IBM WebSphere Transcoding Publisher

Version 4.0

Developer's Guide

Developer's Guide

IBM WebSphere(TM) Transcoding Publisher (referred to from now on as Transcoding Publisher or WTP) enables you to make Web-based information available to users of handheld and other pervasive devices economically and efficiently. This Guide describes the Transcoding Publisher architecture and the tasks required to extend the product's function. In addition, this guide explains the steps involved in writing custom transcoders, utilizing Transcoding Publisher facilities like preference aggregation, packaging files for use with Transcoding Publisher, working with stylesheets and annotators, using XML configuration, and other customization topics.

The Contents view on the left of this page shows the chapters in this Guide. To open a chapter and expand its subheadings, click on the chapter name. You can click on a subheading to go directly to that topic.

Before using this information and the product it supports, read the information in Notices.

Overview

Transcoding Publisher is network software that modifies content presented to users based on the information associated with the request, such as device constraints, network constraints, and organizational policies. This transformation of information from one form to another is the central idea behind ***transcoding***, a technology particularly suited to bridging the gaps between the variety of data formats encountered on the Web and the devices comprising them.

Transcoding Publisher uses specialized programs called transcoders to perform different conversions. Typical conversions performed by the transcoders supplied with Transcoding Publisher include:

- Converting XML documents from one form of XML to another through the use of stylesheets
- Converting HTML documents into other formats, including Wireless Markup Language (WML), Handheld Device Markup Language (HDML), VoiceXML, and the Compact HTML used by i-mode devices
- Simplifying HTML documents with a number of customizations, such as removing objects or features not supported on the target device (like JavaScript) or converting tables to lists
- Working with WebSphere Translation Server to translate text from one language to another
- Manipulating images to affect scaling, compression, color depth, and format

Transcoding Publisher's pluggable architecture not only enables you to selectively apply transcoders known to the system but also provides a mechanism for adding custom transcoders you create yourself or transcoders supplied by third-party vendors. Additionally, Transcoding Publisher includes a set of core services that all transcoders can access, such as the ability to acquire preference information in order to respond differently to requests for different users or different devices.

Transcoding Publisher Toolkit

The Transcoding Publisher Toolkit provides information and tools for programmers who want to customize or extend Transcoding Publisher's capabilities. This might include modifying existing preference profiles, writing a custom text-clipping transcoder to extract text for display on a Smartphone, or creating and packaging a set of transcoder files to support a new device. The Transcoding Publisher Toolkit consists of the following components:

Developer's Guide **(this document)**	Describes the Transcoding Publisher architecture and the tasks required to extend the product's function. In addition, the *Developer's Guide* explains the steps involved in writing custom transcoders and annotators, utilizing Transcoding Publisher facilities like preference aggregation, packaging files for use with Transcoding Publisher, and other customization topics.
Samples	Provides samples of the preference profile and stylesheet files used for administration, as well as source code for samples that demonstrate how transcoders and annotators can be implemented.

API documentation (Javadoc format)	Describes the API available for tailoring Transcoding Publisher's document clipping function and the API documenting the JavaBean transcoders. There are also links to the Web Intermediaries (WBI) API and Sun's Java Servlet API (Version 2.1), for use in developing servlet-based transcoders.
Development tools	Includes tools that aid in developing and debugging of transcoders.

Related Information

For more information about Transcoding Publisher, you can also refer to the following:

IBM WebSphere Transcoding Publisher Web site

The IBM Web site for Transcoding Publisher provides up-to-date information about the product, including the latest release and support information, articles about transcoding technology, and an active developer's area with common questions, additional samples, and tool enhancements.

Transcoding Publisher Administrator's Guide

This online document provides information about installing, configuring, and running Transcoding Publisher, with an emphasis on administrative tasks and day-to-day operation. Troubleshooting tips are also described.

Architecture and Components

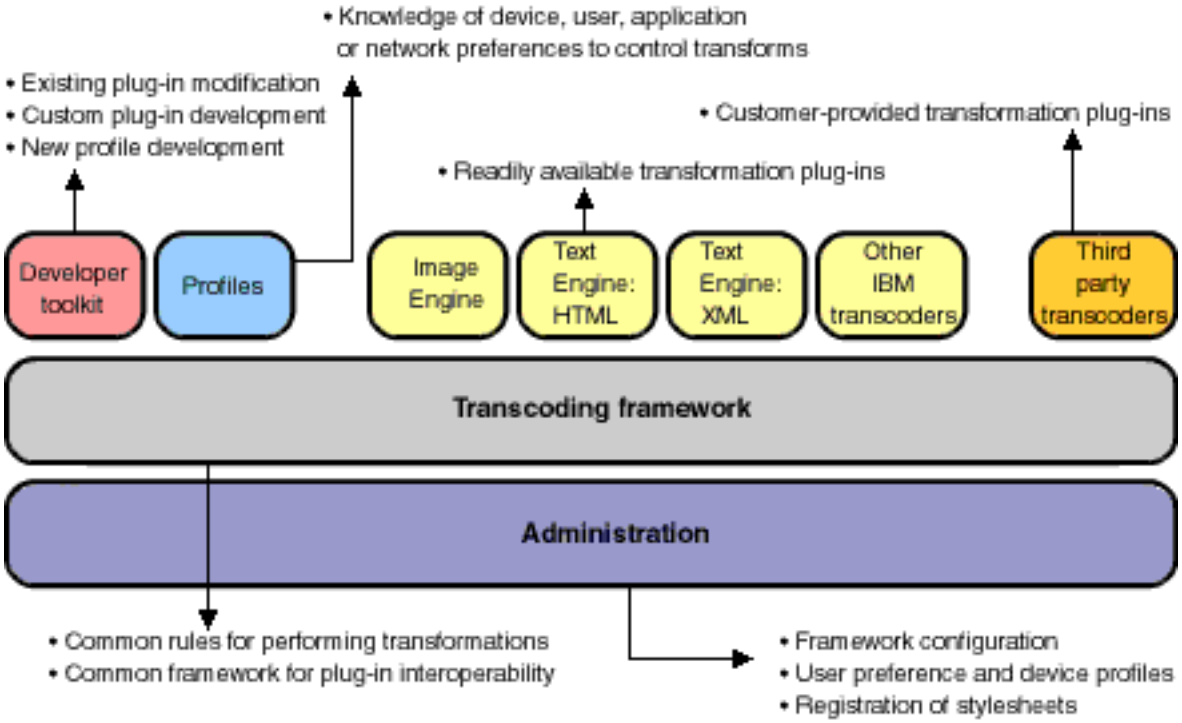
Transcoding Publisher transforms content based on what it knows the requesting device can handle, and on the capacity of the network being used. Web content can be transformed differently for different devices and networks. Transcoding Publisher supports your existing applications and data without requiring that you redesign corporate systems for the different standards found in each device. Transcoding Publisher can support any type of Web data, including HTML pages from a Web server, HTML output from a host application (such as that produced by IBM's Host Publisher), and Extensible Markup Language (XML) data from a back-end transaction system. Transcoding Publisher also tailors images to adjust screen size, file size, and numbers of colors.

Transcoding Publisher can be deployed in your network in several ways. Refer to the *Administrator's Guide* for information about deployment models.

As the figure below indicates, Transcoding Publisher is made up of a number of interrelated components that provide a flexible, extendable platform for building and deploying transcoders. The primary elements of the system include:

- A transcoding framework that provides common services to transcoders that are registered to, or plugged into, the framework.
- A base set of IBM-supplied transcoders that transform Web content. These transcoders provide various image and text transformations, as well as caching support for transcoded content. Customer-provided transcoders can also be used alongside the transcoders included with Transcoding Publisher.

  Transcoding Publisher supports the following types of transcoders: Web Intermediary (WBI) plug-ins, servlets written to the Java Servlet API, and JavaBean versions of selected IBM transcoders.
- A set of profiles that contain information about various types of transformation preferences.
- Administrative services for configuring and managing Transcoding Publisher, including preference and profile manipulation, as well as stylesheet and annotator management.


• Knowledge of device, user, application or network preferences to control transforms
• Existing plug-in modification
• Custom plug-in development
• New profile development
• Customer-provided transformation plug-ins
• Readily available transformation plug-ins
Developer toolkit
Profiles
Image Engine
Text Engine: HTML
Text Engine: XML
Other IBM transcoders
Third party transcoders
Transcoding framework
Administration
• Common rules for performing transformations
• Common framework for plug-in interoperability
• Framework configuration
• User preference and device profiles
• Registration of stylesheets

Transcoding Framework

Transcoding Publisher is built around a framework that provides common services to transcoders that are registered, or plugged into, the framework. In addition to managing the flow of HTTP requests and responses through Transcoding Publisher, the framework supplies mechanisms for manipulating the data as it passes through. This includes the ability to evaluate preference information to respond differently to requests from different kinds of devices.

The backbone of the framework incorporates Version 4.5 of the Web Intermediaries (WBI) technology developed by IBM Research and made available through IBM's alphaWorks Web site. WBI is a technology used to create network intermediaries, which are programmatic entities that are positioned along the HTTP data stream and can monitor, filter, and otherwise customize the data as it flows along the stream. Although WBI itself provides for a wide range of applications, Transcoding Publisher utilizes WBI to create a robust, extendable environment for transcoding tasks in particular.

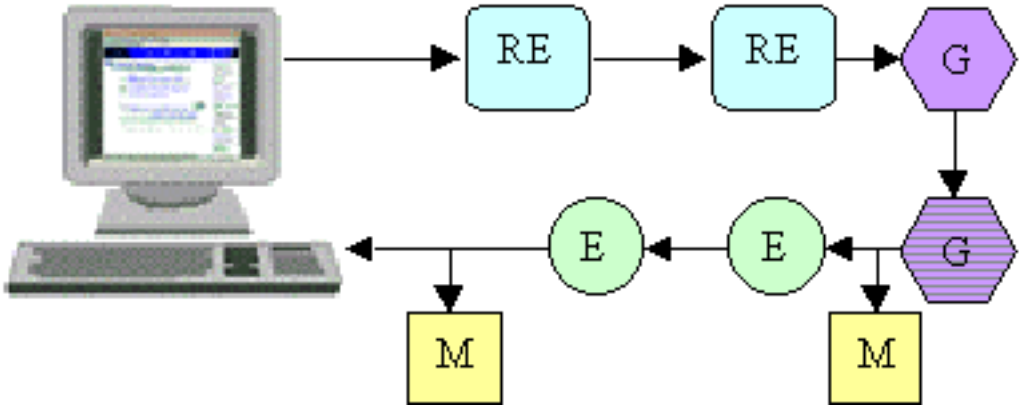
WBI Basics

To better understand and take advantage of the capability of Transcoding Publisher, it is important to understand some of the concepts that are central to the WBI technology. At its core, WBI is an HTTP request and response processor, receiving requests and then manipulating the data stream with one or more programmable modules to form a response. The various module types are described below:

- ***Request editors*** take a request from the stream and modify its contents before placing it back into the stream. A typical transform performed by a request editor might be to modify the fields in the HTTP header.
- ***Generators*** accept the request, either directly or as output from a request editor, and create a response for return to the client. A common example is a generator that retrieves an object from a Web server.
- ***Response editors*** receive a response on its way to the client and modify its contents before inserting it back into the stream. Response editors can be used to customize the response on its way back to the browser.
- ***Monitors*** can participate in the processing flow but cannot alter the data. Instead, a monitor acts as an observer, viewing copies of any designated responses. Though not involved in the request/response processing, a monitor can be used to perform related and supporting tasks, such as counting accesses to a particular Web page or compiling a history of the pages a user has visited.

Taken together, these modules are referred to as ***MEGs*** (monitors, editors, and generators) and are the building blocks of WBI plug-ins. In the context of Transcoding Publisher, a transcoder is a WBI plug-in that consists of one or more MEGs and has a specialized purpose of transforming information based on criteria like device constraints and network requirements.

The following figure illustrates how a WBI transaction might look and how the various MEGs could be arranged.



A WBI transaction. The HTTP data flows through a series of request editors (RE), generators (G), and response editors (E) before being returned to the client. Note that once one generator creates a response, other generators in the stream are ignored. Monitors (M) are also positioned at various points in the response stream.

The flow of data through the MEGs is controlled by a set of rules that specifies the conditions under which a particular MEG is triggered. The conditions can involve various aspects of the request header or the response header, such as the host, the content-type, user-agent field, and so on. If the triggering rules for more than one MEG are satisfied for a transaction, each MEG is executed according to its priority value. However, for generator MEGs, only the first matching generator is executed, and other generators are ignored.

Details regarding the architecture of WBI and documentation for the WBI API are included in the WBI Development Kit.

Java Servlet API Support

In addition to supporting transcoders written as MEG plug-ins, the Transcoding Publisher framework also allows industry-standard servlets to be incorporated as transcoders, using Version 2.1 of the Sun Java Servlet API. Servlets written to this API for other environments can easily be ported to run in Transcoding Publisher, often with few or no changes, and new transcoders will be usable in the many Web servers that support servlets.

Transcoders written to this API are also referred to as ***MEGlets***, because they can play the same roles and perform the same tasks as MEG-based plug-ins but are written to the servlet API rather than the WBI API. Using the Java Servlet API produces code that can be readily ported to other environments and allows existing servlet code to be used in the transcoding framework.

For a more detailed discussion of the servlet API support in Transcoding Publisher and the steps involved in creating and using servlet-based transcoders, read Adding Transcoders to Transcoding Publisher.

Preference Aggregation

Transcoding Publisher modifies documents according to preference settings representing the characteristics of supported devices, networks, and users. Preferences are defined in profiles, and transcoders can access the values for these preferences through the framework. Because the same

preference can appear in different profiles and have different values, depending on the purpose of the profile, Transcoding Publisher provides a ***preference aggregator*** to resolve potential conflicts. For example, a device profile can state that color images are supported, while the network profile can specify that color should be suppressed to save network bandwidth. When a transcoder needs to know the value of this preference, the preference aggregator evaluates the conflicting values and determines which value to supply to the transcoder. For more information about specifying preferences and understanding the implications of their use with the preference aggregator, read Working with Preference Profiles.

Transcoding Publisher Transcoders

IBM provides several transcoders with Transcoding Publisher that can be used for a number of common Web-oriented transformations:

- The ***image transcoder*** modifies images to better support the display capability of a device.
- The ***text transcoder*** converts textual data, such as HTML or XML, from one format to another and can perform a number of transformations to simplify the output.
- The ***fragmentation transcoder*** fragments XML documents into pieces small enough to be managed by the target device.
- The ***HTML DOM generator*** creates a Document Object Model (DOM) version of incoming HTML documents.
- The ***annotation transcoder*** interprets the contents of files written with Transcoding Publisher's annotation language to perform document clipping.
- The ***Palm transcoder*** converts HTML into documents easily rendered by Palm.Net equipped PalmOS devices.
- The ***HTML to HDML transcoder*** converts HTML documents into documents which can be displayed on devices with HDML browsers.
- The ***HTML to VoiceXML transcoder*** converts HTML documents into documents that can be read by VoiceXML browsers.
- The ***HTML to WML transcoder*** converts HTML documents to WML for devices with WAP browsers.
- The ***HTML to Compact HTML transcoder*** converts HTML documents to Compact HTML documents for devices with CHTML browsers.

These transcoders are installed with the product and enabled by default (with the exception of the HTML DOM generator). To verify whether the transcoders are enabled or to change their status, you can use the Administration Console. Instructions for accessing the console and using it are included in the *Administrator's Guide*.

Image Transcoder

Using the Jimage library developed by IBM Research, this transcoder is used to modify images so that they are more closely tailored to the capabilities of the target device. The transcoder accepts both GIF and JPEG images, and, based on the preference settings for the associated device or network, performs several functions, including:

- Converting the image format to GIF, JPEG, or Wireless bitmap (WBMP), depending on the format of the original image.
- Reducing the image's color depth. For example, a 24-bit image can be reduced to an 8, 4, 2, or 1-bit image.
- Converting color images to grayscale images.
- Specifying the quality level of a JPEG image.

- Adjusting the size of an image by scaling according to a percentage value.

The image transcoder contains an editor and a request editor. While the editor performs much of the image transcoding, the request editor provides other benefits:

- Correcting transaction discrepancies between a browser and a Web server. For example, some browsers might not properly indicate that a device can accept GIF images, resulting in the Web server refusing to send GIFs to the device. The image transcoder can modify the HTTP header before forwarding the request to the Web server and ensure that the server transfers the expected images.
- Expanding a device's apparent capabilities. For devices that do not accept JPEG images, the image transcoder can alter the request to indicate that the device can accept JPEGs and then transcode the images to GIFs in the process of sending the response back to the device.

Text Transcoders

The text transcoders include a set of editors and a generator that specialize in transforming textual data, particularly HTML and XML documents. These transforms include:

- Simplification of HTML documents. For example, based on instructions derived from device and network profiles, the text transcoder can replace images embedded in pages with links to images and remove objects or features, such as animation files or JavaScript, that are not supported by the target device.
- Stylesheet application to XML documents. The text engine uses the Xalan-Java stylesheet processor, with enhancements for selecting a XSL stylesheets based on the device or network associated with the specific request. For more information about Transcoding Publisher's support for stylesheets, see Using Stylesheets.
- Format translation from HTML to Wireless Markup Language (WML). This can be the first step in modifying a browser-based application for display on a device enabled for the Wireless Application Protocol (WAP). A custom transcoder called a ***text clipper*** can be used to select the right subset of the application content for effective HTML-to-WML transcoding. For more information about how to create and employ text clippers for use with Transcoding Publisher's data conversion capability, read Document Clipping .

There are several transcoders that by default are not enabled. To convert data, you must make certain you enable the appropriate transcoder. Instructions for enabling transcoders are included in the *Administrator's Guide*.

Fragmentation Transcoder

The fragmentation transcoder is a set of editors and generators that receives requests for WML, HDML, i-mode, Compact HTML, and PalmOS HTML data and, based on the maximum fragment size that the requesting device can store, subdivides the data into fragments that the device can handle. As the device needs additional data, Transcoding Publisher sends the fragmented pieces in sequence until the request has been satisfied. In this way, Transcoding Publisher can help prevent the device from being overwhelmed and potentially losing data.

