specifications and sample DTDs, refer toIBM's developerWorkssite for education and other DTD resources.

4.2.3.3: Providing XML document content

The content of an XML document is the actual data that appears within the document tags. XML implementers must determine the source and the mechanism for putting the data into the document tags. The options include:

Static content. XML document content is created and stored on the Web server as static files. The XML document author composes the document to include valid XML tags and data in a manner similar to how HTML authors compose static HTML files. This approach works well for data that is not expected to change or that will change infrequently. Examples are journal articles, glossaries, and literature.

Dynamically generated content. XML document content can be dynamically generated from a database and user input. In this scenario, XML-capable servlets, Java beans, and even inline Java code within a JavaServer Page (JSP) file can be used to generate the XML document content.

A hybrid of static and dynamically generated content. This scenario involves a prudent combination of static and dynamically generated content.

You can also use XSL to add to or remove information from existing XML content.For details, see the Related information.

4.2.3.4: Rendering XML documents

Options for presenting XML documents include:

Present the XML document in an XML-enabled browser. An XML-enabled browser can parse a document, apply its XSL stylesheet, and present the document to the user. Searching and enabling users to modify an XML document are other possible functions of XML-enabled browsers.

Present the XML document to a browser that converts XML to HTML. Until XML-enabled browsers are readily available, presenting XML documents to users will involve converting the XML document to HTML. That conversion can be handled by conversion-capable browsers. Another option is to use JavaScript or ActiveX controls embedded within the XML document. Microsoft Internet Explorer Version 5 is an XML-to-HTML converter. HTML is not the only format to which XML documents can be converted. It's just the easiest to implement given the commerically available browsers and user agents.

Send an HTML file to the browser. If the users do not have XML-capable browsers, the XML document must be converted at the server before being transmitted to the browser. The server-side XML application that handles the conversion could also determine the capability of the browser before converting the document to HTML, to avoid unnecessary processing if the browser is XML-capable. The XSL processor included with this product supports such server-side functions.

Using XSL to convert XML documents to other formats

IBM WebSphere Application Server includes the Lotus XSL processor and its open-sourceversion, Xalan, for formatting and converting XML documents. Processing can be done at the server or at the browser, to HTML or to other XML-compliant markup languages.For sample code, see the Xalan documentation.

Use of the XSL processor with the Xerces XML parserrequires a liaison object, as follows:

```
XSLTProcessor processor =    XSLTProcessorFactory.getProcessor(new
com.lotus.xml.xml4j2dom.XML4JLiaison4dom());
```

Converting XML documents at the server

One option for presenting an XML document is for the server to convert the XML document to HTML and return the HTML document to the client. On the server side, this typically requires the creation of a servletto handle the processing of one data stream (the XML document)with another (the XSL document).The output of that process is then forwarded back to the browser.

Server-side processing often requires the passing in of parameters through theXSL processor to customize the output.For an example, see the Xalan documentation.

4.2.3.6: Using DOM to incorporate XML documents into applications

The Document Object Model (DOM) is an API for representing XML and HTML documentsas objects that can be accessed by object-oriented programs, such as business logic, for the purposes of creating, navigating, manipulating, and modifying the documents.

Article 0.33.3 introduces DOM concepts and vocabulary. Article 4.1.1.2 tells youwhere to find the DOM specification and org.w3c.dom package.

Article 4.2.3.6.1 providesa quick reference so that you can jump right into DOM development, referring to thepackage and specification as needed.

4.2.3.6.1: Quick reference to DOM object interfaces

This section highlights a few of the object interfaces. Refer to theDOM Specification for details (see article 4.1.1.2).

Node methods

Node methods include:

Method	Description
hasChildNodes	Returns a boolean to indicate whether a node has children
appendNode	Appends a new child node to the end of the list of children for a parent node
insertBefore	Inserts a child node before the existing child node
removeChild	Removes the specified child node from the node list and returns the node
replaceChild	Replaces the specified child node with the specified new node and returns the new node

Document methods

The Document object represents the entire XML document. Document methods include:

Method	Description
createElement	Creates and returns an Element (tag) of the type specified. If the document will be validated against a DTD, that DTD must contain an Element declaration for the created element.
createTextNode	Creates a Text node that contains the specified string
createComment	Creates a Comment node with the specified content (enclosed within <! - - and - -> tags)
createAttribute	Creates an Attribute node of the specified name. Use the setAttribute method of Element to set the value of the Attribute. If the document will be validated against a DTD, that DTD must contain an Attribute declaration for the created attribute.
createProcessingInstruction	Creates a Processing Instruction with the specified name and data (enclosed within <? and ?> tags). A processing instruction is an instruction to the application (such as an XML document formatter) that receives the XML document.

Element methods

Element node methods include:

Method	Description
getAttribute	Returns the value of the specified attribute or empty string
setAttribute	Adds a new attribute-value pair to the element

removeAttribute	Removes the specified attribute from the element
getElementsByTagName	Returns a list of the element descendants that have the specified tag name

A Text node can be a child of an Element or Attribute node and contains the textual content (character data) for the parent node. If the content does not include markup, all of the content is placed within a single Text node. If the content includes markup, that markup is placed in one or more Text nodes that are siblings of the Text node that contains the non-markup content.