The fragmentation transcoder maintains a repository of fragmented components specific to a particular request. Transcoding Publisher uses specialized URLs for each fragment to ensure that the fragments are sent to the requesting device in proper order. Transcoded fragments are maintained in the repository in case they are requested again at a later time. Transcoding Publisher checks to see if the requested data has already been transcoded, and if it has, Transcoding Publisher sends the data to the device without having to process it again.

One clear benefit of the fragmentation transcoder's function is that developers who are creating custom transcoders for use with various devices, such as Smartphones, do not have to worry as much about the capabilities of the device. They can focus on the content that they want to generate and let Transcoding Publisher handle the process of chunking and sending the data.

HTML DOM Generator

The HTML DOM generator is a transcoder that converts an incoming HTML document into a Document Object Model (DOM) form so that it can be available for other transcoders that need to manipulate the document, such as text clippers. Typically, Transcoding Publisher automatically creates an internal HTML DOM as part of its built-in data transformations when a different output format, like WML, is required. However, in these cases, text clippers can only access the HTML DOM, in its original form, after the text transcoder has already worked with and modified the HTML DOM. At this point, it is too late to have any effect on the transcoding process.

The HTML DOM generator provides text clippers with a means of accessing the HTML DOM *before* any transcoding occurs. In this way, for example, a single text clipper could modify an HTML document, which could then in turn be transcoded into multiple formats and delivered to a variety of downstream devices. For more information about using the HTML DOM generator, read HTML Clipping.

Annotation Transcoder

The annotation transcoder, sometimes called the annotation engine, provides document clipping support for use with Transcoding Publisher's annotation language. Annotators are composed of a special set of XML tags that, when combined with an HTML source document, dictate which parts of the HTML document should be clipped. By using these annotators, you can perform document clipping without requiring a programmatic solution.

The annotation engine can process either external annotations, which reside in their own, separate file called an annotator, or it can process internal annotations, which are located directly in the HTML source document. Starting with a DOM form of the HTML document, the annotation engine examines the DOM and responds to the annotation tags by omitting or modifying elements in the DOM, including such functions as replacing images with text, extracting specific cells from a table, or reordering and modifying fields in a form. The resulting document can then be transformed to another format, like WML or HDML, and can take advantage of other transcoders, such as the fragmentation transcoder.

Note that if you intend to apply annotations to documents whose output content type is HTML, you must ensure that the HTML DOM generator is enabled.

For details on using annotations and the annotation language, read Using Annotation.

Other Transcoders

Some transcoders work with other products to render documents usable. For example, the Machine Translation transcoder works with WebSphere Translation Server. These transcoders require some configuration beyond being installed and enabled. They are described in Special Transcoders.

Preference Profiles

Transcoding Publisher uses *preference profiles* to represent the characteristics of devices and networks, and a default user profile to represent organizational policies. Each profile tells Transcoding Publisher how to treat documents that will be delivered to that device or over that network.

A preference profile can represent a particular type of device, such as a WorkPad(R), or a particular network type, such as a wireless network. For example, on a wireless network, you might want to specify that images should be converted to text links, allowing the page to load faster. On a device with limited processing capability, you might want to convert tables to lists to reduce the processing required to present the page.

When Transcoding Publisher processes a document, it selects a network profile and a device profile to apply to the document. The network profile, which is used only when you run Transcoding Publisher as a network proxy, is determined by the port on which Transcoding Publisher received the request. The device profile is determined by matching the value of the User-Agent keyword in the HTTP header of the document with the user-agent value associated with the profile.

Using the preference profiles, Transcoding Publisher determines the specific set of preference parameters that govern the processing of each document. Please refer to the *Administrator's Guide* for more information about how WTP uses profiles.

A profile does not have to include a value for each preference. If a value is not specified for a preference, Transcoding Publisher's preference aggregator will determine an appropriate value based on all the profiles involved in the transaction. If no value is supplied, the transcoder requesting the preference value will specify a default. The transformations that will be applied to the document are selected based on the combined preferences.

Transcoding Publisher provides profiles for several common pervasive devices and for several network types. There are default profiles to be used if none of the existing profiles matches the device or network being used. These profiles are listed in the tree area of the Administration Console, and you can modify some of the preferences through the Administration Console. For information about working with preference profiles through the Administration Console, refer to the *Administrator's Guide*.

You can use the Profile Builder to create new preference profiles or to modify existing profiles.

You can use the Transform Tool to preview the effect of your profiles and preference settings while the Administration Console is running. The Transform Tool takes an input document and, based on the active profile configuration, displays the transcoded output. This gives you the ability to view the transcoded data without requiring the various client devices for which the output is intended.

Profiles are stored in the etc/preferences subdirectory. There are subdirectories for device profiles, network profiles, and user profiles.

Preference Names

To see the names of preferences defined by WTP, view the API documentation for the PreferenceNames class. Each preference has a literal name, such as **screenCapability**, which is used in preference profiles, and a defined variable constant name, such as **SCREEN_CAPABILITY**, whose value is set to the literal name. Use the constant name, prefixed by the class name PreferenceNames, to access the preference value in a Java program. Use the literal name to access the preference value in a stylesheet or annotator. See the introduction to the PreferenceNames class for examples.

User Personalization

A preference profile may also represent a specific user. Transcoding Publisher will use the preferences of a specific user instead of the default user in every case, except when you have specified either:

- text links instead of images
- image transcoding

WTP does not provide a method for modifying user preferences. To use user preferences, you must provide the name of a user profile to WTP.

User Personalization with WebSphere Everyplace Suite

One way of providing user information to WTP is to deploy WTP within WebSphere Everyplace (R) Suite (WES). In WES, user preferences are managed by the WES Common User Preferences .jsp pages.

When your users update their user preferences, the preferences will be shadowed into the central directory. Entries will be in the form <username>@<realm><root>. When WTP receives a request and the header contains <username>@<realm> along with a device associated with that user, WTP will look in the central directory to match preferences to that user. If preferences are found, WTP will render the page applying the user, network, and device preferences. WTP caches the user preferences for 24 hours. If you change the preferences within that time, WTP must be restarted for changes to take effect.

<root> is the directory suffix, where WTP is installed.

<realm> includes the directory suffix, but may contain more information.

<username> is the user, in the realm, in the root.

When WTP is used with WES, WTP provides the implementation of Resource and ResourceDomain classes. For any other installation of WTP, you will need to create your own implementation of these classes. WES provides the X-IBM-PVC-User, X-IBM-PVC-SessionId, X-IBM-PVC-Device-Type in the HTTP header.

Settings that you can configure for user preferences through your central directory utility, such as Directory Management Tool for SecureWay Directory, are:

settingID=userPreferencesBackend is the classname of a class that implements the ResourceDomain interface

settingID=httpUserIdField is the HTTP header field specifying the user Id
settingID=httpSessionIdField is the HTTP header field specifying the session Id
settingID=userAndSessionExtractor is the classname of a class that implements the UserAndSessionExtractor interface

These SettingIDs are held in the central directory under the distinguished name, cid=userPreferenceSettings, cn=CommonServerModelSettings, sys=WTP40, sys=WTPReleases, sys=SDP, <your directory suffix> .

Providing your own user information

For WTP installations that are separate from WES, you will need to specify and store user preferences and create classes that implement the Resource and ResourceDomain classes. Information about the Resource and ResourceDomain classes is available from the javadoc.

When using WTP without WES, you will need to use a tool of your own making to edit user preferences. You must insert values for these fields into the HTTP header. If you already have this information in the HTTP header with names other than those above, you will have to change the SettingID values of the httpUserIdField and httpSessionIdField values in the central directory. Alternately, you can specify different header fields by implementing the userAndSessionExtractor interface. userAndSessionExtractor is a fully qualified classname.

If userAndSessionExtractor is not set to <null> then it must contain a class that implements the following interface:

public UserAndSessionIds getUserAndSession(String httpHeaderAsString, RequestEvent requestEvent);

which will provide the HTTP Header as a String and the entire RequestEvent object and get back a UserAndSessionIds object which will contain the user id and session id. Here is the UserAndSessionIds class which implements the following interface:

```
public class UserAndSessionIds
{
public UserAndSessionIds(String userId, String sessionId);
public String getUserId();
public String getSessionId();
}
```

Administrative Services

Management of Transcoding Publisher and configuration of the transcoding server is performed through the Administration Console. Using the console, you can register new transcoders, stylesheets, annotators, and preference profiles, and manipulate their properties. You can control message and trace logging and other server characteristics. If you use a central directory to store server models, you can work with server models through an Administration Console.

For more information about using the Administration Console, refer to the *Administrator's Guide*.

You can also perform administration of WTP resources by using XML Configuration.

Adding Transcoders to Transcoding Publisher

Because Transcoding Publisher's framework is built around a pluggable architecture, the capability of the transcoding server can be easily extended to support new data formats or to meet special transcoding needs. Transcoding Publisher supports transcoders written either to the Web Intermediaries (WBI) Version 4.5 API or to the Java Servlet Version 2.1 API. Referred to as MEGs and MEGlets, respectively, these transcoders can be used to edit requests flowing through the framework, to generate a response, to edit the response as it flows back to the client, and to the monitor the response.

Determining Your Approach

Deciding which API to use depends on the requirements of your transcoder and the environment in which it is intended to run.

Because the Servlet API is well understood and supported in a number of environments, (such as WebSphere Application Server and Apache), transcoders written to this API can be easily ported to other environments. Similarly, transcoders written for these other environments can be brought into Transcoding Publisher with few modifications, and they will be able to take advantage of other IBM-supplied transcoders and services, like preference aggregation.

Whereas servlet-based transcoders provide for portability across environments, transcoders using the WBI API can leverage WBI's ability to provide greater control over request processing and access to information in the original request (a MEGlet can access request information, but if another MEG or MEGlet upstream has modified the request, the MEGlet can only see the modified version of the request). In addition, WBI-based transcoders provide for more flexibility in how the transcoder's files can be packaged, in that you can include multiple MEGs in the same transcoder. Packaging the Transcoder Files contains more information on the packaging differences between WBI plug-ins and MEGlets.

However, if the portability or environmental advantages of the Servlet API are suited to your transcoder, this section describes the steps required to build and deploy a MEGlet in Transcoding Publisher. Because the concepts of WBI and its MEG-based transcoders are important to the development of MEGlets, it is recommended that you also read WBI Basics, if you are not familiar with WBI's concepts.

WebSphere Application Server Restriction

When you deploy WTP as a filter in WebSphere Application Server (WAS) environment, there is a restriction that you should take into consideration when writing your own transcoders. The servlet that generates content in WebSphere Application Server runs *before* Transcoding Publisher. This means that request editors and generators, whether WBI plug-ins or MEGlets, cannot affect the input that Transcoding Publisher sees. Although request editors can see the headers and URL of the original request, the POST data in the original request is retained by the WebSphere servlet and inaccessible to any request editors. This is not to say that you should not use request editors or generators in the WebSphere Application Server environment, but rather that if you choose to, you will have to abide by this restriction.

Response editors and monitors will prove most useful in this environment, as they are not as likely as request editors and generators to require access to information in the original request that the WebSphere servlet has suppressed.

WebSphere Edge Server Restrictions

When you deploy WTP as a plugin in WebSphere Edge Server Caching Proxy, remember these two restrictions:

- Request editors cannot change the URL of a request.
- POST data from the original request is not available to request editors or generators.

Other Resources

The Java Servlet API is both widely supported and widely documented. A good starting point for information about servlets is Sun's Web site. In particular, the servlet technical resources page provides articles and tutorials on a number of topics, as well as a list of third-party resources such as books and other Web sites. The Java technology zone of IBM's developerWorks Web site also has a variety of resources, downloads, and news items about Java and servlets.

If you think your development needs are such that the WBI API is a more appropriate approach for your transcoder, you can refer to the WBI Development Kit for detailed information on the API and how it can be implemented.

Adding a Transcoder

Adding a new transcoder requires the following steps:

1. Write a transcoder and compile it against either the Servlet 2.1 library or the WBI API, depending on whether you are creating a MEGlet or a WBI plug-in.
2. Specify the transcoder's configuration information.
3. Package the class files and configuration files in a Java Archive (JAR) file.
4. If your transcoder requires its own preference profile, create the profile.
5. Register the transcoder and the preference profile (if necessary) with Transcoding Publisher.

Writing and Compiling the Transcoder

MEGlets

As indicated above, MEGlets can perform any number of functions, singly and in conjunction with other MEGlets through chaining. You can create the MEGlet to act as a request editor, monitor, response editor, or generator. For examples of how MEGlets can be implemented as generators, editors, and monitors, Transcoding Publisher includes several samples that you can experiment with and modify for your own purposes.

The Servlet 2.1 library is provided with Transcoding Publisher as <*install_path*>/lib/servlet.jar, where <*install_path*> indicates the directory where you installed

Transcoding Publisher. Alternately, you can obtain the library directly from Sun. To ensure that your code compiles properly, be sure to add this library to your classpath. For example, on Windows NT you might specify:

javac -classpath "%classpath%;C:\Program Files\IBMTrans\lib\servlet.jar" MyMeglet.java

WBI Plug-ins

Transcoders written to the WBI API are composed of one or more Java class files, one for each MEG in the plug-in. Detailed information on creating WBI plug-ins is provided in the WBI Development Kit, along with sample code and API documentation.

To compile your WBI plug-in, ensure that the Transcoding Publisher installation directory is included in your classpath. For example, on Windows NT you might specify:

javac -classpath "%classpath%;C:\Program Files\IBMTrans" MyMeg.java

Naming Your Transcoder

Because transcoders used with Transcoding Publisher must have unique class names, IBM recommends conforming to Java's name space approach to help ensure that naming conflicts between transcoders do not occur. Transcoding Publisher organizes class files and property files relative to the addedPlugins and etc/plugins directories, respectively. If hierarchical names are not employed, the possibility of naming conflicts increases as more transcoders are added.

By following a name space approach with your transcoders, you minimize both the chance of naming conflicts with other transcoders and the possibility that your transcoder might be inadvertently replaced during registration. For example, Transcoding Publisher places all the class files for new transcoders under the addedPlugins directory. If your class is named com.myCompany.myPackage.myMeglet, the directory structure under addedPlugins will reflect the package name and help ensure that your class name does not conflict with others.

For details on how to package your transcoders files for proper registration with Transcoding Publisher, refer to Packaging the Transcoder Files.

Specifying Transcoder Configuration Information

With Transcoding Publisher, configuration information for the transcoder is specified in one or more property files. This section describes the property files that are associated with both MEGlets and WBI plug-ins.

MEGlet Property Files

Once you have written the MEGlet, there is still configuration information required by Transcoding Publisher to indicate such things as what type of MEG the MEGlet represents or the circumstances under which the MEGlet will be triggered. This information is defined in a property file. The contents of the Snoop.prop file, available as part of the Snoop sample, illustrates what can be included in a property file:

```
Class = Snoop
Name = Snoop
MEGType = generator
Condition = path = */Snoop
Priority = 10
InitParameter = [logging = true, level = 5]
Description = Header/attribute snoop servlet
```

A complete list of allowed properties and their values is described below.

Property	Description
Class	The name of the class that implements the MEGlet, including the package name, if you specify one. For example, Class = com.ibm.transform.toolkit.TransformTool. Refer to Naming Your Transcoder for information on naming and avoiding conflicts.
Name	The name that the MEGlet will be known by in Transcoding Publisher. This is used by the RequestDispatcher class and by the Administration Console. It must be unique. For example, /servlet/myServlet.
MEGType	The role this MEGlet plays when involved in transcoding transactions. The possible values are: • generator • editor • requestEditor • monitor
Condition	A Boolean condition that, when true, causes this MEGlet to be triggered. For example, with the Snoop MEGlet defined by Snoop.prop above, the MEGlet will be invoked if the path in the HTTP request ends with the string "/Snoop". Details on the format for these conditions are provided in Setting Triggering Conditions.

MonitorPosition	For MEGlets acting as monitors, this property specifies the location of the monitor in the response editor chain. Possible values are: • START - the MEGlet runs before the response is edited (right after the generator runs, producing the response) • END - the MEGlet runs after all editing is complete • EDITOR name - the MEGlet runs after a particular editor servlet, with name identifying the editor servlet.
Priority	The ordering priority relative to other MEGlets. The allowable range for priority is between 100 and 1, with 100 indicating the highest priority and 1 indicating the lowest. If the triggering conditions of more than one MEGlet match the request, the MEGlet with the highest priority is invoked first. The sequencing for multiple MEGlets is the same as that for MEGs, and both MEGlets and MEGs can interoperate on the same request.
InitParameter	Initialization parameters to be supplied to the MEGlet. The format for the parameters is [name = value, ...], with <name, value> pairs separated by commas. Use successive commas (,,) to embed a comma in a parameter. These parameters are specific to your code and can be whatever is meaningful to your MEGlet.
Description	Descriptive text that identifies the MEGlet. This information is displayed in the Administration Console after the transcoder has been registered. You can use whatever description you wish.

WBI Plug-in Property Files

Typically, there are two types of property file included with a WBI plug-in:

- A property file for the plug-in itself, which is used when the transcoder is registered with Transcoding Publisher.
- A property file for each MEG in the plug-in. This file defines information specific to the MEG, such as triggering conditions and priority value.

Note: If your WBI plug-in is composed of only one MEG, you can combine the contents of the two property files described above into a single property file. Read MEG Property Files to understand how references to the property file can be specified in the MEG's source code.

Plug-in Property Files

For an example of the plug-in property file, you can look at the IBMStockClipper.prop file that is included with the IBMStockClipper sample in the toolkit:

```
#Properties of IBMStockClipper sample plugin
Class=IBMStockClipper
Description=Performs text clipping on IBM Stock Page transcoded to WML
DescriptiveName=Text-Based WML Text Clipper for IBM Stock Page
Major=1
Minor=0
```

The following list describes the properties that should be in the file.