The Text node extends the CharacterData interface, which has methods for setting, getting, replacing, inserting, and making other modifications to a Text node. In addition to those methods, the Text node adds a method:

Method	**Description**
splitText	Splits the Text node at the specified offset. Returns a new Text node, which contains the original content starting at the offset. The original Text node contains the content from the beginning to the offset.

4.2.3.6.2: Manually generating an XML element node

You can manually create any XML element node by using the PseudoNode construct,as follows:

```
new PseudoNode("literal");
```

If a DOM tree contains PseudoNode instances, you can use the tree for printing only. PseudoNode prevents validation against a DTD.

4.2.3.7: SiteOutliner sample

The SiteOutliner servlet illustrates how to use the XML Document Structure Services to generate and view a Channel Definition Format (CDF) file for a target directory on the servlet's Web server. Use Lotus Notes 5 (the Headlines page), Microsoft Internet Explorer 4 Channel Bar, PointCast, Netscape Navigator 4.06, orother CDF-capable viewers to view and manipulate the CDF file.

SiteOutliner is part of the WebSphere Samples Gallery. When you open the gallery,follow the links to SiteOutliner to run it on your local machine.

6.6.0.2: Command line administration

The following command line administrative tools are available for interacting with theWebSphere administrative server.

- Use **wscp** for operational and configuration tasks such as starting, stopping, and configuring application servers and other object types.
- Use **XMLConfig** for configuration tasks. This tool provides a way to work with the administrative domain as represented in WebSphere XML.

6.6.0.2.1: XMLConfig command line interface for XML configuration

Use the *XMLConfig* tool to import and export configuration data to and from the WebSphere Application Server administration repository. This XML-based approach complements the administration you can perform through the WebSphere administrative console.

You may use this tool to perform multiple changes to the WebSphere Application Server administration repository at a single time without having to go through the repetitive routines on the Administration Console. You may also use this tool to extract the repository information from one server and import it onto a cloned server.

The XMLConfig tool provides three fundamental features:

- **full export**

  Generates an XML document describing the configuration of the entire administrative domain. In effect, the full export takes a "snapshot" of the administrative repository contents. Click here for an example of a full export, including a discussion of the various parts of the resulting XML file.

- **partial export**

  Provides an XML document specifying administrative objects to export from the administrative domain. The objects and their children are exported to an XML document that you specify. Click here for an example of a partial export.

- **import**

  Imports a newly created XML document or a document you previously exported and modified. In the XML document, you can add, modify, or remove administrative objects such as servlets and virtual hosts. Your XML document can replace the administrative repository contents partially or entirely.

6.6.0.2.1.1: XMLConfig - Command syntax

This section describes the command line syntax for the XMLConfig tool.

Because setting the classpath appropriately is vital to the tool's success, WebSphere Application Server Version 3.5 contains an XMLConfig.bat (Windows NT) or XMLConfig.sh (*IX) file for starting the tool. The file is located in the bin directory of the product installation root, uses the com.ibm.websphere.xmlconfig.XMLConfig java class, and has the following command line syntax:

```
        { ( -import <xml data file> ) ||        [ ( -export <xml output file> [-partial <xml data
file>] ) }        -adminNodeName <primary node name>        [ -nameServiceHost <host name> [
-nameServicePort <port number> ]]        [-traceString <trace spec> [-traceFile <file name>]]
[-substitute <"key1=value1[;key2=value2;[...]]">]}        In input xml file, the key(s) should
appear as $key$ for substitution.
```

The arguments include:

-adminNodeName

Required argument specifies the node containing the administrative server to which you are connecting. The argument value must match the node name given in the topology tree on the Topology tab of the WebSphere Administrative Console.

-import || -export || -export -partial

Required argument specifies the operation to perform -- an import or export. Unless you also specify the parameter -partial, the export will be treated as a full export.

-nameServiceHost, -nameServicePort

Optional arguments specify the hostname of the machine containing the name service, and the port by which to communicate with the name service. The default value of -nameServicePort is 900.

-traceString

Optional argument specifies the WebSphere Application Server internal code to trace.

For more information, see the traceString section of the trace help.

-substitute

Optional argument specifies the variables to be substituted. For example:

```
-substitute "NODE_NAME=admin_node;APP_SERVER=default_server"
```

This argument substitutes any occurrence of $NODE_NAME$ with `admin_node` and any occurrence of APP_SERVER with `default_server` that is found in the input XML file.

If the substitution string contains semicolons,use `$semiColon$` to separate it from the ";" delimiter. On UNIX platforms,be sure to add an escape character to each dollar sign ($) within the substitution string(for example, `\$semiColon\$`).

The following examples demonstrate correct syntax. 'Node1' is the name by which the node containing the administrative server is administered.

Import operation:

```
XMLConfig -adminNodeName Node1 -import import.xml
```

Full export operation:

```
XMLConfig -adminNodeName Node1 -export export.xml
```

Partial export operation:

```
XMLConfig -adminNodeName Node1 -export export.xml -partial imput.xml
```

6.6.0.2.1.1.1: XMLConfig - Example of a full export

The following example export shows the XML elements for each object type in the WebSphere administrative domain. It is a full export of the administrative repository of a Standard Edition installation featuring the default administrative configuration.

To produce a similar export, you would issue the command:

```
XMLConfig -adminNodeName buccaneer -export export.xml
```

where the hostname of the machine containing the administrative server is "buccaneer" and the output will be directed to a file named export.xml.

When you perform an export, the XML output file will not contain blank lines or gaps. In contrast, the following export has been broken into segments, each of which is briefly discussed.

Also, this example was obtained from a system that used the default configuration, and might vary slightly from actual output due to changes on your particular system. It is recommended you try the export command yourself to see exactly the output that is produced.

```
<?xml version="1.0"?><!DOCTYPE websphere-sa-config SYSTEM
"$server_root$$dsep$bin$dsep$xmlconfig.dtd" >
<websphere-sa-config>
```

The above tags mark the beginning of the export. The following part of the export contains a tag for the default virtual host, default_host, in the administrative domain.

The default host recognizes several MIME types, which are listed as part of a MIME table in the virtual-host tag. In fact, there are so many MIME types that some are omitted from the example code below.

The MIME types are followed by the URIs (also known as Web resources) hosted by default_host:

- Skip ahead
- View object syntax

```
  <virtual-host name="default_host" action="update">    <mime-table>      <mime type="audio/x-wav">
<ext>wav</ext>      </mime>      <mime type="application/x-sv4cpio">        <ext>sv4cpio</ext>
</mime>      <mime type="text/x-ssi-html">        <ext>htmls</ext>        <ext>shtml</ext>
</mime>        ...        ...              <mime type="application/x-netcdf">        <ext>nc</ext>
<ext>cdf</ext>      </mime>      <mime type="video/x-motion-jpeg">        <ext>mjpg</ext>
</mime>    </mime-table>    <alias-list>      <alias>localhost</alias>      <alias>127.0.0.1</alias>
<alias>buccaneer.raleigh.ibm.com</alias>      <alias>buccaneer</alias>
<alias>9.67.127.58</alias>    </alias-list>    <uri name="/" rootURI="/" action="create"/>    <uri
name="/servlet/snoop" rootURI="/" action="create"/>    <uri name="/servlet/snoop2" rootURI="/"
action="create"/>    <uri name="/servlet/hello" rootURI="/" action="create"/>    <uri
name="/ErrorReporter" rootURI="/" action="create"/>    <uri name="/servlet" rootURI="/"
action="create"/>    <uri name="/*.jsp" rootURI="/" action="create"/>    <uri name="/*.jsv"
rootURI="/" action="create"/>    <uri name="/*.jsw" rootURI="/" action="create"/>    <uri
name="/admin" rootURI="/admin" action="create"/>    <uri name="/admin/install" rootURI="/admin"
action="create"/>    <uri name="/admin/*.jsp" rootURI="/admin" action="create"/>    <uri
name="/admin/*.jsv" rootURI="/admin" action="create"/>    <uri name="/admin/*.jsw" rootURI="/admin"
action="create"/>    <uri name="/admin/" rootURI="/admin" action="create"/>    <uri
name="/admin/servlet" rootURI="/admin" action="create"/>    <uri name="/admin/ErrorReporter"
rootURI="/admin" action="create"/>    <uri name="/webapp/examples" rootURI="/webapp/examples"
action="create"/>    <uri name="/webapp/examples/simpleJSP.servlet" rootURI="/webapp/examples"
action="create"/>    <uri name="/webapp/examples/simpleJSP" rootURI="/webapp/examples"
action="create"/>    <uri name="/webapp/examples/ErrorServlet" rootURI="/webapp/examples"
action="create"/>    <uri name="/webapp/examples/ping" rootURI="/webapp/examples" action="create"/>
<uri name="/webapp/examples/SourceCodeViewer" rootURI="/webapp/examples" action="create"/>    <uri
name="/webapp/examples/showCfg" rootURI="/webapp/examples" action="create"/>    <uri
name="/webapp/examples/ShowCfg" rootURI="/webapp/examples" action="create"/>    <uri
name="/webapp/examples/showConfig" rootURI="/webapp/examples" action="create"/>    <uri
name="/webapp/examples/ShowConfig" rootURI="/webapp/examples" action="create"/>    <uri
name="/webapp/examples/HitCount" rootURI="/webapp/examples" action="create"/>    <uri
name="/webapp/examples/verify" rootURI="/webapp/examples" action="create"/>    <uri
name="/webapp/examples/*.jsp" rootURI="/webapp/examples" action="create"/>    <uri
name="/webapp/examples/*.jsv" rootURI="/webapp/examples" action="create"/>    <uri
name="/webapp/examples/*.jsw" rootURI="/webapp/examples" action="create"/>    <uri
name="/webapp/examples/" rootURI="/webapp/examples" action="create"/>    <uri
name="/webapp/examples/HelloPervasive" rootURI="/webapp/examples" action="create"/>    <uri
name="/webapp/examples/StockQuote" rootURI="/webapp/examples" action="create"/>    <uri
name="/webapp/examples/BeenThere" rootURI="/webapp/examples" action="create"/>  </virtual-host>
```