Property	Description
Class	The name of the class that implements the transcoder, including the package name, if you specify one. For example, Class = com.myCompany.myPlugin. Note that the ".class" extension is not included.
Description	The value used for the Description field in the Administration Console. This value will be used when the transcoder is registered and can be changed at that time if desired. This property is optional.
DescriptiveName	The name used to identify this transcoder in the Administration Console. This value will be used when the transcoder is registered and can be changed at that time if desired. This property is optional.
Major	The major portion of the plug-in's version information, specified as an integer. For example, if the property file supports Version 1.0 of the transcoder, the major portion of the version is "1." This property is required.
Minor	The minor portion of the plug-in's version information, specified as an integer. For example, if the property file supports Version 1.0 of the transcoder, the minor portion of the version is "0." This property is required.
ClasspathDependencies	A list of external jar files, or directories containing class files, required by this plug-in. You can specify each entry as a URL or a file path. The URLs and file paths can point to directories, jar files, or zip files. Relative file paths are relative to the WTP install path. Multiple entries are separated by plus signs (+) unless you specify a different ClasspathSeparationChar.

ClasspathSeparationChar	If you need to change the character used to separate entries in the ClasspathDependencies list (for example, if you have a URL that includes a plus sign), use this property to specify a different separation character.

When you package this file into a JAR file, ensure that it is the first property file in the JAR. When Transcoding Publisher registers a transcoder, it assumes that the first property file it encounters is the plug-in property file. See Packaging the Transcoder Files for more information.

MEG Property Files

In addition to the plug-in property file, you can also include property files for each MEG in the plug-in. Generally, these files are used to specify the triggering conditions for the MEG, as well as other information such as the MEG's priority setting. Although the WBI API does provide the capability to specify this information directly in the MEG's source code (with setCondition() and setPriority(), for example), we recommend using a property file for its flexibility. You can make changes to the MEG's triggering conditions or priority without recompiling the source.

The following example shows how MEG configuration information can be specified:

```
Name=IBM Transcoding Publisher Text Engine Text-Based IBM Stock Clipper
Condition=(url=*www.ibm.com/ibm/stock*) & (content-type=text/vnd.wap.wml)
Priority=3
```

The following list describes the properties in the file.

Property	Description
Name	The name of the MEG as it is known to Transcoding Publisher. This is primarily used in tracing information. Refer to Naming Your Transcoder for information on naming and avoiding conflicts.
Condition	A Boolean condition that, when true, causes this MEG to be triggered. Details on the format for these conditions are provided in Setting Triggering Conditions.
Priority	The ordering priority relative to other MEGs. The allowable range for priority is between 100 and 1. Higher priority numbers are given precedence over lower priority numbers. If the triggering conditions of more than one MEG match the request, the MEG with the highest priority is invoked first.

When you write your MEG, you can point to the appropriate property file in the source code. One way to do this is to define the file location in a variable, such as:

```
private static final String PROPERTY_FILE = "plugins/myCompany/myPackage/MyMeg";
```

Then you can load the property file with the setup() method:

```
setup(PROPERTY_FILE);
```

Note that Transcoding Publisher installs property files under the plugins directory of the /etc tree, according to the path structure specified in the transcoder's JAR file. As shown above, when referencing a property file from the within a MEG, it is not necessary to include the "/etc" portion of the path or to specify the ".prop" extension on the file (MyMeg.prop, for the above example).

Custom Properties

If you have other properties that are specific to a MEG but are not related to WBI properties, you can also include them in the MEG's property file. However, if you choose to do this, the MEG that requires the properties must open the property file as a com.ibm.wbi.persistent.Section object and access the additional properties itself.

Setting Triggering Conditions

A MEGlet or MEG (in a WBI plug-in) is selected to handle a request based on its condition, as defined by the Condition property in its configuration (Prop) file. At most, one generator will produce content in response to a request. Generators whose condition matches the request will be invoked in priority order until one produces a response. If no generators produce a response, a default generator forwards the request on to the server specified in the request.

MEGlet Notes: A generator can throw a ServletException to indicate that it does not wish to handle the request. Request and response editor MEGlets are also invoked based on their condition, in priority order, with each one acting in the manner of a chained servlet to filter the headers and data in the request or response. Monitors are associated with the response chain, and can be run (again based on condition) at various points in the chain. The output stream is not available to Monitor MEGlets.

Conditions are Boolean expressions that test the HTTP request or response headers. The following keywords are defined:

- **method:** The request method. For example, GET.
- **protocol:** The request protocol. For example, HTTP.
- **query:** The request query. For example, ?v=9&q=code+fixes&x=9&y=12.
- **user-agent:** The user-agent field of the request. For example, Mozilla/1.1-(compatible;-MSPIE-2.0;-Windows-CE).
- **host:** The hostname of the server. For example, www.ibm.com.
- **path:** The path in the URL. For example, /network/mobile/index.html.
- **url:** The fully qualified URL. For example, http://www.ibm.com/network/mobile/index.html.
- **port:** The local port. For example, 8080.
- **content-type:** The content type of the response. For example, text/html.

- **responseCode:** The response code. For example, 200 or 404.
- **server:** The server software that generates the response. For example, Apache/1.3.4.
- **x-ibm-wtp:** Whether to transcode this document. This keyword provides the capability to designate documents that should not be transcoded. It should be checked in the conditions for every document editor. See Canceling Document Transcoding for more information.

Keywords are followed either by an equals sign (=), which indicates that the argument is case-sensitive, or by a tilde (~), which indicates that the case is ignored. After the equals sign or tilde is a single argument. The argument can be grouped into clauses using parentheses to make a Boolean expression. To negate an argument, precede it with an exclamation point (!). The operators AND and OR (& and | , respectively) can be used between clauses. Arguments can include:

- Asterisks (*) used as wild cards. An asterisk must be before or after the argument. They may not be embedded within the argument.
- %true% used to indicate that all values for the given rule are valid.
- %false% used to indicate that no values for the given rule are valid

For example, to register an editor that triggers when a response is issued for:

- Any page with a content type that contains image or the path (file name) ends with .jpg or .gif, regardless of the case of the extension:

  Condition = (content-type=*image* | path~*.jpg | path~*.gif)
- Any text/html page that has a response code in the 400's (invalid page):

  Condition = ((content-type=text/html) & (responseCode=4*))
- Any text page (HTML, XML, etc.) that has a response code other than the 400's:

  Condition = ((content-type=text*) & !(responseCode=4*))
- Any page originating from the support section of the IBM Web site:

  Condition = url~*www.ibm.com/support*

If you want the transcoder to be triggered for every request, regardless of its content or source, you can use the following:

Condition = %true%

However, because the use of %true% in this way can degrade performance, we recommend that it only be used when other, more specific triggering conditions cannot be specified.

Notes on Using the Query Keyword: When specifying a triggering condition based on the query keyword, put the value between quotes to ensure that the value is correctly parsed. For example:

Condition = query="?v=9&q=code+fixes&x=9&y=12"

Because the question mark (?) is part of the query string, it must be included in the value for the condition, as in query~"?TICKER=IBM". However, if you prefer, you can use the wild card character (*) instead, as in query~"*TICKER=IBM."

Canceling Document Transcoding

A content provider or application programmer might wish to specify that a document should not be transcoded. This might be because the application program tailors the document for different devices and does not want it transcoded or because the provider views transcoding as a copyright infringement. The keyword x-ibm-wtp can be used to specify that a document should not be transcoded. Adding x-ibm-wtp=no to the header of a document will cause the document editors provided with Transcoding Publisher not to be selected to process the document.

This exclusion is accomplished by adding !(x-ibm-wtp~no)to the conditions for each document editor. For example, you might specify:

Condition = ((content-type=text/html) & !(x-ibm-wtp~no))

We recommend that you add this check to the conditions for any document editor you create. Request editors and generators do not check this keyword.

By writing a custom transcoder, you can add x-ibm-wtp=no to the header of a document on which you do not want any further transcoding to occur. Two samples are included with Transcoding Publisher's toolkit to demonstrate how this could be done:

- NoOpTranscoder, which is a WBI plug-in. This sample is located in <*install_path*>/toolkit/megs/NoOptranscoder and includes a JAR file (NoOpTranscoder.jar) you can use to install the sample and the Java source code.
- NoOp, which is a MEGlet. This sample is located in <*install_path*>/toolkit/meglets/NoOp and includes a JAR file (NoOp.jar) you can use to install the sample and the Java source code.

Packaging the Transcoder Files

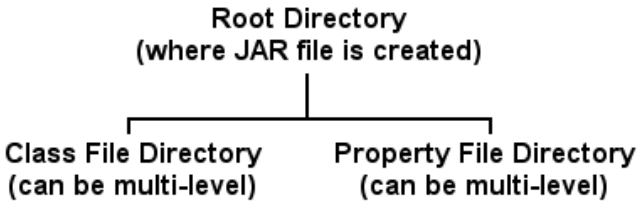
In order to make your transcoder available to Transcoding Publisher administrators, you must package the class files and property file associated with the MEGlet or WBI plug-in in a JAR file. Use the normal Java class directory hierarchy rules for adding classes that are part of a package to the JAR.

Transcoding Publisher requires that each JAR file contain the files associated with only one MEGlet. If you have more than one MEGlet, you will need to create a separate JAR file for each (as with the HelloWorld sample).

Organizing Your Files

When an administrator registers the transcoder with Transcoding Publisher, the Administration Console will unzip the contents of the JAR file and install the transcoder. As described in Naming Your Transcoder, the transcoder's class files will be stored under the <*install_path*>/addedPlugins directory, where <*install_path*> indicates the directory where you installed Transcoding Publisher. Property files for the transcoder will be stored under the <*install_path*>/etc/plugins directory.

To ensure that your transcoder's files are installed according to the hierarchical naming convention you have selected, the contents of your JAR file should be organized a certain way. Before you create your JAR file, organize your files according to the following structure:



When the transcoder is installed, the directory structure you specify for the class files will be appended to the <*install_path*>/addedPlugins directory, while the directory structure you specify for the property files will be appended to the <*install_path*>/etc/plugins directory. In addition, the directory structure for the property files is reflected in the Administration Console.

Note that there is no reason you could not use the same directory path for both the class and property files when building the JAR file. It simply depends on what structure you require to provide an appropriate level of differentiation between your transcoder and those of other developers.

If your transcoder requires additional files, for transcoder-specific information or other data, these should be put in the same directory as the class file and included in the JAR with similar path information.

The examples below describe how to package transcoder files for both MEGlets and WBI plug-ins.

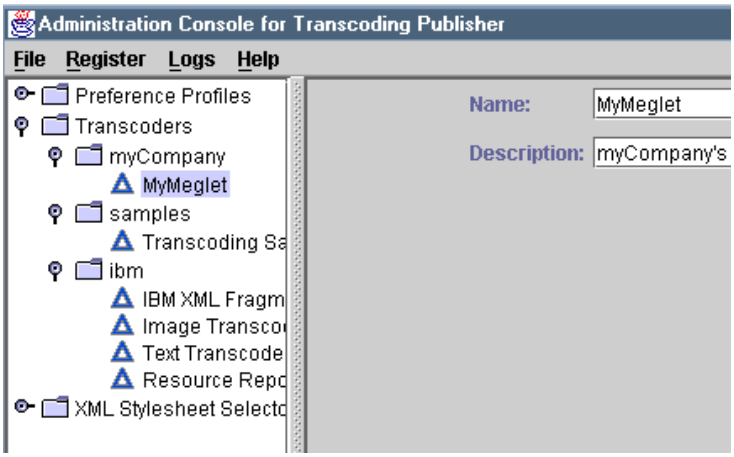
Example: Packaging a MEGlet

Suppose you have created a transcoder class called MyMeglet.class and you want to install it in a directory structure according to your company name. This will separate your company's transcoder from others and display it in the Administration Console under its own node of the Transcoders tree.

For our MyMeglet example, from the current directory, you might create a com/myCompany/myPackage directory path and put MyMeglet.class in it. Note that because addedPlugins is in the classpath, the package for this would be com.myCompany.myPackage. For the property file, you might create a myCompany directory under the current one. Then you could create the JAR file in the current directory, reflecting this organization by specifying

jar -cvf MyMeglet.jar com/myCompany/myPackage/MyMeglet.class myCompany/MyMeglet.prop

After registering MyMeglet with the Administration Console, the class file is stored in addedPlugins/com/myCompany/myPackage/MyMeglet.class, and the directory path used for the property file is reflected in the folder organization of the Transcoders tree, as shown in the following screen fragment:



You can look at the JAR files included with the toolkit samples for examples of different ways of packaging MEGlet files. For more information about creating JAR files or using the Java Archive Tool (the jar command) in the JDK, you can refer to The Java Archive (JAR) File Format.

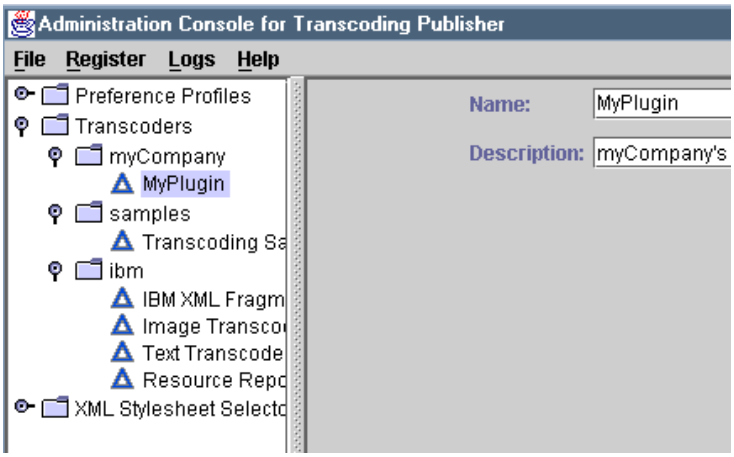
Example: Packaging a WBI Plug-in

Suppose you have created a WBI plug-in consisting of a class called MyPlugin.class and another class called MyMeg.class. You want to install the plug-in in a directory structure according to your company name. This will separate your company's transcoder from others and display it in the Administration Console under its own node of the Transcoders tree.

For our MyPlugin example, from the current directory, you might create a com/myCompany/myPackage directory path and put MyPlugin.class and MyMeg.class in it. Note that because addedPlugins is in the classpath, the package for this would be com.myCompany.myPackage. For the property files, you might create a myCompany directory under the current one. Then you could create the JAR file in the current directory, reflecting this organization by specifying

jar -cvf MyPlugin.jar com/myCompany/myPackage/MyPlugin.class com/myCompany/myPackage/MyMeg.class myCompany/MyPlugin.prop myCompany/MyMeg.prop

After registering MyPlugin with the Administration Console, the class files are stored in addedPlugins/com/myCompany/myPackage directory, and the directory path used for the property files is reflected in the folder organization of the Transcoders tree, as shown in the following screen fragment:



You can look at the JAR files included with the IBMStockClipper sample for examples of different ways of packaging transcoder files. For more information about creating JAR files or using the Java Archive Tool (the jar command) in the JDK, you can refer to The Java Archive (JAR) File Format.

Adding a New Preference Profile

Depending on what your transcoder is designed to do, you might find that you need to take advantage of Transcoding Publisher's preference profile support. For example, your transcoder might require an entirely new profile composed of known preferences but with values and settings not supported in existing profiles.

For information about how Transcoding Publisher uses preference profiles and the different ways that you can customize them, read Working with Preference Profiles.

Registering the Transcoder with Transcoding Publisher

The final step in making a MEGlet or WBI plug-in ready for use is to make the transcoder files available to Transcoding Publisher. By doing so, you can verify that the transcoder functions as intended before you distribute it to others for their own use.

To install the transcoder, you will need to use the Administration Console. The console provides wizards for registering new transcoders and preference profiles and also provides access to Transcoding Publisher's tracing functions. For details on using the Administration Console, refer to the Administrator's Guide.

Debugging and MEGlets

To help with the debugging process when developing your MEGlet, Transcoding Publisher provides several means of accessing information about how your MEGlet is running. These include the logging capability of the Servlet API, the tracing function of Transcoding Publisher, and the Request Viewer tool.

Using Logging and Tracing

To record information about how your transcoder is functioning, particularly when it encounters errors, you can use the ServletContext.log() methods provided by the Servlet API to write information to a servlet log. As described in the Servlet API, you can log a given message or the stack trace and message for an exception, depending on which method you choose.

The ServletContext.log() method is defined as generating output to a log specific to the engine. In the case of Transcoding Publisher, messages logged with ServletContext.log() are routed to the Transcoding Publisher trace files, when tracing has been enabled on the transcoding server. Transcoding Publisher supports three levels of tracing that provide progressively more detail. MEGlet messages are included at the lowest (and least detailed) level.

To view the messages generated by your MEGlet, use the Administration Console to enable tracing and specify the **Low** level of trace. You can then either use the view function of the Console or open the trace files with an editor. Details on using the Administration Console and Transcoding Publisher's tracing functions are provided in the *Administrator's Guide*.

Important Note on Tracing:

- For unusual situations where low-level tracing and the Request Viewer tool are insufficient for determining a problem with your MEGlet, medium-level tracing can be turned on. Medium-level tracing includes additional information regarding the MEGlet method calls that are invoked and their associated parameters. ***Medium-level tracing includes call-by-call tracing, as well as information for other components in Transcoding Publisher. Because it can severely degrade performance, it should only be used when absolutely necessary.***
- ***The use of high-level tracing should be reserved only for very special situations, as instructed by an IBM service representative.*** Because high-level tracing generates an enormous amount of information, the Request Viewer can scroll excessively as it attempts to display the trace information. If you need to examine the high-level trace information, open the trace files directly with an editor, rather than trying to view the information through the Request Viewer.

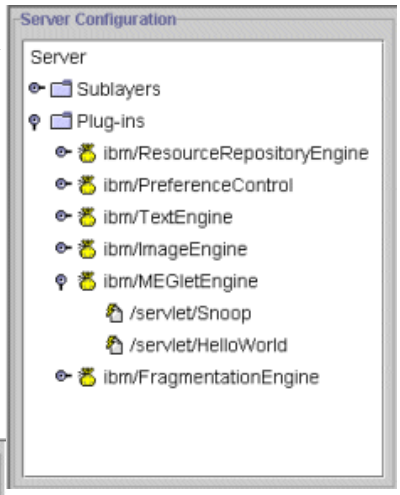
Using the Request Viewer Tool

The Request Viewer is a useful visual tool for monitoring the processing of your transcoder by the Transcoding Publisher transcoding server. This section describes some of the ways you might incorporate the Request Viewer into the debugging process. See Developer Tools for a more general discussion of the Request Viewer, along with some important considerations regarding the use of the tool in a production environment.

A variety of information is available from the Request Viewer that might be of interest when trying to determine whether your MEGlet was triggered, whether it ran properly, and what output it created (if appropriate).

Note: Although this section describes how to use the tool to monitor a MEGlet, the Request Viewer displays the same information for WBI plug-ins that are registered with Transcoding Publisher.

Viewing MEGlet Configuration Details



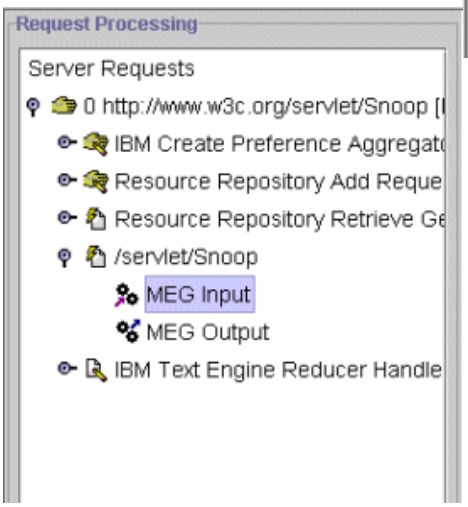
MEGlets that have been registered with Transcoding Publisher appear in the Server Configuration page under the IBM/MEGletEngine entry in the Plug-ins tree. For example, registering the Snoop and HelloWorld sample MEGlets results in a configuration similar to that shown in the screen fragment at right.

The configuration information specified in the MEGlet's property file is displayed in the Server Configuration Details pane of the Request Viewer when the MEGlet is selected from the list. This enables you to verify the triggering conditions for the MEGlet, which can be compared with the request information on the Request Processing page. In addition to the triggering conditions, the priority setting for the MEGlet is also available.

Viewing Logging Information

If you have enabled tracing to generate log messages, they will be displayed when the MEGlet runs in the Output-Messages pane at the bottom of the Server Configuration page. This is the same information that is available from the Administration Console, and the level of detail corresponds to the tracing level specified with the Console.

Monitoring Requests



The Request Processing page of the Request Viewer displays each request that Transcoding Publisher receives, including a sequential list of transcoders that act on the request. The screen fragment at right shows a number of Transcoding Publisher transcoders that are triggered when a request is received, such as the preference aggregator and several built-in request editors.

In addition to being able to see whether your MEGlet has been triggered for a particular request, the Transaction Header and Transaction Content panes of the page display the header information from the request and the content of the transaction, respectively. For example, in the case of the Snoop sample, the HTML code generated by the MEGlet is displayed in the Transaction Content pane as it is executed.

MEGlet Programming Notes

This section describes considerations to keep in mind when developing MEGlets.

Accessing Preference Information

The Servlet API allows attributes to be associated with a request through the ServletRequest object. When it receives a request, Transcoding Publisher adds additional attribute information, based on the preference profiles that are involved in processing the request. If your MEGlet needs to operate differently depending on the settings for a particular device or network, you can use Transcoding Publisher's preference aggregator to retrieve the desired preference values. This is done by calling the ServletRequest.getAttribute() method and specifying the preference name, with com.ibm.transform.preferences prepended to the name. This practice is in keeping with the Servlet API specification's position that attribute names follow the same, hierarchical convention as package names.

For example, if you needed to know the preference setting for whether JavaScript is supported by the target device, you might use something like:

Object value = request.getAttribute("com.ibm.transform.preferences.javaScriptSupported");

Similarly, you can use getAttributeNames() to retrieve an enumeration of all the current preference settings. In this case, each preference name is returned with the com.ibm.transform.preferences prefix. Note that although a preference might be returned, getAttributeNames() does not check to see if the preference does in fact have a value (unlike getAttribute()). The Snoop sample is an example of how getAttributeNames() can be used in this fashion.

Setting Request Properties with MEGlets

Due to the way the Servlet API is specified, servlets acting as request editors are not given access to the method line of the request, including information identifying the URL. This poses problems for request editors that need to modify certain request properties.

To address this limitation, Transcoding Publisher's MEGlet support provides a means for manipulating this information with the ServletRequest.setAttribute() method. The following attribute names are treated as special directives to Transcoding Publisher to modify the request properties:

- com.ibm.transform.request.URL (sets the URL)
- com.ibm.transform.request.method (sets the method)
- com.ibm.transform.request.protocol (sets the protocol)

For example, if your MEGlet needs to change the URL of the request, you can use something like:

request.setAttribute("com.ibm.transform.request.URL", "http://www.ibm.com");

Keep in mind that this function is specific to Transcoding Publisher and that servlets will not generally have this capability in other environments.

Servlet API Implementation Notes

The Java Servlet 2.1 API specification leaves many decisions on the details of supporting servlets up to the provider. This allows servlets to be used in many environments but can lead to questions about just what a method really does in a given environment. Here are specific points of clarification on the support provided by Transcoding Publisher:

- HttpServletRequest.getServletPath() specifies that it will return null if the match was made by some mechanism other than path match. Transcoding Publisher always uses a more general condition matching mechanism and currently always returns null here. This is only likely to be an issue when using getPathInfo().
- Both SessionContext and ServletRequest support an Attribute mechanism to allow access to additional server and request information and to allow information to be communicated between MEGlets. There is currently no additional information in SessionContext attributes beyond what MEGlets have added. ServletRequest attributes include both what MEGlets have added and attributes that have been defined by the preference aggregator, based on the incoming request and on preference files relevant to this request. For more information on using preferences, read Accessing Preference Information.
- ServletContext.getResource() and ServletContext.getResourceAsStream() interpret paths relative to the directory where Transcoding Publisher is installed. Note that the only resources currently supported are files.
- There is a single ServletContext covering all of the MEGlets registered with Transcoding Publisher. ServletContext.getContext() always returns this context, regardless of the path specified. On a related point, ServletContext.getRealPath() doesn't currently perform a translation but instead returns itself.
- The specification does not define header handling for "chained" MEGlets such as editor MEGlets in Transcoding Publisher. For compatibility with other servlet engines that support chaining, Transcoding Publisher leaves the responsibility for copying through headers to the MEGlet. Refer to the editor MEGlet (ReplaceBlink.java) supplied with the HelloWorld sample for an example of how this can be done.
- The implementation of sessions in Transcoding Publisher is cookie-based. Sessions last as long as the transcoding server and browser are up but do not survive either one being taken down (that is, neither client-side cookies nor server-side sessions are persistent). Note also that since cookies are, by definition, domain-based, sessions can span MEGlets in the same domain (the session support uses "/" as the path within a domain) and cover all MEGlets handling a particular request, but they do not extend beyond that.
- The SingleThreadModel interface is used to indicate that only a single thread may execute a MEGlet at a time. The current implementation provides this synchronizing access to a single instance of the MEGlet. Other MEGlets are multithreaded, as usual, and should be coded to support concurrent access.

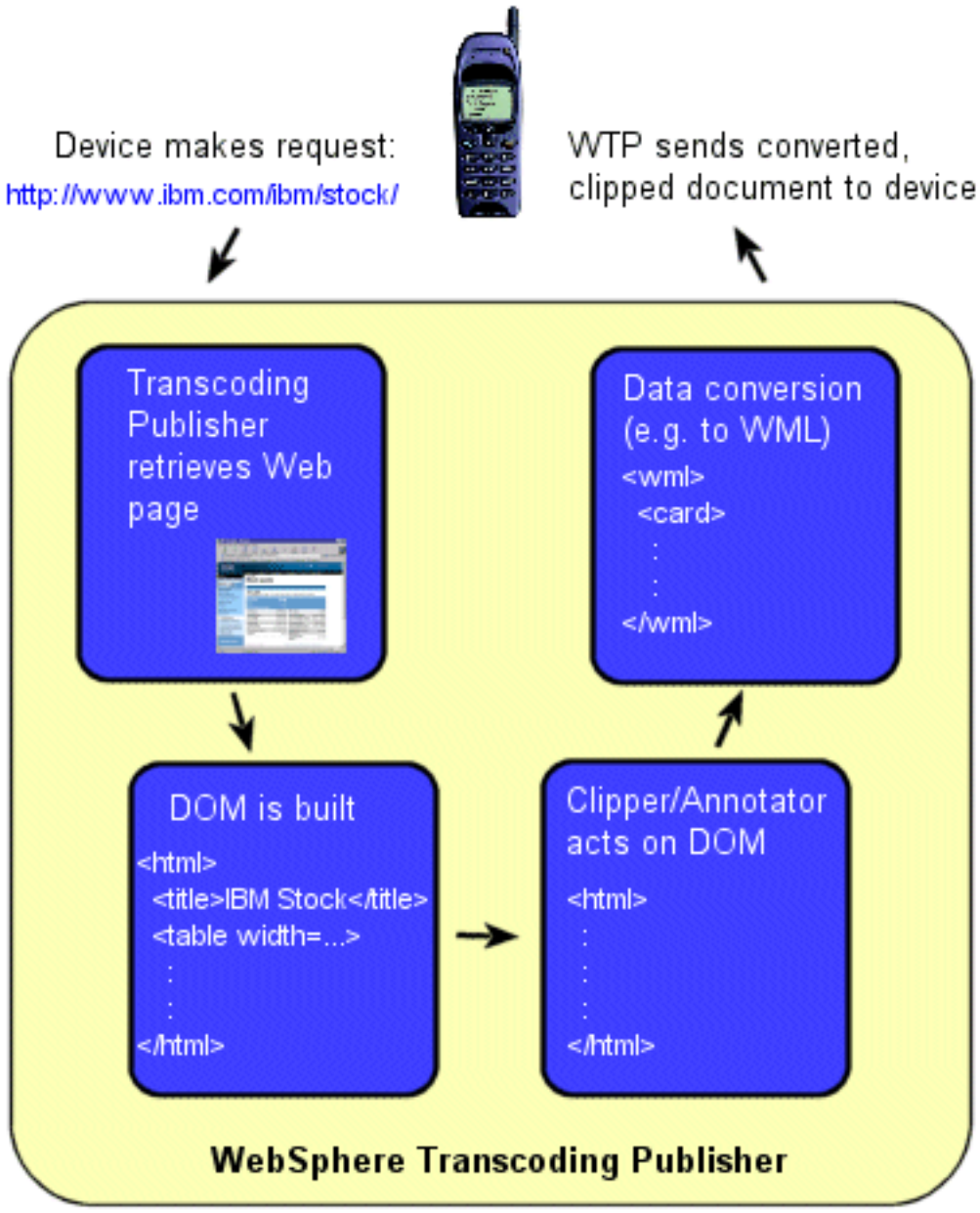
Character set and Java encoding

WTP 4.0 adds the com.ibm.transform.textengine.EncodingInformation object. This object can be used to query and update the MIME Character set and associated Java encoding String for a document. An EncodingInformation object will be created automatically when WTP initially processes a document. User-written MEGs, MEGlets, and TextClippers should use the EncodingInformation object to determine or update the encoding information. See the Javadoc for com.ibm.transform.textengine.EncodingInformation for more information.

Document Clipping

Although equipped to perform a number of common data transformations, Transcoding Publisher also provides extended function for transcoding data into other formats, whether that be other markup languages or tailored forms of the original data. With the rapid growth in the number of portable devices, including phones and personal digital assistants (PDAs), the ability to tailor content to these small displays is increasingly important.

Many Web sites and browser-based applications are designed for desktop displays that support large windows, high-resolution images, and thousands of colors. For users with portable devices, this often means a garbled interface or, more likely, an inability to access these sites and the information they contain. Transcoding Publisher addresses this problem by including several facilities for ***document clipping*** and ***data transformation***, as shown in the figure below.



Document clipping can be used to identify and extract specific portions of a document. In this way, you can keep those pieces of information that are important while discarding images or complicated markup that the client device is incapable of displaying. This simplifies what is being sent to the device while reducing the amount of data being transmitted, a particularly useful aspect when wireless devices are involved. Moreover, no changes to the source content are required.

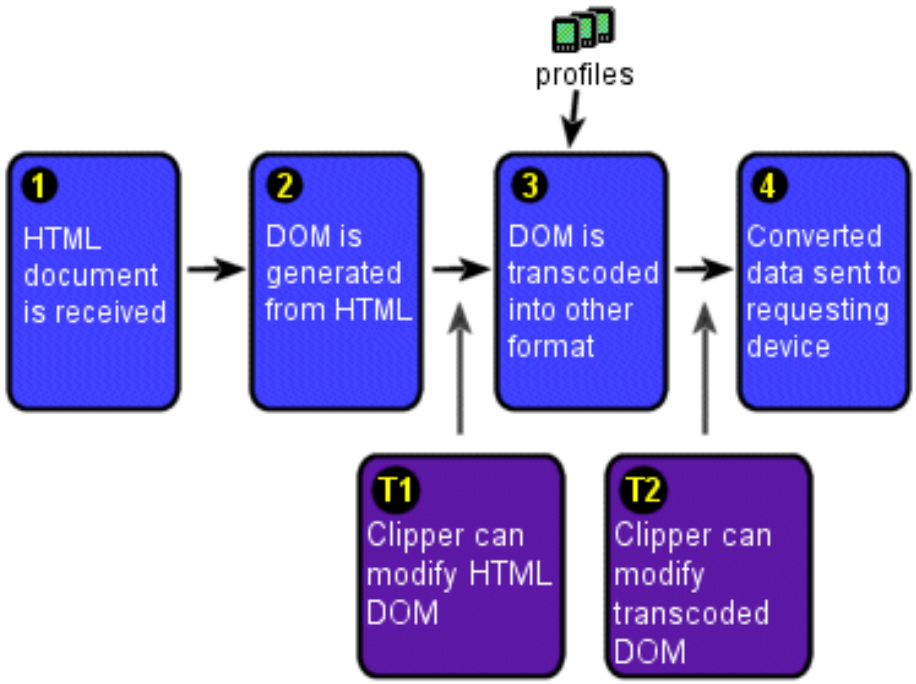
To accomplish clipping, Transcoding Publisher provides two approaches:

- Text clippers, which are custom transcoders written in Java
- Annotators, which are files that define clipping behavior through the use of an annotation language

Using Text Clippers

For those who are comfortable with Java or who require the flexibility and power available from a programmatic solution, Transcoding Publisher provides the capability of performing document clipping with custom transcoders called text clippers. A text clipper is a special WBI response editor that enables you to manipulate a text document, either in its text form or as a Document Object Model (DOM). The DOM defines the logical structure of an HTML or XML document and essentially provides a standard API for adding, changing, and deleting parts of a document.

Note: For further explanation of the Document Object Model and how it represents a document, please visit the World Wide Web Consortium Document Object Model site at http://www.w3.org/DOM.



Document clipping with text clippers. Transcoding Publisher receives an HTML file (1), and the HTML DOM Generator transcoder generates an HTML DOM from the document (2). Depending on the device profile associated with this request, the text transcoder converts the HTML DOM to another DOM of a different data format, such as WML (3). The transcoded data is then sent on to the requesting device (4). A text clipper could modify the HTML DOM directly before it is converted to another format (T1), or it could work instead on the modified DOM (T2).

Clipping by Text or by DOM

When working with a document that you want to clip, there are several approaches to handling the extracting of needed information from the document:

- Operate on the text form of the document with Java code (for example, with a response editor).
- Operate on the DOM with Java code (for example, with a response editor).
- Operate on the DOM with stylesheets (XML and XML-derived languages only). Note, however, that because information on how to apply stylesheets to handle this type of problem is more readily available, this discussion of text clippers, as well as the samples provided with the product, address the first two approaches described above.

In most cases, operating on the DOM form of a document is preferable to operating on its text form, provided that a useful set of functions to operate on the DOM are available. To support manipulation of a DOM, Transcoding Publisher includes such convenience functions in classes like TextClipper and DOMUtilities.

One of the challenges of operating on the text form of a document can be explained with a simple example. Suppose you have the following markup in your document:

** **
** Weather**
** **

If we search for Weather and extract our text from that point, we will have the following:

Weather
** **

which has a closing without a corresponding and a closing without a corresponding . Similarly, if we search for Weather and extract our text from that point, we will have the following:

Weather
** **

which still has a closing without a corresponding . Also, if we choose to search for something like Weather, we are relying on the fact that Weather will always be a child of a Bold (B) element.

By operating instead on the DOM form of the document, we can sidestep some of this complexity. For example, if we wanted to search the DOM for the occurrence of Weather above, we could use the findNodeMatchingPattern method that is part of DOMUtilities to locate the node containing the target text. In locating and working with the DOM node containing the target text, we avoid the problem of removing closing tags without their corresponding opening tags by removing the entire node, including the opening and closing tags.

Although clipping according to the DOM presents some advantages over clipping the text form of the document directly, we do not want to make too many assumptions about the type or variability of the source content. In some cases, parsing the text form of the document to extract the final document may be quite feasible. For that reason, the text clipper samples we provide demonstrate how to work with either the text form of the document or the DOM.

IBMStockClipper Samples

The IBMStockClipper samples are provided in the toolkit to demonstrate how a text clipper can be constructed and to provide a starting point for developers who wish to create their own text clippers. The IBMStockClipper is a response editor (implemented as a WBI plug-in) that takes the current IBM stock price from IBM's external Web site at www.ibm.com (converted into WML) and displays it on a WML device.

The HDMLIBMStockClipper samples are similar to the IBMStockClipper samples but operate on an HDML form of the data.

Because there are options regarding how you might want to operate on the data, the toolkit includes two variations of the text clippers: one for working with the DOM version of the document and one for working with the text version of the document. Each of these variations is referred to in various areas of this section to help illustrate the concepts being described.