The next section contains the database driver and data source information :

- Skip ahead
- View object syntax

```
  <jdbc-driver name="Admin DB Driver" action="update">
<implementation-class>jdbc.idbDriver</implementation-class>    <url-prefix>jdbc:idb</url-prefix>
<jta-enabled>false</jta-enabled>    <install-info>      <node-name>buccaneer</node-name>
<jdbc-zipfile-location>/usr/WebSphere/AppServer//lib/idb.jar</jdbc-zipfile-location>
</install-info>  </jdbc-driver>  <jdbc-driver name="AccountDB2Driver" action="update">
<implementation-class>com.ibm.db2.jdbc.app.DB2Driver</implementation-class>
<url-prefix>jdbc:db2</url-prefix>    <jta-enabled>false</jta-enabled>    <install-info>
<node-name>buccaneer</node-name>
<jdbc-zipfile-location>/home/db2as/sqllib/java/db2java.zip</jdbc-zipfile-location>
</install-info>  </jdbc-driver>  <data-source name="Sample Datasource" action="update">
<database-name>/usr/WebSphere/AppServer/bin/myidb.prp</database-name>    <jdbc-driver-name>Admin DB
Driver</jdbc-driver-name>    <minimum-pool-size>1</minimum-pool-size>
<maximum-pool-size>10</maximum-pool-size>    <connection-timeout>120000</connection-timeout>
<idle-timeout>180000</idle-timeout>    <orphan-timeout>1800000</orphan-timeout>
```

The following section contains the node, or physical machine, in the administrative domain. Its hostname is buccaneer:

- Skip ahead
- View object syntax

```
  <node name="buccaneer" action="update">
<deployed-jar-directory>/usr/WebSphere/AppServer/deployedEJBs</deployed-jar-directory>
<dependent-classpath></dependent-classpath>
```

Next there is an application server. In this case, it is the default application server, "Default Server":

- Skip ahead
- View object syntax

```
    <application-server name="Default Server" action="update">      <executable>java</executable>
<command-line-arguments/>      <environment/>      <user-id></user-id>      <group-id></group-id>
<working-directory></working-directory>      <umask>18</umask>      <stdin></stdin>
<stdout>/usr/WebSphere/AppServer/logs/default_server_stdout.log</stdout>
<stderr>/usr/WebSphere/AppServer/logs/default_server_stderr.log</stderr>
<process-priority>20</process-priority>      <maximum-startup-attempts>2</maximum-startup-attempts>
<ping-interval>60</ping-interval>      <ping-timeout>200</ping-timeout>
<ping-initial-timeout>300</ping-initial-timeout>
<selection-policy>roundrobinpreferlocal</selection-policy>
<trace-specification></trace-specification>      <trace-output></trace-output>
<transaction-log-file></transaction-log-file>      <system-properties/>
<debug-enabled>false</debug-enabled>      <transaction-timeout>120</transaction-timeout>
<transaction-inactivity-timeout>60000</transaction-inactivity-timeout>
<thread-pool-size>20</thread-pool-size>      <security-enabled>false</security-enabled>
```

The application server also contains a servlet engine

- Skip ahead
- View object syntax

```
      <servlet-engine name="Default Servlet Engine" action="update">
<maximum-connections>25</maximum-connections>        <transport-port>-1</transport-port>
<servlet-mode>1</servlet-mode>        <transport-type name="ose">          <ose-transport>
<link-type>local</link-type>            <log-file-mask trace="false" inform="false" warning="false"
error="true"/>            <queue-name>ibmoselink</queue-name>
<clone-index>1</clone-index>            <native-log-file>native.log</native-log-file>
</ose-transport>        </transport-type>        <isclone>false</isclone>
```

In turn, the servlet engine contains a few Web applications. Two are omitted from this example. The third (shown here) is named "examples":

- Skip ahead
- View object syntax

```
        <web-application name="examples" action="update">          <description>Useful Example
Servlets</description>
<document-root>/usr/WebSphere/AppServer/hosts/default_host/examples/web</document-root>
<classpath>            <path value="/usr/WebSphere/AppServer/hosts/default_host/examples/servlets"/>
</classpath>          <error-page>/debug_error.jsp</error-page>          <session-config>
            <session-timeout>0</session-timeout>
          </session-config>          <mime-mapping>
            <extension>*.gam</extension>
            <mime-type>nixon</mime-type>
          </mime-mapping>
          <welcome-file-list>
            <welcome-file>slappy.html</welcome-file>
          </welcome-file-list>
          <error-page-j2ee>
```

```
            <exception-type>anException</exception-type>
            <location>except.html</location>
          </error-page-j2ee>
          <error-page-j2ee>
            <error-code>333</error-code>
            <location>horrid.html</location>
          </error-page-j2ee>
          <taglib>
            <taglib-uri>/WEB-INF/app.tld</taglib-uri>
            <taglib-location>/WEB-INF/app.tld</taglib-location>
          </taglib>
          <filter-list/>          <group-attributes/>          <auto-reload>true</auto-reload>
<reload-interval>9000</reload-interval>          <enabled>true</enabled>
<root-uri>default_host/webapp/examples</root-uri>          <shared-context>false</shared-context>
<shared-context-jndi-name>SrdSrvltCtxHome</shared-context-jndi-name>
<isclone>false</isclone>
```