IBMStockClipper.java and IBMStockClipperDom.java contain the source code for the WML samples. In addition to these files, the toolkit provides property files that define the plug-ins to Transcoding Publisher and a pair of batch files (mkscjar.bat and mkscdjar.bat) that demonstrate how the text clippers are packaged into JAR files.

The source code for the HDML samples is contained in HDMLIBMStockClipper.java and HDMLIBMStockClipperDom.java. The transcoder package files for these samples can be found in mkhscjar.bat

and mkhscdjar.bat.

To use the samples, register one of the stock clipper transcoders with the Administration Console and ensure that it is enabled. To see how the text clipper renders the page, request http://www.ibm.com/ibm/stock/.

Creating Your Own Text Clipper

To perform document clipping with Transcoding Publisher, you can create your own text clipper by extending the com.ibm.transform.textengine.mutator.TextClipper class. The TextClipper class provides simplified initialization and also provides some text processing utilities, as well as methods to get the input document as a String and another to set the content length in the HTTP header following the clipping process.

If you prefer, you can copy one of the example classes as a starting point and modify it to suit your needs. Note that each sample has a version that works on the DOM form of a document and a version that works on the text form. So, for example, if you wanted to write a text clipper that manipulates the DOM of a WML document, you would copy IBMStockClipperDom.java and make your own changes to this copy. Otherwise, you would copy IBMStockClipper.java.

Setting Up the Classpath

Before you compile your text clipper, ensure that the Transcoding Publisher installation directory and the following archive files are added to your classpath (note that all of the items listed here are relative to the installation directory):

- .\lib\xerces.jar
- .\lib\xalan.jar
- .\lib\log.jar
- .\lib\htmltemplate.jar
- .\lib\servlet.jar
- .\lib\bsf.jar
- .\lib\wtpcommon.jar
- .\lib\wtpserver.jar
- .\lib\wtpadmin.jar

Modifying the Property File

Because a text clipper is implemented as a WBI plug-in, there can be one or more property files associated with the transcoder that you will also want to adapt for your own text clipper. In the case of the samples provided with Transcoding Publisher, each sample has one plug-in property file containing information that identifies the plug-in during registration and that indicates the conditions under which the text clipper is triggered.

For a DOM-based text clipper, you would adapt toolkit\textclippers\IBMStock\samples\IBMStockClipperDom.prop, which contains the following:

```
#Properties of IBMStockClipperDom sample plugin
Class=IBMStockClipperDom
Description=Performs text clipping on IBM Stock Page transcoded to WML
DescriptiveName=IBM Stock Page DOM-Based WML Text Clipper
Major=1
Minor=0
```

```
Name=IBM Stock Page DOM-Based WML Text Clipper
Condition="(url~*www.ibm.com/IBM/Stock*) & (content-type=text/vnd.wap.wml)"
Priority=3
```

In addition to ensuring that the Class property matches your text clipper's class name, you will also want to change the Condition property, which specifies the conditions under which your plug-in will be triggered. For example, if the URLs you want your text clipper to process contain www.myURL.com, you would use the following value:

```
Condition=(url=*www.myURL.com*) & (content-type=text/vnd.wap.wml)
```

You will also want to customize the other properties in this file to apply to your text clipper. For more information on these properties and how they are used, refer to WBI Plug-in Property Files.

Referencing the Property File in Your Source Code

For your text clipper to run properly, it must know where to find the property file that it is associated with. When you register a transcoder, the registration process places the property file included in the jar file into the /etc/plugins/ directory structure under the WTP installation directory. You should specify a path relative to /etc. The IBMStockClipperDom.java file handles this association with the following line:

```
private static final String SETUP_PROPERTIES =
"plugins/samples/IBMStockClipperDomEditor";
```

Your Java source file must point to your own property file. For instance, yours might look something like this:

```
private static final String SETUP_PROPERTIES = "plugins/myCompany/MyClipper";
```

Accessing the HTML DOM

If your text clipper runs after the text transcoder has converted an HTML DOM into some other type of DOM (e.g. WML), you can retrieve the original HTML DOM from within your TextClipper subclass as follows:

```
Document doc = (Document) getOriginalDOM(reRequest);
```

Typically, you will not need to access the original HTML DOM, but only the converted DOM. You would only need to access the original HTML DOM if you needed access to information that was deleted in the process of transcoding the HTML into another format.

Accessing the Generated DOM

Transcoding Publisher's text transcoder can convert the HTML DOM representing the incoming document into a DOM of another format, such as WML. Within your TextClipper subclass, you can retrieve the DOM generated by the text transcoder as follows:

```
Document doc = (Document) getTranscodedDOM(reRequest);
```

In the process of converting HTML data into WML, HDML, or Compact HTML, the text transcoder performs a number of transformations on the elements in the HTML DOM. These transformations vary depending on the target format. For example, the current HTML-to-WML transcoding done by the text transcoder generates a WML element containing a single CARD element, which in turn contains a single Paragraph (P) element. The rest of the document is contained under the P element.

Note: To address the content-size constraints of some devices (for example, the 1400-byte limit of some Nokia phones), Transcoding

Publisher provides the fragmentation transcoder for segmenting data into multiple chunks. Refer to Text Clippers and the Fragmentation Transcoder for more information.

For more specific information about how HTML elements are transcoded into the other formats, refer to Data Transformation. This section takes each HTML element and explains how it is handled in the new format.

Working with the Generated DOM

After you have retrieved the generated DOM, you can then manipulate its elements to extract the information you want. If you look at the IBMStockClipperDom.java file, you will notice that it uses methods in the DOMUtilities class to work with the DOM, along with basic DOM manipulation functions provided by the org.w3c.dom classes. With these methods you can perform a range of functions, including:

- Searching for a node containing a particular string
- Finding a given type of node
- Moving children nodes from one parent node to another
- Retrieving and replacing attributes of nodes
- Deleting a node and promoting its children to its place in the DOM

For more detailed information on these methods and what they do, refer to the text-clipper API information provided with the toolkit.

As an example of how a text clipper can operate on the generated DOM, consider IBMStockClipperDom.java, which clips a WML DOM created by the text transcoder. The basic technique is to start with the paragraph (P) node and look for direct descendants of the P node that either contain or have children that contain text strings indicating points in the DOM where clipping is to be performed. The children of the P node that precede the child node under which we found the desired text are deleted. After the process is completed, we use the DOM to create a DomMegObject as follows:

```
WMLPrinter printer = new WMLPrinter();
DomMegObject domMegObject = new DomMegObject(doc, contentType, printer);
```

where doc is the DOM (beginning with the Document node) and contentType is the content type of the output, such as text/vnd.wap.wml for WML. Note that an appropriate DOMPrinter object is used for printing out a document, depending on the content type required. For example, for a WML document, you would use a WMLPrinter object, while for an HDML document, you would use an HDMLPrinter object.

After the DomMegObject is created, you can write it out with the writeOutput() method:

```
writeOutput(reRequest, domMegObject);
```

The DomMegObject approach uses support in WBI that allows objects (MegObjects, in this case) to be passed to downstream MEGs without having to write and read from streams. Writing out the DomMegObject is more efficient than writing out bytes to the output stream because the downstream MEGs, such as the fragmentation transcoder, will work with the DOM itself when possible, so writing to the output stream in this case is an unnecessary step. Note that downstream MEGs that wish to access an input stream rather than a MegObject can do so, because the MegObject supports accessing streams if that is desired.

If you prefer to output a String form of the document rather than a DOM, you should do so as follows:

```
writeOutput(reRequest, sOutputPage);
```

Note that both writeOutput() methods above also cause the content length in the HTTP response header to be set,

so a call to setContentLength() is not necessary.

Packaging Your Text Clipper

Because a text clipper is implemented as a WBI plug-in, which can contain one or more MEGs, there are several packaging steps that you need to complete before you can register the text clipper with Transcoding Publisher or make the text clipper available for distribution to others. Refer to Adding Transcoders to Transcoding Publisher for information on how to handle the property files for the plug-in and the MEGs, as well as how to create a JAR file that contains all the necessary files related to the clipper.

HTML Clipping

Although Transcoding Publisher's text clipping classes and DOM-based utilities are well suited to working on documents that have been transcoded into different formats, like WML or i-mode's Compact HTML, you can also use these same facilities to perform HTML clipping. By working on a DOM form of the HTML document, you can similarly add, change, and delete nodes in the DOM to create a customized version of the document.

Examples of situations where you might want to consider using HTML clipping could include:

- Transcoding Publisher's built-in HTML simplification is not sufficient for the needs of your target device. For example, suppose you are transcoding an HTML document for display on a device running the Windows CE operating system. HTML simplification can address many features of HTML that are not supported by such devices, but it does not account for some characteristics of the device, such as the size of the screen. You might elect to perform HTML clipping to further customize the document before it is transcoded and delivered to the device.
- You intend to deliver an HTML document to a number of different devices, which require some document clipping to be performed and different output formats. Rather than create a text clipper to customize the WML output, another one for the HDML output, and still another for the i-mode Compact HTML output, you could use a single text clipper to modify the HTML document once. Transcoding Publisher could then automatically transcode this customized HTML document into the required data formats.

Typically, Transcoding Publisher automatically creates an internal HTML DOM as part of its built-in data transformations when a different output format, like WML, is required. However, in these cases, text clippers can only access the HTML DOM, in its original form, after the text transcoder has already worked with and modified the HTML DOM. At this point, it is too late to have any effect on the transcoding process. To access the HTML DOM before any transcoding occurs, you need to ensure than one of Transcoding Publisher's special transcoders, the HTML DOM generator, is enabled. When the product is installed, the HTML DOM generator is automatically registered, but it is disabled by default.

HTML Clipping for HTML Output

For a text clipper that acts on an HTML DOM that is to be converted to another format, you will need to set the text clipper's priority so that it runs after the HTML DOM generator but before the text transcoder, which handles the tasks of HTML simplification and data format conversion. The HTML DOM generator runs at a high priority (92) when it is enabled, while the two text transcoder MEGs involved run at low priorities (ReducerHandler = 40 and HTMLHandler = 1). So for your text clipper, a priority value like 50 would ensure your transcoder is triggered in the proper sequence.

Note that if the HTML DOM generator creates a DOM, the internal DOM normally created for data transformation is not built. Instead, the DOM created by the HTML DOM generator is saved using WBI's MegObject support. Your HTML text clipper can then access the DOM through the MegObject object and pass it along to the text

transcoder in the same manner.

To access the HTML DOM, you can use the getTranscodedDOM() method described in Accessing the Generated DOM, as long as your clipper class is a subclass of TextClipper. Although at this point the DOM has not actually been transcoded, getTranscodedDOM() is used to access the DOM most recently stored by the previous MEG. You can then create a DomMegObject and write it out as follows:

```
HtmlPrinter printer = new HtmlPrinter();
DomMegObject domMegObject = new DomMegObject(doc,
TranscoderConstants.CONTENT_TYPE_HTML, printer);
writeOutput(reRequest, domMegObject);
```

HTML Clipping for HTML Output

For a text clipper that acts on an HTML DOM that will not be converted to another format, you will need to set the text clipper's priority so that it runs after the HTML DOM generator, which has a high priority (92). As long as your transcoder's priority value is less than that of the HTML DOM generator's, the text clipper will be triggered in the proper sequence.

If you want the text transcoder to be able to perform HTML simplification on the output from your text clipper, do not set the MegObject object as part of the RequestEvent. Instead use something like the following:

```
HtmlPrinter printer = new HtmlPrinter();
String outputPage = printer.printNodes(doc);
writeOutput(reRequest, outputPage);
```

Text Clippers and the Fragmentation Transcoder

Depending on the size of the document that your text clipper produces, the Transcoding Publisher fragmentation transcoder can be used to subdivide the output into chunks that the requesting device can handle. As implemented with the IBMStockClipper samples, the text clippers will run before the fragmentation transcoder, since the clippers are assigned a higher priority in their respective property files. This ensures that the text clipper receives an unfragmented document on which to work.

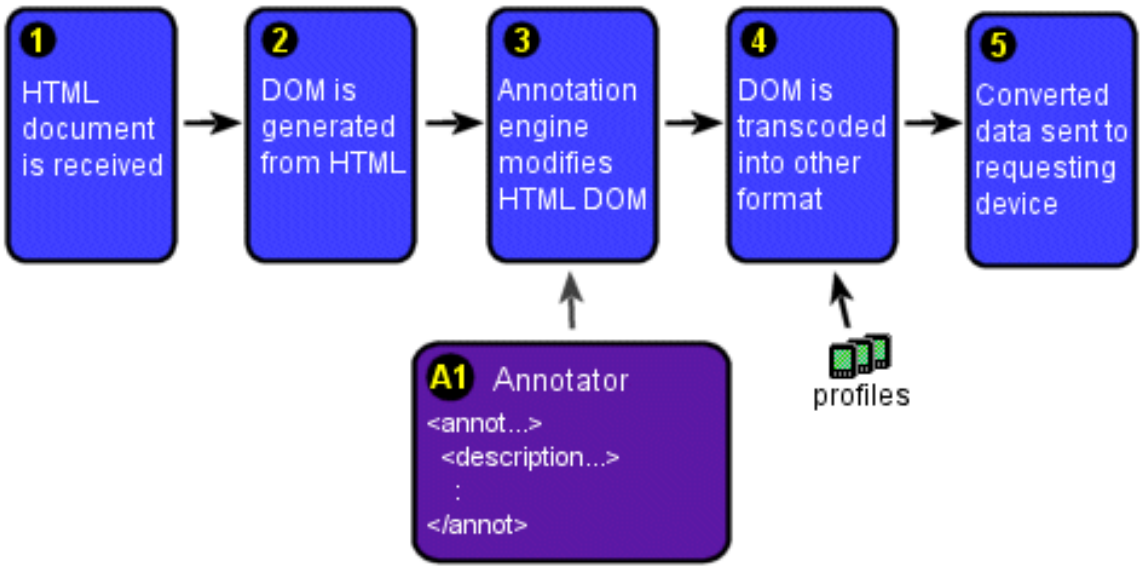
If the text clipper reduces the document to a size that fits within the constraints specified for the requesting device, the fragmentation transcoder will not alter the document. Otherwise, the document will be fragmented into chunks, which will be sent to the device in sequence.

Using Annotators

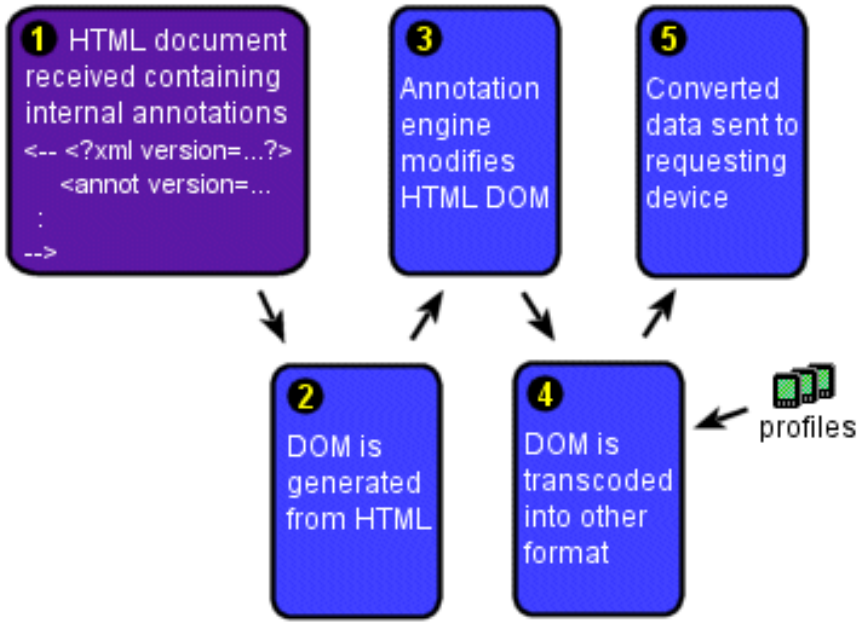
As an alternative to performing document clipping by writing text clippers, Transcoding Publisher also provides an annotation language that enables you to perform HTML document clipping with nearly the same level of flexibility as a text clipper but without having to know how to program in Java. A special transcoder, the annotation engine, is included to handle annotation processing.

Internal and external annotation

Annotations are composed of a special set of XML tags that, when combined with an HTML source file, dictate which parts of the HTML document should be clipped. You can put all of your annotations for a particular document in a separate annotation file called an annotator (referred to as ***external annotation***) or you can embed annotations directly in the HTML file itself as comments (referred to as ***internal annotation***). Regardless of whether you use internal annotation or an annotator, Transcoding Publisher's annotation engine processes the annotations with the HTML file and produces a clipped version of the document. Because the annotation engine runs before the text transcoder, you can use a single annotator to generate output for multiple devices. The following images show the processing flow for the two types of annotations.



Document clipping with an annotator. Transcoding Publisher receives an HTML file (1), and the HTML DOM Generator transcoder generates an HTML DOM from the document (2). Elements in the HTML DOM that are to be manipulated are identified in the annotator (A1) with XPath statements. The annotation engine modifies the HTML DOM according to the annotations (3). Depending on the device profile associated with this request, the text transcoder can then convert the modified HTML DOM to a different data format, such as WML (4). The transcoded data is then sent on to the requesting device (5).



Document clipping with internal annotation. Transcoding Publisher receives an HTML file containing annotations (1), and the HTML DOM Generator transcoder generates an HTML DOM from the document (2). The annotation engine modifies the HTML document according to the annotations (3). Note that unlike with an annotator, elements in the HTML document are not identified by their XPath location but rather by the location of the annotations in the HTML file itself. Depending on the device profile associated with this request, the text transcoder can then convert the modified HTML file to a different data format, such as WML (4). The transcoded data is then sent on to the requesting device (5).