The example app contains several servlets. The first one is administered by the name "simpleJSP":

- Skip ahead
- View object syntax

```
          <servlet name="simpleJSP" action="update">            <description>Simple JSP
Servlet</description>            <code>SimpleJSPServlet</code>            <init-parameters>
<parameter name="" value=""/>            </init-parameters>
<load-at-startup>false</load-at-startup>            <debug-mode>false</debug-mode>
<uri-paths>              <uri value="/simpleJSP.servlet"/>              <uri value="/simpleJSP"/>
</uri-paths>            <enabled>true</enabled>            <isclone>false</isclone>
</servlet>
```

The Web application contains some other servlets that are not listed here. The Web application ends:

```
        </web-application>
```

Now, a session manager follows. Recall, the tag for the servlet engine is still open, indicating that all of these objects are contained by the servlet engine:

- Skip ahead
- View object syntax

```
        <session-manager name="Session Manager" action="update">
<enable-sessions>true</enable-sessions>          <enable-url-rewriting>false</enable-url-rewriting>
<enable-cookies>true</enable-cookies>
<enable-protocol-switch-rewriting>false</enable-protocol-switch-rewriting>          <cookie
name="sesessionid">            <comment>servlet session support</comment>
<domain></domain>            <maximum>-1</maximum>            <path>/</path>
<secure>false</secure>          </cookie>
<interval-invalidation-time>1800</interval-invalidation-time>
<persistent-sessions>false</persistent-sessions>
<persistence-type>directodb</persistence-type>          <database location="jdbc:db2:was">
<driver>COM.ibm.db2.jdbc.app.DB2Driver</driver>            <user-id></user-id>
<password></password>            <number-of-connections>30</number-of-connections>
</database>          <enable-stat-collection>true</enable-stat-collection>
<using-cache>false</using-cache>          <using-multi-row>false</using-multi-row>
<using-manual-update>false</using-manual-update>
<using-native-access>false</using-native-access>          <base-memory-size>1000</base-memory-size>
<allow-overflow>true</allow-overflow>          <data-source name=""/>        </session-manager>
```

Next, the servlet engine contains a user profile manager:

- Skip ahead
- View object syntax

```
        <user-profile-manager name="User Profile Manager" action="update">
<enable-user-profile>false</enable-user-profile>
<data-wrapper>com.ibm.servlet.personalization.userprofile.UserProfile</data-wrapper>
<remote-interface-ro>com.ibm.servlet.personalization.userprofile.UP_ReadOnly</remote-interface-ro>
<remote-interface-rw>com.ibm.servlet.personalization.userprofile.UP_ReadWrite</remote-interface-rw>
<home-interface-ro>com.ibm.servlet.personalization.userprofile.UP_ReadOnlyHome</home-interface-ro>
<home-interface-rw>com.ibm.servlet.personalization.userprofile.UP_ReadWriteHome</home-interface-rw>
<jndi-name-ro>UP_ReadOnlyHome</jndi-name-ro>          <jndi-name-rw>UP_ReadWriteHome</jndi-name-rw>
</user-profile-manager>
```

The user profile manager is the last object in the servlet engine and the application server. The end tags for each are displayed:

```
      </servlet-engine>    </application-server>
```

Then the end tag for the node is placed, indicating that all the elements pertaining to the node have been addressed:

```
  </node>
```

Several enterprise application entries come after this (only the first is shown here):

- Skip ahead
- View object syntax

```
  <enterprise-application name="AdminApplication" action="create">    <uri name="/admin"/>
<web-application name="admin"/>  </enterprise-application>
```

Followed by several URI security entries (only one is shown):

- Skip ahead
- View object syntax

```
  <uri-security>    <uri name="/admin"/>    <method-group-mapping method="HTTP_GET"
method-group="ReadMethods"/>    <method-group-mapping method="HTTP_POST"
method-group="ReadMethods"/>    <method-group-mapping method="HTTP_PUT"
method-group="WriteMethods"/>    <method-group-mapping method="HTTP_DELETE"
method-group="RemoveMethods"/>  </uri-security>
```

Followed by the server security configuration:

- Skip ahead
- View object syntax

```
  <security-config security-enabled="false" security-cache-timeout="600">    <app-security-defaults>
<realm-name>raleigh.ibm.com</realm-name>      <challenge-type ssl-enabled="false">
<basic-challenge/>      </challenge-type>    </app-security-defaults>    <auth-mechanism>
<localos>        <user-id>root</user-id>        <password>$server-password$</password>
</localos>    </auth-mechanism>  </security-config>
```

The last set of tags defines the method groups:

- Skip ahead
- View object syntax

```
  <method-group>ReadMethods</method-group>  <method-group>WriteMethods</method-group>
<method-group>RemoveMethods</method-group>  <method-group>CreateMethods</method-group>
<method-group>ExecuteMethods</method-group>  <method-group>FinderMethods</method-group>
```

The last item is the end tag for export itself:

```
</websphere-sa-config>
```

6.6.0.2.1.1.2: XMLConfig - Example of a partial export

To do a partial export of your Websphere Administrative Domain configuration into an XML file, you need to create an XML file specifying the resources you would like to export. This partial file is then used as an input parameter in the XMLConfig export command line.