Annotation clipping states

One of the main ideas behind Transcoding Publisher's annotation function is the notion of a ***clipping state***. By using the <keep /> and <remove /> tags in the annotation language, you can specify the clipping state to indicate whether the content being processed should be preserved or removed. For example, if you are clipping a document for display on a small device, you might activate clipping with the first node within the BODY element. Thereafter the clipping state is set to remove all elements until another annotation is encountered that turns the clipping state to keep. At this point, document elements will be preserved until the clipping state is again set to remove. In this way, you can move through the document, alternately setting the clipping state to keep elements or remove elements as required.

Because tables can be complex elements to clip, the annotation language includes some special-purpose elements to make clipping tables easier. The <row> and <column> elements enable you to mark entire rows and column for clipping without requiring potentially complicated XPath expressions. In addition, if a table is converted to a list as part of the transcoding process, you can specify whether the row or column labels will have precedence in the resulting list, with an attribute on the <table> tag. Refer to the sample annotators, TableScoping and ExternalAnnotationTest, for more information about using annotations with tables.

When Transcoding Publisher processes an annotator, the annotations are merged in with the HTML DOM and surrounded by HTML comment tags. Then the annotation engine goes through the updated DOM and applies the annotations, deleting, adding, and changing nodes as required. Annotators specify where an annotation should be inserted by using an XML Path Language (XPath) expression in a <description> element. The XPath expression identifies the node where the annotation is to be applied, and the take-effect attribute of the <description> tag indicates whether the annotation is applied before or after the node.

For example, take the following annotation, which indicates that the annotation should be applied before the first

table in the HTML document:

```
<description take-effect="before"  target="/descendant::TABLE[1]" >
    <remove />
</description>
```

When this annotation is merged with the HTML DOM to which it applies, you get the result:

```
<BODY>
  <H1>
    <#text>Annotation in Action</#text>
  </H1>
  <!--<?xml version="1.0"  encoding="ISO-8859-1" ?>
   <description take-effect="before"  target="/descendant::TABLE[1]" >
     <remove />    </description>-->
  <TABLE>
   .
   .
   .
  </TABLE>
.
.
.
</BODY>
```

Note that the placement of the annotation in the DOM is governed by the take-effect attribute of the <description> element. Had you wanted the annotation to take effect after the first table, you would specify a value of after, which in turn would result in the following:

```
<BODY>
  <H1>
    <#text>Annotation in Action</#text>
  </H1>
  <TABLE>
   .
   .
   .
  </TABLE>
  <!--<?xml version="1.0"  encoding="ISO-8859-1" ?>
   <description take-effect="after"  target="/descendant::TABLE[1]" >
     <remove />    </description>-->
.
.
.
</BODY>
```

Here the annotation has been inserted after the <TABLE> element, including its children. You can think of the after value as indicating that the annotation will be applied after the closing tag of the target element.

Notes on Using XPath Expressions:

- When creating an annotator, base your XPath expressions on the HTML DOM that is used by the annotation engine. If the HTML document you are clipping is ill-formed, it is possible that your XPath expressions might not coincide with the expected elements after a DOM has been generated. To view a representation of the

DOM, use the Request Viewer to request the HTML document and examine the MEG input for the annotation engine in the request processing view. The HTML document representing the DOM is displayed in the Transaction Content window.

- The processing of the XPath expressions in the annotator is based on the original DOM, with all annotations being inserted at one time. So, for example, if you remove a table from the document with one annotation, you do not need to recalculate the XPath expressions that identify later nodes to account for the missing table. For the purpose of specifying XPath expressions, Table 2 in the document will always be identified as Table 2, even if you remove Table 1.
- For detailed information about using XPath expressions, read Document Clipping with Annotation: How to keep the good stuff and throw out the rest at developerWorks.

Making annotations conditional

You can make annotations depend on fields in the HTTP header or on preferences that are in effect for the document being processed. You can add the **condition** attribute to either the **annot** element or the **description** element, which are described in Annotation Language. The annotation with the condition will be executed only if the condition is true. By making annotators or individual annotations conditional, you can reduce the number of annotators that you must manage and avoid adding new markup inappropriate for the target device.

You can set conditions based on the values of HTTP header fields or preferences that are in effect for the document. You can use preferences whose values are defined as strings. For information about preference names, see Preference Names. In addition to preference values defined in profiles, you can use these keywords:

- **device:** the name of the device profile in effect for this request. The value assigned for this keyword must match the filename of the device profile, not the user-agent value.
- **network:** the name of the network profile, if any, in effect for this request. The value assigned for his keyword must match the filename of the network profile.
- **user:** the name of the user profile in effect for this request

To see valid values for device and network, look at the file names under etc/preferences/ in the WTP installation directory. When you use one of these file names, do not include the ".prop" extension. The second example below illustrates the use of a device profile file name.

When you add a condition, place the expression to be evaluated in quotation marks. If you want to combine conditions by "ANDing" them, you must use & between the conditions as in these examples. XML uses & as the beginning of symbols, so if you use & by itself it will be interpreted as the beginning of a symbol rather than as AND. WTP will change the & into AND in order to evaluate the conditions. The condition specified on a description element has precedence over any condition specified on the enclosing annot element.

Here is an example of a condition on an external annotation, illustrating a combination of conditions, designed to match only those requests with a user agent starting with "Mozilla/4." or "Mozilla/5." and not containing "MSIE":

```
<description take-effect="before" target="/HTML[1]/BODY[1]/*[1]" condition="((User_Agent=Mozilla/4.*) |
(User_Agent=Mozilla/5.*)) &amp;amp; !(User_Agent=*MSIE*)" >
<remove />
</description>
```

Here is an example of a condition on an internal annotation, using the device keyword and the name of a device preference profile:

```
<!--
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

```
<annot version="1.0" condition="device=WinCE.PocketIE20">
<remove/>
</annot>
-->
```

Creating annotators

You can create annotators by using the Annotation Editor, which displays the source page and provides graphical tools that you can use to create annotators and view their effects. You can use WebSphere Studio to add internal annotations to your own HTML pages.

Because Transcoding Publisher's annotation language is based on XML, you can also create annotators with the text editor of your choice. If you are using internal annotation, you would put your annotations directly into the HTML file to which they apply; however, if you are creating an annotator, you would put all of your annotations in a separate file. Much of the discussion in this section deals with annotators.

To create an annotator, do the following:

1. Familiarize yourself with the HTML source file that you want to annotate. The annotation language assumes you can identify specific elements in the source file that indicate where clipping should be performed. If the HTML source file is ill-formed, refer to Using XPath Expressions for information on referencing nodes in the document.
2. Using a text editor, construct your annotations according to the syntax defined in the annotation language. Save your file with a file extension of .ann. Transcoding Publisher provides some sample annotators that you can use as a guide.
3. After you have completed your annotator, you must register it with Transcoding Publisher by using the Administration Console. This enables you to associate the annotator with a specific URL that identifies the HTML file on which you based the annotations. In this way, when a client device requests the URL for this page, the annotation transcoder applies the annotations you created to the data before sending the response back to the requesting device.

Internal annotation is largely the same as that used with annotators. However, because there are some minor differences between the two, refer to the InternalAnnotationTest sample for details you need to be aware of if you want to create an internal annotation file.

When you create an annotator, be sure that it does not delete the BODY tag near the beginning of the document. If the BODY tag is missing from an HTML document, Transcoding Publisher will not attempt to transcode the document and will issue a message saying, "Transcoding resulted in an empty document."

Annotation Samples

To help illustrate how annotations can be applied, Transcoding Publisher provides several sample annotators in the toolkit, demonstrating both internal and external annotations. The samples are located in the directory toolkit/annotators/samples. Each annotator is accompanied by an HTML file that you can place on an available Web server. You can register the annotators with Transcoding Publisher and then request the HTML files to see how the files are clipped. The internal annotation examples do not require any registration; as long as the annotation engine is enabled, the HTML file will be clipped when it is requested through Transcoding Publisher.

As you read the samples, refer to Annotation Language for information about each annotation element.

These samples are included in the toolkit:

- TableScoping Sample: illustrates basic table clipping and shows the effect of table clipping on the clipping state of the document
- ExternalAnnotationTest Sample #1: illustrates more aspects of table annotation
- AnnotationTest Version 2: illustrates many new features such as a conditional annotation, setting a preference, inserting new markup, and designating a fragmentation point
- InternalAnnotationTest Sample #1: shows annotations inside an HTML file
- InternalAnnotationTest Sample #2: illustrates new features like those shown in ExternalAnnotationTest Sample #2, but inside an HTML file
- FormTest Sample: shows how to manipulate HTML forms with annotations
- IBMStock Sample: illustrates the use of XPath statements to find the desired content even if the HTML page is changed
- InternalAnnotationSnippets: includes a collection of internal annotations for cut-and-paste
- StudioTest Sample: shows how to use annotations with a tool such as WebSphere Studio

TableScoping Sample

The TableScoping sample is a simple example of basic table manipulation, but it illustrates an important aspect of the annotation language, specifically that the clipping state that is active when the table is encountered is restored when the table processing is complete, *regardless of how you might have changed the clipping state within the table itself*.

For example, the clipping state is explicitly set to keep prior to the first table in the HTML document. Then the following annotation sets the clipping state to remove before the second row (of the first table encountered in the document):

```
<description take-effect="before" target="/descendant::TR[2]">
  <remove />
</description>
```

When the annotation engine processes this instruction and removes the table row, it then completes its processing of the table and restores the clipping state that was active before the table processing began. So, although the annotation above sets the clipping state to remove while within the table, that clipping state is only maintained for the remainder of that table.

ExternalAnnotationTest Sample #1

The ExternalAnnotationTest sample provides more examples of how tables can be clipped. There are two approaches to table clipping that the sample illustrates:

- Clipping where you want to maintain the table structure in the output document but want to discard (or preserve) entire rows and columns
- Clipping where you want to discard the table structure in the output document but want to preserve the contents of some of its cells

At the beginning of the document, the annotator sets the clipping state to remove before the first node after the <BODY> tag:

```
<description take-effect="before" target="/HTML[1]/BODY[1]/*[1]">
   <remove />
```

```
</description>
```

Thereafter all elements in the document are discarded until the first table is encountered. Note that the " *[1]" notation assumes that the first item after the <BODY> tag is another tag, such as <P>. If the HTML document you are clipping has text immediately following the <BODY> tag, you would use an XPath expression like this: target="/HTML[1]/BODY[1]/text()[1]".

There are special instructions for processing the table when it is reached:

```
<description take-effect="before" target="/descendant::TABLE[1]">
  <keep />
  .
  .
  .
  <table majoraxis="row">
     <column index="1" clipping="remove" />
     <column index="*" clipping="keep" />
     <row index="*" clipping="keep" />
     <row index="2" clipping="remove" />
  </table>
</description>
```

The clipping state is set to keep just before the table is entered, and a series of shorthand clipping instructions is specified to process the rows and columns. For example, the first column (indicated by the index value of the <column> tag) is discarded, while the remaining columns are preserved. Note the use of the wildcard character to indicate multiple columns (index="*"). As the <row> elements show, the ordering of <row> or <column> elements does not affect their processing; if a wildcard is specified, all rows (or columns) will be affected, except those specifically denoted in separate <row> (or <column>) element. So for this table, all rows but the second will be preserved.

Whereas the previous annotation produces a table in the HTML output, the sample handles the second table differently. After setting the clipping state to remove after the first table, the next annotation actually targets a specific cell in the second table:

```
<description take-effect="before" target="/descendant::TABLE[2]/TBODY[1]/TR[1]/TH[1]/*[1]">
  <keep />
</description>
```

In this case, the clipping state is not set to keep again until the first table-heading cell of the second table in the document. Another annotation sets the clipping state back to remove immediately after that cell. The result is that only the first table-heading cell of the table is preserved, and the rest of the table (including the table tagging) is discarded.

Note: The <TBODY> element that appears in the XPath expression above is automatically added to the HTML DOM, even though commonly it is not used in many HTML documents. Be sure to include the <TBODY> element in your XPath expressions.

When performing clipping on a table with annotation, it is recommended that you use *either* the shortcut approach (as in the first table) or the standard keep/remove approach (as in the second table). Combining the two notations in the same table annotation can sometimes generate unexpected results.

AnnotationTest Version 2

This sample builds upon the ideas developed in ExternalAnnotationTest Sample #1, such as special table-related annotations. It also illustrates:

- The use of conditions to produce different output for different device types: if the user-agent field in the HTTP header contains the string "WAP" then some additional WML markup will be added to the document. If the condition is true, the markup will be added regardless of the current clipping state.
- The setting of a preference, which will override the preference set by the preference aggregator
- The setting of the language and subject matter to facilitate translation by WebSphere Translation Server
- The removal of the second row and second column from the first table
- The replacing of the second table with a new HTML heading
- The insertion of new HTML into the document
- The designation of an advantageous location for fragmentation

InternalAnnotationTest Sample #1

The InternalAnnotationTest sample demonstrates how to apply annotations within an HTML source file, unlike the external annotation represented by an annotator file. For this reason, the sample consists of only an HTML file and no corresponding Ann file.

This sample performs a similar table clipping function as that in the ExternalAnnotationTest sample. In addition to the fact that <description> elements are not used, a special feature of internal annotation is that you can specify a default clipping state at the beginning of the document with an HTML <META> tag. The following element is required to use internal annotation:

<meta name="Annotation_v1.0" content="remove">

where the content attribute indicates the clipping state (keep or remove). Subsequent annotations are placed at the point in the document where they are to take effect and are enclosed by HTML comment tags. The elements typically used within a <description> element in an annotator (<keep> or <remove>, for example) can be used within the <annot> element in an internal annotation file.

For example, the following annotation sets the clipping state to keep, so that the text that follows will be preserved in the HTML output:

```
<!--
   <?xml version="1.0" encoding="ISO-8859-1"?>
   <annot version="1.0">
      <keep/>
   </annot>
-->
<b>Goodbye</b>
```

XPath expressions indicating a target node are allowed in an internal annotation file, but they are ignored. This is to enable you to migrate an annotator to an internal annotation without requiring you to remove the XPath expressions.

The sample is shown below:

```
<HTML>
<HEAD>
<META name="Annotation_v2.0" content="remove" >
```

```
<title>Internal Ano</title>
</HEAD>
<BODY>
<B>Hello world.</B>

<!--
<?xml version="1.0" encoding="ISO-8859-1" ?>
<annot version="2.0" >
<setpreference key="disposeImages" value="1" />
<contentlanguage>en</contentlanguage>
<wtssubject>automobiles </wtssubject>
<keep/>
<table>
<column clipping="remove" index="2" />
<column clipping="keep" index="*" />
<row clipping="remove" index="2" />
<row clipping="keep" index="*" />
</table>
</annot>
-->
<TABLE>
<TR><TH>Cars</TH><TH>Colors</TH><TH>Interiors</TH></TR>
<TR><TH>2 Door</TH><TH>Red</TH><TH>Leather</TH></TR>
<TR><TH>4 Door</TH><TH>Blue</TH><TH>Cloth</TH></TR>
<TR><TD>Convertible</TD><TD>Black</TD><TD>Vinyl</TD></TR>
</TABLE>


<!--
<?xml version="1.0" encoding="ISO-8859-1" ?>
<annot version="2.0" condition="(User-Agent=*WAP*)" >
<remove />
<insertmarkup><![CDATA[<br/> This conditionally included WML markup is inserted after transcoding occurs
<br/>]]>
</insertmarkup>
</annot>
-->
<BIG> Another Heading </BIG>

<!--
<?xml version="1.0" encoding="ISO-8859-1" ?>
<annot version="2.0" condition="(User-Agent=*WAP*)" >
<keep />
</annot>
-->

<!--
<?xml version="1.0" encoding="ISO-8859-1" ?>
<annot version="2.0" >
<replacewithhtml><![CDATA[ <h1>Second table replaced with this new HTML</h1>]]></replacewithhtml>
</annot>
-->
```

```
<TABLE>
<TR><TH>Motorcycles</TH><TH>Colors</TH><TH>Displacement</TH></TR>
<TR><TH>Touring</TH><TH>Red</TH><TH>250cc</TH></TR>
<TR><TH>Sport</TH><TH>Blue</TH><TH>600cc</TH></TR>
<TR><TD>Cruiser</TD><TD>Black</TD><TD>1000cc</TD></TR>
</TABLE>


<!--
<?xml version="1.0" encoding="ISO-8859-1" ?>
<annot version="2.0" >
<inserthtml><![CDATA[<P> Inserting A New HTML paragraph </P> <br/>]]></inserthtml>
</annot>
-->
<B> Warranties </B>


<!--
<?xml version="1.0" encoding="ISO-8859-1" ?>
<annot version="2.0" >
<splitpoint />
<keep />
<table>
<column clipping="keep" index="*" />
<row index="2" clipping="remove" />
<row index="*" clipping="keep" />
</table>
</annot>
-->
<TABLE>
<TR><TH><B>Warranty</B></TH><TH><B>Length</B></TH><TH><B>Restrictions</B></TH></TR>
<TR><TH>Silver</TH><TH>2 Years</TH><TH>Yes</TH></TR>
<TR><TH>Gold</TH><TH>5 Years</TH><TH>No</TH></TR>
</TABLE>
<!--
<?xml version="1.0" encoding="ISO-8859-1" ?>
<annot version="2.0" >
<remove />
</annot>
-->

</BODY>
</HTML>
```

InternalAnnotationTest Sample #2

This sample builds upon the ideas developed in InternalAnnotationTest Sample #1. This sample provides the same annotations provided by AnnotationTest Version 2, but in this sample the annotations are included in the HTML file, and there is no separate annotator file. This sample illustrates:

- The use of conditions to produce different output for different device types: if the user-agent field in the HTTP header contains the string "WAP" then some additional WML markup will be added to the document.

If the condition is true, the markup will be added regardless of the current clipping state.

- The setting of a preference, which will override the preference set by the preference aggregator
- The setting of the language and subject matter to facilitate translation by WebSphere Translation Server
- The replacing of a table with a new HTML heading. This must occur in a section where the clipping state is keep.
- The insertion of new HTML into the document
- The designation of an advantageous location for fragmentation
- The removal of the second row of the Warranties table

FormTest

The FormTest sample illustrates how you can use annotation to manipulate HTML forms. After specifying the first <FORM> in the document as the target node, the <replace> element is invoked to indicate that the target node is to be replaced with the <replace> element's contents. A <form> element is then defined that reorders the input fields in the form, adds and changes label text, and retrieves the SUBMIT button.

For example, consider the following annotations:

```
<text>NEW prompt for INPUTTEXT3(now a select)</text>
<field name="INPUTTEXT3" type="SELECT">
   <option value="No" label="No." />
   <option value="Yes" label="Yes." />
   <Option value="Maybe" label="Don't Care" />
</field>
```

The <text> element inserts a new text label, while the <field> element modifies the INPUTTEXT3 field to be a select list and defines its options. The name attribute must match the name attribute of the corresponding <input> tag in the HTML document to correctly identify the field for manipulation.

In addition, the sample shows how you can reorder the input fields according to the order of the <field> elements. In this case, INPUTTEXT3 has been placed before INPUTTEXT2, and INPUTTEXT1 has been changed into a hidden field with a default value. It is good practice to position a text label before the field to which it applies to ensure proper transcoding into other formats, particularly HDML.

If you need to make more extensive modifications to a form, use <replacewithhtml> rather than replace.

IBMStock

The IBMStock sample clips the IBM stock price from the company's Web site (www.ibm.com). Because this annotator works with an HTML page available over the Internet, the HTML source is not included. To see how the page is rendered, register this file as an annotator with the Administration Console and then request http://www.ibm.com/ibm/stock through Transcoding Publisher.

The IBMStock example illustrates how XPaths can be used to consistently identify a particular HTML element on a page even when large portions of the page are changed frequently. Consider the following XPath from this example:

```
<description take-effect="before" target="/descendant::text()[contains(.,'Last
trade:')]/ancestor::TABLE[1]">
     <keep />
     <remove tag="IMG"/>
```

```
</description>
```

This XPath says the following: find the String, "Last trade:", and from this string, look for the closest table that encloses this string (this is denoted by the ancestor::TABLE[1] portion of the XPath). We must denote it as the closest enclosing table (i.e., we need to say ancestor::TABLE[1], not just ancestor::TABLE) because in this example the table we want to keep is enclosed in another table and as a result there are multiple tables that fit the description of being a table that encloses the String "Last trade:".

With this XPath in place, we are guaranteed to always identify the table we want, no matter how many new tables (or other HTML elements) are added to the page. All that is required to stay the same is that the table should always have the text string, "Last trade:" inside of it. For more examples of annotation XPath usage, read Document Clipping with Annotation: How to keep the good stuff and throw out the rest at developerWorks.

InternalAnnotationSnippets

The InternalAnnotationSnippets file provides a set of ready-made annotations that you can cut and paste into your HTML documents to create internal annotations. The snippets cover all the main areas of the annotation language and can be adapted as you see fit for your own application.

StudioTest

The StudioTest sample provides an example of how annotations could be used in conjunction with tools for Web applications, such as IBM WebSphere Studio.

Annotation Language

Transcoding Publisher's annotation language is organized around a straightforward notion of document clipping. Using annotations, you can delimit parts of an HTML document and then choose to either keep them or discard them. Whereas the approach with Java-based text clippers is usually to clip out the parts you want to retain, annotations give you a flexible means to do either, as well as to combine them if you wish.

The elements in the language cover several general areas:

- General purpose clipping, wherein you simply turn clipping on or off at various points.
- Annotations that are specific to a particular HTML element. These include annotations to simplify the display of forms on a pervasive device, such as a mobile phone.
- Shortcut annotations that streamline the annotation task itself. For example, this includes the ability to perform fine-grained clipping within a table, which could be tedious and error-prone were it done according to the standard annotation approach.
- Generic DOM-based annotations that enable you to set an attribute value of an HTML element or to add additional comments to the HTML document.
- Inserting HTML or other markup, such as WML, or replacing an HTML node with a new HTML fragment.
- Designating advantageous locations for fragmentation.

Annotation Language Elements

<annot>

Indicates the beginning and end of an annotation. Only one <annot> element can appear in an external annotation. Internal annotations must use an <annot> element to identify each point where clipping is to occur and so can have more than one.

Can contain: <description>, <comment>

Attributes

version=*text*

Indicates the version of the annotation language that is used by the annotation. Current version is 2.0.

condition=*text*

Defines a condition that must be true for the enclosed annotation to be applied. See Making annotations conditional.

Example

```
<annot version="2.0" >
  ...
</annot>
```

<column>

Enables you to select an entire column to be kept or discarded.

Appears within <table> element.

Attributes

index=*number*

Indicates which column the annotation pertains to. You can also use a wildcard character (*) for this attribute to specify that the annotation pertains to all columns in the table, except those specifically denoted in a separate <column> element.

clipping=keep|remove

Indicates whether the column is to be kept or discarded.

Example

```
<description take-effect="before"
      target="/descendant::TABLE[2]" >
  <keep tag="all" />
  <insertattribute name="summary" value="Summary of the table"  />
  <table majoraxis="row" >
    <column index="*" clipping="remove"  />
    <column index="2"  clipping="keep"  />
    <column index="3" clipping="keep"  />
    <row index="*" clipping="remove"  />
    <row index="2"  clipping="keep"  />
    <row index="3" clipping="keep"  />
  </table>
</description>
```

<comment>	Adds a comment regarding the annotation. The information specified in this element is not rendered in the HTML output. Appears within <annot> element.

Example

```
<annot version="1.0" >
  <comment>IBM Stock Annotator: clips IBM stock page</comment>
  ...
</annot>
```

<contentlanguage>, <wtssubject>	Provide information about documents that assists in translating: 1. Language of the document (mandatory) -- While this may be configured to come with the document, it often is not. The machine translation transcoder assumes that an unmarked document is English, but when it is not, the results will not be correct. 2. Subject of the document (optional) -- Machine Translation has much better results if the general subject area is understood. The subject area provides the context for resolving vocabulary. Using annotation, we provide ways for you to associate source language and subject with specific documents and/or sites. Using <contentlanguage> and <wtssubject>, you have the capability to specify this information separately from the source document. Additionally, this allows you to specify the information once for a collection of documents, rather than have to specify it individually for each document. Clipping state has no effect on the use of these elements.

Examples

If you have the Translation Server, you will have a Dictionary Editor that can be used to add vocabulary and to refine pairings between one language and another. You also have the list of 3000+ subjects that are understood by the WebSphere Translation Server in all language pairs. These components result in more accurately translated pages.

Here is a sample for external annotation:

```
<description take-effect="before" target="HTML[1]/BODY[1]/*[1]" >
<contentlanguage>fr (that is, the two letter ISO language identifier) </contentlanguage>
</description>

<description take-effect="before" target="HTML[1]/BODY[1]/*[1]" >
<wtssubject>Term defined in WTS list (otherwise WTS will ignore it) </wtssubject>
<wtssubject>Term</wtssubject>
...
</description>
```

Here is a sample for internal annotation:

```
<!--
<?xml version="1.0" encoding="ISO-8859-1" ?>
<annot version="2.0" >
<contentlanguage>fr </contentlanguage>
```

```
<wtssubject>WTS term</wtssubject>
<wtssubject>WTS term</wtssubject>
....
</annot>
-->
```

<description>

Describes a specific annotation action. This element is the main element used to implement an annotation.

Note: When you use internal annotation, this element is not used, and its children (<keep>, <remove>, etc.) are used with the <annot> element instead.

This element may have a condition attribute. In external annotation where both <description> and <annot> have associated condition attributes, the <description> annotation takes precedence over the <annot> annotation.

Appears within <annot> element.

Can contain: <comment>, <contentlanguage>, <insertattribute>, <insertcomment>, <inserthtml>, <insertmarkup>, <keep>, <remove>, <replace>, <replacewithhtml>, <setpreference>, <splitpoint>, <table>, <wtssubject>

Attributes

condition=*text*

Defines a condition that must be true for the enclosed annotation to be applied. See Making annotations conditional. A condition specified on a description element overrides any condition on an enclosing annot element.

take-effect=before|after

Indicates whether the annotation occurs before or after the node identified by the target attribute. If the annotation is to occur after the referenced node, the children of the node will also be ignored. The annotation will occur after the closing tag of the referenced node.

target=*XPath expression*

Indicates where the annotation will occur. The annotation language uses the XML Path Language (XPath), version 1.0, to specify at which node the annotation should be inserted. Transcoding Publisher operates on an HTML DOM form of the document and applies the XPath expressions to modify the DOM. If the HTML source file is ill-formed, refer to Using XPath Expressions for information on referencing nodes in the document.

Example

```
<description take-effect="before"
  target="/descendant::HTML[1]/BODY[1]/*[1]" >
  ...
</description>
```

Using this construct, the annotation takes effect before the first child of the HTML <BODY> tag.

<field>

Enables you to explicitly reorder, hide, or preset basic input fields in an HTML form, such as <input>, <select>, <textarea>, or <buttons>. In addition, you can change how some fields are represented; for example, you can change <textarea> and <input> fields into a <select> menu. If your form uses fields other than <input>, <select>, <textarea>, or <buttons>, or requires other modifications than those provided by <field>, use <replacewithhtml> instead. The <replacewithhtml> element replaces a form with HTML in any way you want.

For each field in the original HTML document that you want to keep, you must specify a <field> annotation element whose name attribute matches the name attribute on the corresponding HTML <input> tag. If an HTML field does not have a name attribute, the field is deleted.

The order of the <field> elements in the annotation file indicates the order in which the fields will be displayed in the output document.

Appears within <form> element.

Can contain: <option>

Attributes

name=*text*

Uniquely identifies each field in the form. This field is required and must match the name attribute of the corresponding <input> tag in the HTML document.

value=*text*

Sets the default value for the field.

type=hidden|select|submit|reset

Specifies the kind of field that should be used in the output document. By setting this attribute you can change the field from one format in the original HTML to another format that might be more suitable for a pervasive device. If you specify select for this value, you can set options for the HTML <select> menu with the <option> element.

Example

```
<replace>
  <form>
    <text>Submit selection</text>
    <field name="submit" value="retrieve" />
    <text>Choose your country</text>
    <field name="textfield" type="select" >
      <option value="japan"  label="Japan"  />
      <option value="usa"  label="USA"  />
      <option value="other"  label="Other country"  />
    </field>
    <field name="hiddenvalue"  value="secret" type="hidden"  />
  </form>
</replace>
```

<form>

Enables you to replace a form in the original HTML document with customized elements better suited to display on a pervasive device. The annotated form creation process produces
's separating the components. This results in a better presentation of forms.

Appears within <replace> element.

Can contain: <text>, <field>

Example

```
<replace>
  <form>
    <text>Submit selection</text>
    <field name="submit" value="retrieve" />
  </form>
</replace>
```

<insertattribute>

Enables you to add a new attribute to, or modify an existing attribute of, any element specified in the related <description> tag's target attribute.

Appears within <description> element.

Attributes

name=*text*

Indicates the name of the attribute to be added to the HTML tag.

Value=*text*

Indicates the value of the name attribute added to the HTML tag.

Example

```
<description take-effect="before"
      target="/descendant::TABLE[2]" >
  <keep tag="all" />
  <insertattribute name="summary" value="Summary of the table"  />
  <table majoraxis="row" >
    <column index="1" clipping="remove" />
  </table>
</description>
```

<insertcomment>

Used to add a comment to the HTML output. Currently this element is only used for internal processing by Transcoding Publisher and does not produce output on the requesting device.

Appears within <description> element.

Example

```
<description take-effect="before"
      target="/descendant::IMG[1]" >
  <keep />
  <insertcomment>Replace image with text</insertcomment>
  <replace>
    <text>IBM Stock Quote</text>
```

```
  </replace>
</description>
```

<inserthtml>	Inserts an HTML fragment into a page, using HTML contained within a CDATA section. Once the HTML content is included, it is parsed as an HTML fragment and then spliced into the HTML DOM. This insertion is done before transcoding of the DOM takes place. Even if you place this HTML fragment in a <remove> section, <inserthtml> overrides the clipped state and the inserted HTML is retained.

Examples

Here is an external annotation example of this feature. Note how we use a CDATA section in the XML to hide this HTML content from the XML parser.

```
<description take-effect="before" target="/descendant::IMG[2]" >
<inserthtml><![CDATA[ <p> Hello World]]></inserthtml>
</description>
```

Here is an internal annotation example of this feature.

```
<!--
<?xml version="1.0" encoding="ISO-8859-1" ?>
<annot version="2.0" >
<inserthtml><![CDATA[ <p> Hello World]]></inserthtml>
</annot>
-->
```

<insertmarkup>	Inserts rendered markup into a page. This is accomplished by the use of a CDATA section inside the element.

Examples

<insertmarkup> inserts rendered markup into a page. The rendered markup is inserted only after transcoding has been performed. This element is not affected by the current clipping state.

We recommend using insertmarkup conditionally, to avoid inserting markup of one variety in a page intended for a device expecting another variety of markup. Be careful not to break conventions of the markup language that the device uses, such as not breaking the WML by inserting more than one <p> in a card. Do not use insertmarkup when the requesting browser is Pocket Internet Explorer, a handweb browser on a Palm device, or a desktop browser such as Netscape or Internet Explorer. Use inserthtml for these devices.

Here is an external annotation example of this feature. Note how we use a CDATA section in the XML to hide this rendered markup from the XML parser and a condition on the description tag to prevent this annotation's use with a desktop browser.

```
<description take-effect="before" target="/descendant::IMG[2]"
condition="!(User_Agent=Mozilla/4.*) &amp;amp; !(User_Agent=Mozilla/5.*) &amp;amp; !(User_Agent=*MSIE*)" >
<insertmarkup><![CDATA[ <a href="wtai://wp/mc; +4917112345678 ! $ret" >]]></insertmarkup>
</description>
```

Here is an internal annotation example of this feature.

```
<!--
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<annot version="2.0" >
<insertmarkup><![CDATA[ <a href="wtai://wp/mc; +4917112345678 ! $ret" >]]></insertmarkup>
</annot>
-->
```

<keep>

Indicates that the node identified by the annotation should be retained.

Appears within <description> element.

Attributes

tag=all|*tag name*

Indicates which node or nodes should be retained. A value of all causes every HTML tag from the starting node (identified in the <description> tag) to be retained in the output. You can specify individual HTML tags as well, and then each occurrence of that element will be retained.

Example

<keep tag="all" /> ; Can also be abbreviated as **<keep />**
<keep tag="IMG" />

You can also combine <keep> and <remove> to set a clipping state and then specify exceptions:

<keep />
<remove tag="IMG" />

Note that the exception to the clipping state (removing images, in this example) is only maintained until the next <keep> or <remove> instruction.

<option>

Enables you to specify options that will appear in the <select> menu of the output document.

Appears within <field> element.

Attributes

value=*text*

Indicates the value used for this option in the HTML <select> menu.

label=*text*

Indicates the text used as a label for this option in the HTML <select> menu.

Example

```
<replace>
  <form>
    <text>Submit selection</text>
    <field name="submit" value="retrieve" />
    <text>Choose your country</text>
    <field name="textfield" type="select" >
      <option value="Japan" label="Japan" />
      <option value="USA" label="USA" />
      <option value="other" label="Other country" />
    </field>
```

```
  </form>
</replace>
```

<remove>

Indicates that the node identified by the annotation should be removed.

Appears within <description> element.

Attributes

tag=all|*tag name*

Indicates which node or nodes should be removed. A value of all causes every HTML tag from the starting node (identified in the <description> tag) to be removed from the output. You can specify individual HTML tags as well, and then each occurrence of that element will be removed.

Example

<remove tag="all" /> ; Can also be abbreviated as **<remove />**
<remove tag="IMG" />

You can also combine <remove> and <keep> to set a clipping state and then specify exceptions:

<remove />
<keep tag="IMG" />

Note that the exception to the clipping state (keeping images, in this example) is only maintained until the next <keep> or <remove> instruction.

<replace>

Indicates that the node identified by the annotation should be replaced with the contents of this element. The replaced content must be in a <keep> section to be retained.

Appears within <description> element.

Can contain: <text>, <form>

Example

```
<replace>
  <form>
    <text>Submit selection</text>
    <field name="submit" value="retrieve" />
  </form>
</replace>
```

This element only allows nodes to be replaced with a text string, and only allows a form to be modified in limited ways. For more flexibility and control, use <replacewithhtml> instead. The <replacewithhtml> element replaces a node with HTML instead of a text string, and replaces a form with HTML in any way you want.

<replacewithhtml>

Replaces an existing HTML node with a new HTML fragment in a page, using HTML contained within a CDATA section. Once the HTML content is included, it needs to be parsed as an HTML fragment by the Dharma parser, and then spliced into the HTML DOM. The replaced HTML is done before transcoding. The replaced fragment must be in a <keep> section to be retained.

You should consider using <replacewithhtml> instead of <replace>, because <replacewithhtml> provides a wider range of function.

Examples

Here is an external annotation example of this feature. Note how we use a CDATA section in the XML to hide this HTML content from the XML parser.

```
<description take-effect="before" target="/descendant::IMG[2]" >
<replacewithhtml><![CDATA[ <p> Hello World]]></replacewithhtml>
</description>
```

Here is an internal annotation example of this feature.