The partial XML file always begins with the following two header lines:

```
<?xml version="1.0"?><!DOCTYPE websphere-sa-config SYSTEM
"$server_root$$dsep$bin$dsep$xmlconfig.dtd" >
```

The contents of the Websphere Administrative Domain that you wish to extract into an XML file start with <websphere-sa-config> and end with </websphere-sa-config>. What goes in between these tags depends on what you want to export. This is an example of a partial XML file you would create to export the entire contents of an Application Server on a node named "mynode":

```
<?xml version="1.0"?><!DOCTYPE websphere-sa-config SYSTEM
"$server_root$$dsep$bin$dsep$xmlconfig.dtd"><websphere-sa-config>    <node name="mynode"
action="locate">       <application-server name="Default Server" action="export">
</application-server>    </node></websphere-sa-config>
```

The next example is of an XML file you would create to export a single Web application from the same Application Server on the same node:

```
<?xml version="1.0"?><!DOCTYPE websphere-sa-config SYSTEM
"$server_root$$dsep$bin$dsep$xmlconfig.dtd" ><websphere-sa-config>    <node name="mynode"
action="locate">       <application-server name="Default Server" action="locate">
<servlet-engine name="Default Servlet Engine" action="locate">             <web-application
name="default_app" action="export">             </web-application>          </servlet-engine>
</application-server>    </node></websphere-sa-config>
```

As an example of how you would do a partial configuration export into an XML file, we can name the second file above (the one that would export a single Web application from the node "mynode") *PartialFile.xml*, place it in the appserver/bin directory, and run the following command from that directory:

```
XMLConfig -export NewExport.xml -adminNodeName mynode -partial PartialFile.xml
```

An XML file named *NewExport.xml* will then be created in the appserver/bin directory with the following output:

```
<?xml version="1.0"?><!DOCTYPE websphere-sa-config SYSTEM
"$XMLConfigDTDLocation$$dsep$xmlconfig.dtd" ><websphere-sa-config>  <node name="mynode"
action="locate">    <application-server name="Default Server" action="locate">      <servlet-engine
name="Default Servlet Engine" action="locate">        <web-application name="default_app"
action="update">          <description>Default Application</description>
<document-root>d:\IBM HTTP Server\htdocs</document-root>          <classpath>            <path
value="d:\WebSphere\AppServer\hosts\default_host\default_app\servlets"/>            <path
value="d:\WebSphere\AppServer\servlets"/>          </classpath>
<error-page>/ErrorReporter</error-page>          <session-config>
<session-timeout>0</session-timeout>          </session-config>          <welcome-file-list/>
<filter-list/>          <group-attributes/>          <auto-reload>true</auto-reload>
<reload-interval>9</reload-interval>          <enabled>true</enabled>
<root-uri>default_host/</root-uri>          <shared-context>false</shared-context>
<shared-context-jndi-name>SrdSrvltCtxHome</shared-context-jndi-name>          <servlet name="snoop"
action="update">            <description>Snoop servlet</description>
<code>SnoopServlet</code>            <init-parameters>              <parameter name="param1"
value="test-value1"/>            </init-parameters>
<load-at-startup>false</load-at-startup>            <debug-mode>false</debug-mode>
<uri-paths>              <uri value="/servlet/snoop/*"/>              <uri
value="/servlet/snoop2/*"/>            </uri-paths>            <enabled>true</enabled>
</servlet>          <servlet name="hello" action="update">            <description>Simple Hello
servlet</description>            <code>HelloWorldServlet</code>            <init-parameters/>
<load-at-startup>false</load-at-startup>            <debug-mode>false</debug-mode>
<uri-paths>              <uri value="/servlet/hello"/>            </uri-paths>
<enabled>true</enabled>          </servlet>          <servlet name="ErrorReporter" action="update">
<description>Default error reporter servlet</description>
<code>com.ibm.servlet.engine.webapp.DefaultErrorReporter</code>            <init-parameters/>
<load-at-startup>true</load-at-startup>            <debug-mode>false</debug-mode>
<uri-paths>              <uri value="/ErrorReporter"/>            </uri-paths>
<enabled>true</enabled>          </servlet>          <servlet name="invoker" action="update">
<description>Enables Serving Servlets By Classname</description>
<code>com.ibm.servlet.engine.webapp.InvokerServlet</code>            <init-parameters/>
<load-at-startup>true</load-at-startup>            <debug-mode>false</debug-mode>
<uri-paths>              <uri value="/servlet/*"/>            </uri-paths>
<enabled>true</enabled>          </servlet>          <servlet name="jsp10" action="update">
<description>JSP 1.0 support servlet</description>
<code>com.sun.jsp.runtime.JspServlet</code>            <init-parameters/>
<load-at-startup>true</load-at-startup>            <debug-mode>false</debug-mode>
<uri-paths>              <uri value="*.jsp"/>              <uri value="*.jsv"/>              <uri
value="*.jsw"/>            </uri-paths>            <enabled>true</enabled>          </servlet>
</web-application>      </servlet-engine>    </application-server>  </node></websphere-sa-config>
```

6.6.0.2.1.2: XMLConfig grammar

This section discusses the general structure of an XML element. For detailed information about each object, see the package summary file. See also the full export example page.

Each object tag in the XML document contains:

- An object type, such as "servlet" or "virtual-host"
- A **name** attribute that identifies the particular resource on which to perform the action
- An **action** attribute that controls the behavior of the import or partial export operation

For example, a servlet engine named MyServletEngine has the object tag:

```
<servlet-engine name="MyServletEngine" action="update">
```

In this case, the action is "update." The list below describes all of the available actions.

Unless otherwise stated, the actions apply only to the import operation.

create

Adds the specified resource to the administrative domain. If the resource already exists, the create action is treated as an update action.When using this action, you must specify all required attributes for the object type.

update

Updates the properties of a specified resource. You need only specify the properties you want to update.

However, if the resource does not exist, the update action becomes a create. In such a case, if some of the properties are not specified, an error will occur.

delete

Removes the specified resource and its children (recursive delete).

locate

Locates the specified resource. Use this action to provide the containment path to specific child resources. Applies to the partial export operation as well as the import operation.

export

Exports the configuration of the specified resource and its children to the output XML document. Applies only to the partial export operation.

start

Starts the specified resource (if applicable).

stop

Stops the specified resource, (if applicable).

stopforrestart

Stops the specified resource, and restarts it.(if applicable). Node only.

restart

Stops the specified resource and starts it again (if applicable).

enable

Makes the specified resource available for user requests (currently applies only to servlets and Web applications).

disable

Makes the specified resource unavailable for user requests (currently applies only to servlets and Web applications).

createclone

create a clone

associateclone

associate a clone to a model

disassociateclone

disassociate a clone from a model

 Some of the operational attributes, such as start and stop, do not apply to all object types. In general, if the operation is supported in the WebSphere Administrative Console, it is supported in the XML import operation.

6.6.0.2.1.3: XMLConfig - Using the tool programmatically

The XMLConfig class is structured so that you can use it programmatically to retrieve information as a TXDocument or Element. The import/export facility can thus be included in a Java program, as well as being operated from a command line.

Creating platform-neutral configurations

For import and partial export operations, a variable substitution operation is performed on the input XML document, allowing you to create platform-neutral XML documents. These variables are available:

$server_root$

Replace with the WebSphere Application Server installation directory, such as C:\WebSphere\AppServer on Windows NT.

$psep$

Replace with the path separator as specified by the operating system JDK.

- On Windows NT, it is ; (semicolon)
- On AIX and Solaris, it is : (colon)

$dsep$

Replace with the directory separator as specified by the operating system JDK.

- On Windows NT, it is \ (backward slash)
- On AIX and Solaris, it is / (forward slash)

Security XML configurations

When configuring security with XML, you should be familiar with the following two factors:

1. Passwords and variable substitution
2. Searches of the user registry

Javadoc for the tool

It is recommended that you refer to the Javadoc for the latest programmatic use of XMLConfig, and refer to the exported xml for the sample xml for repository objects.

Javadoc for com.ibm.websphere.xmlconfig class and all of the related object classes resides in the apidocs directory:

`<installation_root>\web\apidocs\package and class name`

See the package summary file for a list of the class names, such as ApplicationServerConfig. The Javadoc is labeled by the class name preceded by the package name, com.ibm.websphere.

6.6.0.2.1.3.1: XMLConfig - Passwords and variable substitution

Passwords are required for the WebSphere security configuration, but exposing passwords during an XML import or export by putting an unencrypted password in the XML text file would create an unacceptable security risk. Thus, WebSphere Application Server uses special password tags in the exported XML file that signify XML variables that replace the real passwords during an import. Consequently, the passwords are only part of the XMLConfig command line, which you can clear after invoking the utility.

Three of the password variables have constant names while the others depend on the name to what they are related. The three constant variable names are:

server-password

> This is the password for the ID that has all permissions and rights to access the administrative server. It is the password that is entered during the installation process, and the one that appears on the User Registry tab of the Configure Global Security Settings task.

ltpa-password

> This is the password for generating LTPA keys. It is the password entered on the LTPA Password dialog when changing the Authentication Mechanism to LTPA for the first time.

ldap-bindpwd

> This is the password that the security server will use to bind to an LDAP directory during searches. It is the password entered on the User Registry tab of the Configure Global Security Settings task when LTPA is the Authentication Mechanism.

<enterprise app name>_AppSecurityPwd *[where <enterprise app name> is the name of an enterprise application with a defined application identity]*

> This is the password associated with the identity defined as the application identity. Application identities are set on the last panel of the Configure Application Security task and are limited to users defined in the configured user registry.

All password variables must be substituted on the XMLConfig command line using the -substitute parameter. Multiple substitutions are separated by a semicolon. For example, the server-password and LTPA-password variables may be substituted with the following command line:

```
XMLConfig -import was.xml -adminNodeName myhost -substitute
"server-password=pwd1;LTPA-password=pwd2"
```

6.6.0.2.1.3.2: XMLConfig - User registry searches

The permission element accepts access-id-search children that allow you to specify user registry searches instead of explicit user registry entries. The values of these elements are substituted into the appropriate search filters for the specified types before the searches execute. This gives you an easy way to specify user registry entries for permissions, as opposed to knowing the exact and often complicated unique ID for each user registry entry.

There is a condition for searches that must be met for a permission import to succeed. Since an XML import is not interactive and the XML import utility has no way of choosing the intended match from multiple matches, user registry searches must return one and only one match. A permission import will fail if more than one match is returned from a search.

Not all types may be supported by a user registry even though the type attribute of an access-id-search element allows "user", "group", and "role". Therefore, importing a permission element will fail when the type specified is not supported by the user registry.

6.6.0.2.1.4: wartoxmlconfig script

The wartoxmlconfig.(bat|sh) scripts performs from the command line the same function as the Convert WAR File task on the console. The command line syntax is shown below:

```
wartoxmlconfig [war filename] [webapp destination] [admin node] [node] [server]    [servlet engine]
[virtual host] [webapp path] [webapp name]
```

- **war filename** - full path to the WAR file
- **webapp destination** - directory where web application will be rooted (a subdirectory will be created that matches the specified web app name)
- **admin node** -name of the node containing the admin server you are connecting to (as shown in the Admin GUI)
- **node** - name of the node you are installing the WAR on to (as shown in the Admin GUI)
- **server** - name of the server you are installing the WAR on to (as shown in the Admin GUI)
- **servlet engine** - name of the servlet engine you are installing the WAR on to (as shown in the Admin GUI)
- **virtual host** - name of the virtual host you wish this application to be accessible from (as shown in the Admin GUI)
- **webapp path** - the context path of the web application
- **webapp name** - the name of the web application to be shown in the Admin GUI

Example:

```
wartoxmlconfig c:\temp\servlet-tests.war c:\websphere\appserver\hosts\default_host   mynode mynode
"Default Server" "Default Servlet Engine" default_host   /servlet-tests Servlet_Tests
```

6.6.0.2.1.5: Troubleshooting XMLConfig

This article describes what to do if XMLConfig fails and displays the following message at its command line.

```
Unable to create Initial Context. Please check name service settings:
org.omg.CORBA.COMM_FAILURE:   minor code: 3  completed: Nojava com.ibm.websphere.xmlconfig.XMLConfig
{ ( -import <xml data file> ) ||        [ ( -export <xml output file> [-partial <xml data file>] ) }
-adminNodeName <primary node name>        [ -nameServiceHost <host name> [ -nameServicePort <port
number> ]]        [-traceString <trace spec> [-traceFile <file name>]]        [-substitute
<"key1=value1[;key2=value2[...]]">]}        In input xml file, the key(s)  should appear as $key$
for substitution.
```

One or more of the following conditions is likely to have caused the problem:

- The admin server didn't start properly.
- XMLConfig is being run remotely from the machine on which the product administrative serveris installed, and the -nameServiceHost parameter has not been set to the hostname of the machine containing the administrative server.
- The naming service port was changed from the default 900 on the application server and the -nameServicePort parameter was not set to the changed port value
- The -nameServicePort parameter was defined, but the -nameServiceHost parameter was not.
- If the fully qualified host name (hostname + domain) is specified for the -adminNodeName parameter, instead of the node name (short host name), expect this error:

  ```
  001.553 22f45b XMLConfig     X Unabled to export Virtual Host Data: {0}
  javax.naming.NameNotFoundException:
  ```

To resolve the problem:

1. Check whether the administrative server started successfully, as describedin the documentation for starting the administrative server.
2. If the XMLConfig tool is being run remotely with respect to the machine on whichthe IBM Websphere administrative server is running, ensure that the -nameServiceHost (as well as the -adminNodeName) parameter are set to the host name of the remote machine.
3. If the naming service port has been changed from the default 900 on the machineon which the administrative server is running, look in the administrative server configuration file for the parameter:

   ```
   com.ibm.ejs.sm.adminServer.bootstrapPort
   ```

   Set the XMLConfig -nameServicePort parameter to this port value.
4. If you use the -nameServicePort parameter, you must also use the -nameServiceHost parameter even if you are running XMLConfig on the same machine asthe administrative server.
5. Modify the -adminNodeName parameter to use the node name, which can be found as your hostname in your TCP/IP networking configuration and also in the Websphere administrative console under "Websphere Administrative Domain."

6.6.0.2.2: WebSphere Control Program (wscp)

The WebSphere Control Program (wscp) is a command line client for the administrativeserver. It can be used to administer application servers, enterprise applications, andother types of WebSphere Application Server objects.