```
<!--
<?xml version="1.0" encoding="ISO-8859-1" ?>
<annot version="2.0" >
<replacewithhtml><![CDATA[ <p> Hello World]]></replacewithhtml>
</annot>
-->
```

<row>

Enables you to select an entire row to be kept or discarded.

Appears within <table> element.

Attributes

index=*number*

Indicates which row the annotation pertains to. You can also use a wildcard character (*) for this attribute to specify that the annotation pertains to all rows in the table, except those specifically denoted in a separate <row> element.

clipping=keep|remove

Indicates whether the row is to be kept or discarded.

Example

```
<description take-effect="before"
      target="/descendant::TABLE[2]" >
  <keep tag="all" />
  <insertattribute name="summary" value="Summary of the table"  />
  <table majoraxis="row" >
    <column index="*" clipping="remove"  />
    <column index="2"  clipping="keep"  />
    <column index="3" clipping="keep"  />
    <row index="*" clipping="remove"  />
    <row index="2"  clipping="keep"  />
    <row index="3" clipping="keep"  />
  </table>
```

</description>

<setpreference>	A method of setting preference values to do change the transcoding that will be done on a document

Examples

You can use this element to change a preference to get a result that you would have had to write a transcoder to achieve in the past. For instance, to remove all images on the page simply set the preference key to remove images. The clipping state of the page has no effect on the setpreference key and its values.

You can insert several setpreference elements into a page, setting the same preference to different values. In this case, only the last value you give will take effect, and it will take effect for the whole page.

You can use this element to modify String preferences. For information about preference names, see Preference Names.

An example of setting the preference key is:

<setpreference key="disposeImages" value="1"/>

<splitpoint>	Informs the fragmentation engine that it has reached a desirable point to fragment the content and start a new card. This complements the current model of solely counting bytes for determining when to fragment. <splitpoint> does not need to be in a <keep> section to function.

Examples

When this element is in the annotation, the document is fragmented at each such indicated point, if it can be. If you place a <splitpoint/> element at a point where a document cannot be fragmented, such as inside a table, it will be ignored.

The fragmentation takes place before or after a given element, based on the value of the take-effect attribute in the description element. For example, if fragmenting is desired at the beginning of a given FORM element, then the annotation element should be pointing at that FORM and the take-effect attribute should have a value of " before" .

When fragmenting, if the fragmentation support encounters a document which contains fragmentation directives or is too large, it will split at the first fragmentation directive that it encounters or at a split point based on maximum size, whichever comes first. It will then continue to fragment in this way as long as it contains additional fragmentation directives or is too large.

For example, the following annotation would place a fragmentation directive before the first table in the document:

```
<description take-effect="before"  target="/descendant::TABLE[1]" >
<splitpoint/>
</description>
```

<table>

Enables you to identify tables that should be clipped using row and column shortcuts. Also enables to specify whether column headings are propagated when the HTML table is converted to a list.

Appears within <description> element.

Can contain: <row>, <column>

Attributes

majoraxis=row

Optional attribute that indicates that column headings contained in the first row of the table have precedence if the HTML table is converted to a list as the result of a transcoding process. Note that the annotation engine itself will not perform this kind of conversion; if it receives a table, it will output a table. However, if the output from the annotation engine is transcoded into another markup language (such as WML), it is possible that tables could be converted to lists. In this case, this attribute will be used to help ensure the readability of the resulting list when the labels occur in the first row of the table.

For example, consider the following table:

Car	Color	Interior
2-Door	Red	Leather
4-Door	Blue	Cloth

If majoraxis is set, the resulting list would be ordered as follows: Car, 2-Door, Color, Red, Interior, Leather, Car, 4-Door, Color, Blue, Interior, Cloth.

Example

```
<description take-effect="before"
      target="/descendant::TABLE[2]" >
  <keep tag="all" />
  <insertattribute name="summary" value="Summary of the table"  />
  <table majoraxis="row" >
    <column index="*" clipping="remove"  />
    <column index="2"  clipping="keep"  />
    <column index="3" clipping="keep"  />
    <row index="*" clipping="remove"  />
    <row index="2"  clipping="keep"  />
    <row index="3" clipping="keep"  />
  </table>
</description>
```

<text>	Either replaces an element identified by the <description> tag's target value with a text element or inserts a text element into a form. Appears within <replace> and <form> elements.

Examples

```
<replace>
  <text>Replace the target node with this text.</text>
</replace>

<replace>
  <form>
    <text>Submit selection</text>
    <field name="submit" value="retrieve" />
  </form>
</replace>
```

<wtssubject>	Provides information about the subject of the document to assist in translating. Machine Translation has much better results if the general subject area is understood. The subject area provides the context for resolving vocabulary. This element is described together with contentlanguage.

Annotation Document Type Definitions

This section provides the Document Type Definitions (DTDs) that define the elements and attributes for external and internal annotation.

External Annotation (Annotator) DTD

The following Document Type Definition (DTD) defines the external annotation format used with annotators.

```
<!DOCTYPE annot [

<!ELEMENT annot (description|comment)*>
<!ATTLIST annot
  version CDATA #REQUIRED
  condition CDATA #IMPLIED>

<!ELEMENT description (keep|remove|replace|table|insertcomment|insertattribute|comment
                      |setpreference|contentlanguage|wtssubject|splitpoint|inserthtml
                      |insertmarkup|replacewithhtml)* >
<!ATTLIST description
  take-effect (before|after) "before"
  target CDATA #REQUIRED
  condition CDATA #IMPLIED>

<!ELEMENT keep EMPTY>
<!ATTLIST keep tag CDATA "all">

<!ELEMENT remove EMPTY>
```

```
<!ATTLIST remove tag CDATA "all">

<!ELEMENT table (column|row)* >
<!ATTLIST table majoraxis CDATA #IMPLIED>

<!ELEMENT column EMPTY>
<!ATTLIST column
  index NMTOKEN #REQUIRED
  clipping (keep|remove) #REQUIRED>

<!ELEMENT row EMPTY>
<!ATTLIST row
  index NMTOKEN #REQUIRED
  clipping (keep|remove) #REQUIRED>

<!ELEMENT replace (text|form) >

<!ELEMENT text (#PCDATA) >

<!ELEMENT form (text|field)* >

<!ELEMENT field (option*) >
<!ATTLIST field
  name CDATA #REQUIRED
  value CDATA #IMPLIED
  type (hidden|select|submit|reset) #IMPLIED>

<!ELEMENT option EMPTY>
<!ATTLIST option
  value CDATA #REQUIRED
  label CDATA #REQUIRED>

<!ELEMENT insertattribute EMPTY>
<!ATTLIST insertattribute
  name CDATA #REQUIRED
  value CDATA #REQUIRED>

<!ELEMENT comment (#PCDATA)>
<!ELEMENT insertcomment (#PCDATA)>
<!ELEMENT setpreference EMPTY>
<!ATTLIST setpreference
  key CDATA #REQUIRED
  value CDATA #REQUIRED>

<!ELEMENT contentlanguage (#PCDATA)>
<!ELEMENT wtssubject (#PCDATA)>
<!ELEMENT splitpoint EMPTY>

<!ELEMENT inserthtml (#PCDATA)> <!-- This element´s contents should be enclosed in a CDATA section -->
<!ELEMENT insertmarkup (#PCDATA)> <!-- This element´s contents should be enclosed in a CDATA section -->
```

```
<!ELEMENT replacewithhtml (#PCDATA)> <!-- This element´s contents should be enclosed in a CDATA section
-->
]>
```

Internal Annotation DTD

The following DTD defines the internal annotation format:

```
<!DOCTYPE annot [
<!ELEMENT annot (keep|remove|replace|table|comment|setpreference|contentlanguage|wtssubject|splitpoint
    |inserthtml|insertmarkup|replacewithhtml|insertcomment|insertattribute)* >
<!ATTLIST annot
  version CDATA #REQUIRED
  condition CDATA #IMPLIED>

<!ELEMENT keep EMPTY>
<!ATTLIST keep tag CDATA "all">

<!ELEMENT remove EMPTY>
<!ATTLIST remove tag CDATA "all">

<!ELEMENT table (column|row)* >
<!ATTLIST table majoraxis CDATA #IMPLIED>

<!ELEMENT column EMPTY>
<!ATTLIST column
  index NMTOKEN #REQUIRED
  clipping (keep|remove) #REQUIRED>

<!ELEMENT row EMPTY>
<!ATTLIST row
  index NMTOKEN #REQUIRED
  clipping (keep|remove) #REQUIRED>

<!ELEMENT replace (text|form) >

<!ELEMENT text (#PCDATA) >

<!ELEMENT form (text|field)* >

<!ELEMENT field (option*) >
<!ATTLIST field
  name CDATA #REQUIRED
  value CDATA #IMPLIED
  type (hidden|select|submit|reset) #IMPLIED>

<!ELEMENT option EMPTY>
<!ATTLIST option
  value CDATA #REQUIRED
  label CDATA #REQUIRED>
```

```
<!ELEMENT insertattribute EMPTY>
<!ATTLIST insertattribute
  name CDATA #REQUIRED
  value CDATA #REQUIRED>

<!ELEMENT comment (#PCDATA)>
<!ELEMENT insertcomment (#PCDATA)>
<!ELEMENT setpreference EMPTY>
<!ATTLIST setpreference
  key CDATA #REQUIRED
  value CDATA #REQUIRED>

<!ELEMENT contentlanguage (#PCDATA)>
<!ELEMENT wtssubject (#PCDATA)>
<!ELEMENT splitpoint EMPTY>

<!ELEMENT inserthtml (#PCDATA)> <!-- This element´s contents should be enclosed in a CDATA section -->
<!ELEMENT insertmarkup (#PCDATA)> <!-- This element´s contents should be enclosed in a CDATA section -->
<!ELEMENT replacewithhtml (#PCDATA)> <!-- This element´s contents should be enclosed in a CDATA section
-->
]>
```

Using Stylesheets

Transcoding Publisher can perform transformations on XML data using Extensible Stylesheet Language (XSL) stylesheets. Examples of formats that can be produced by applying stylesheets include: HTML for display on a desktop browser, Wireless Markup Language (WML) for display on a smart phone, and VoiceXML for auditory rendering. XSL stylesheets can also be used to convert between different business-to-business XML formats to enable information exchange between companies.

This section describes some considerations to keep in mind when writing stylesheets for use with Transcoding Publisher, particularly if you are exploiting the parameterization or localization support. The stylesheet samples included with the product are also described.

Xerces-Java and Xalan-Java

As a basis for its XML transcoding function, Transcoding Publisher incorporates two primary technologies: the Xerces-Java XML parser and Xalan-Java stylesheet processor. Xerces-Java performs a number of tasks when used in stylesheet conversion, including parsing, generating, manipulating, and validating XML documents. Xalan-Java formats and transforms XML pages based on XSL stylesheets. Both Xerces-Java and Xalan-Java are open-source software packages available from the Apache Software Foundation.

Transcoding Publisher uses Xerces-Java to produce Document Object Model (DOM) input for Xalan-Java and supports XML 1.0, as defined by the World Wide Web Consortium (W3C). The version of Xalan-Java included with Transcoding Publisher supports XSL Transformations (XSLT) Version 1.0 (W3C Recommendation 16 November 1999) and XML Path Language (XPath) Version 1.0 (W3C Recommendation 16 November 1999). See the W3C site for information about the latest version of XSL.

JavaScript Extensions to Stylesheet Functionality

Xalan permits you to execute complex JavaScript extensions from within a stylesheet. JavaScript extensions are described in the Xalan documentation. In order to get JavaScript extensions to work, you must do two things to WebSphere Transcoding Publisher.

- Get the bsfengines.jar from http://oss.software.ibm.com/developerworks/projects/bsf and add it to the classpath specified in RunTranscoding.bat and/or RunTranscoding.sh.
- You also need to update the classpath specified in cmagicsrv.dat in the line beginning **addclasspath** if WebSphere Transcoding Publisher runs as a WebSphere Application Server service,

If you want to use the JavaScript extensions via the TransformTool, update the classpath in TransformTool.bat.

After updating the classpath, you need to get an updated copy of js.jar and replace the one in the WebSphere Transcoding Publisher lib directory. The newer js.jar can be found at http://www.mozilla.org/rhino/ in some of the packages available at the download page.

Specifying stylesheets within a document

The *Administrator's Guide* describes how WTP selects a stylesheet to be applied to an XML document. As an alternative to selecting a registered stylesheet to be applied to an XML document, WTP can select a stylesheet that is specified within the XML document. Stylesheets specified in this way do not have to be registered with WTP. The stylesheet specification includes the conditions under which that stylesheet is to be applied. Several stylesheet specifications can be included in the same XML document.

The preferred method for specifying stylesheets within an XML document is to use the wtp-condition processing instruction. You can add several of these instructions to the prologue of an XML document to specify different stylesheets for different conditions. The wtp-condition instruction can use the values of any fields in the HTTP header and any preferences in effect. It can also use the three special variables **device**, **network**, and **user**, which contain the respective profile names determined during preference aggregation. (For consistency, these three variables can also be used when specifying conditions for the selection of a registered stylesheet in the Administration Console.)

If you want to combine conditions by "ANDing" them, you must use & between the conditions as in these examples. XML uses & as the beginning of symbols, so if you use & by itself it will be interpreted as the beginning of a symbol rather than as AND.

WTP will change the & into AND in order to evaluate the conditions.

```
<?xml version="1.0"?>
<?wtp-condition  stylesheet="file://mywml.xsl"
condition="(targetContentType=text/vnd.wap.wml)
&amp; (textLinksPreferredToImages=1) &amp; (url=*orders/recent*)
&amp; (device=WMLDevice)"?>

<?wtp-condition  stylesheet="file://myhdml.xsl"
condition="(textLinksPreferredToImages=1)
&amp; (url=*orders/recent*) &amp; (device=HDML-Device)"?>

<?wtp-condition  stylesheet="http://www.ibm.com/mypalm.xsl"
condition="(textLinksPreferredToImages=1)
&amp; (url=*orders/recent*) &amp; (device=Palm-Pilot3.HandWeb11)"?>
```

For another example, if user Juan wants a different stylesheet applied when he requests a certain page from his voice-enabled browser, his administrator could add this instruction to the document:

```
<?wtp-condition  stylesheet="file://Juan-voice.xsl" condition="(user=Juan)
& (device=VoiceXML-Device) & (url=*orders/recent*)"?>
```

This example assumes that another program has passed the name of the user profile to WTP.

The second method for specifying a stylesheet within a document is based on the function provided by the Cocoon application, which maps stylesheets to documents based on the target device. This is accomplished by coding the media element on an xml-stylesheet processing instruction. For example:

```
<?xml-stylesheet href="hello-text.xsl" type="text/xsl" media="lynx"?>
```

This method does not enable you to use the HTTP header fields and preference values. To use this method, you must import your cocoon.properties file into WTP using the command:

```
ImportCocoon -File <path>/cocoon.properties
```

By default, WTP uses its usual process of selecting a registered stylesheet to be applied to an XML document, and looks for stylesheet specifications within the document only if no appropriate registered stylesheet is found. You can change this default by modifying the etc/plugins/ibm/TextEngine/XMLHandler.prop file. Change the value of **checkForDocumentStylesheetsFirst** to true if you want WTP to look for stylesheet specifications within the file before looking for an appropriate registered stylesheet.

If you are using server models to store your servers' configuration, you can modify this value in either of two ways:

- Use the SecureWay Directory Management Tool (DMT) to modify the entry within each server model in the central directory. The entry is **settingID=checkForDocumentStylesheetsFirst, cid=XMLHandler,cid=TextEngine,cid=ibm,cid=plugins,cid=*server_model_name*, sys=wtp40,cn=wtpReleases,sys=SDP,*your_directory_suffix***.
- Use XML configuration commands to export the properties of XMLHandler so that you can modify the property and re-import the properties. Follow these steps:
  1. Edit the first server model you want to modify.
  2. Export the XMLHandler properties:

     ```
     ExportResources -Node "plugins/ibm/TextEngine/XMLHandler" -File "XMLHandler.xml"
     ```
  3. Use your favorite editor to change the value of **checkForDocumentStylesheetsFirst**.
  4. Import the file:

     ```
     ImportResources -Node "*"  -File "XMLHandler.xml"
     ```
  5. For each other server model that you want to change, edit the server model and run the import command again.

If you tell WTP to look first for stylesheet specifications within documents, WTP will examine each XML document that it processes to look for wtp-condition or xml-stylesheet processing instructions. WTP will apply the first stylesheet found whose conditions are satisfied. If a stylesheet is selected but not found, WTP will write a message to the trace log. If a stylesheet has not been selected when WTP encounters the end of the document prologue, WTP will use its usual preference-based process to select a registered stylesheet to be applied to the document.

Looking in each XML document for stylesheet specifications will affect the performance of your WTP server.

Creating Your Own Stylesheets

WTP 4.0 includes the new XSL Stylesheet Editor, which enables you to create stylesheets and view their impact immediately.

Because Xalan-Java has specific requirements regarding the supported version of XSL specification, be sure to specify the correct transform type in your stylesheet.

For stylesheet transformations using Transcoding Publisher, identify the stylesheet version in the stylesheet file as follows:

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

Information about how stylesheets are functioning is written to the Transcoding Publisher tracing files. Errors from the stylesheet processor are displayed in the trace log, along with the name of the stylesheet. Exceptions are included with the Low level of trace. The Medium level of trace also indicates which stylesheet is selected for processing.

For information on registering stylesheets, associating stylesheets with preference profiles, and Transcoding Publisher's tracing and logging capability, refer to the *Administrator's Guide*.

A stylesheet can include other stylesheets. If the location of the included stylesheet is specified as a relative path, WTP uses a default URI as the base of that relative path. Prior to Version 4.0, WTP used the etc directory as the default URI for included stylesheets. The default URI is changed in WTP 4.0 to be the URL of the main stylesheet.

Managing Your Stylesheets

While XSL stylesheets provide an accessible and adaptable means of transforming XML documents, they can also provide a file management challenge when a number of stylesheets are used to perform similar transformations. For example, you might choose to have a stylesheet for each type of output you want to generate from the original XML document. If you expand this example to cover many XML documents and if each XML document is associated in turn with many stylesheets for conversion to various formats, you soon have a flood of stylesheets to deal with.

There are several ways to deal with this potential proliferation of stylesheets. Through the Administration Console, you can organize your stylesheets into folders according to any scheme that helps you manage them. Two other ways of limiting the burden of managing your stylesheets are using parameters with stylesheets and using a common intermediate format.

Using Parameters with Stylesheets

One way of dealing with the complexity of stylesheet management is by using parameters with stylesheets. Parameters can be defined outside the stylesheet and then passed to the stylesheet for processing. In the example of proliferating stylesheets above, you might choose to use a single stylesheet for all output types, while using a parameter in the stylesheet to select the type of output to be generated. Or, suppose you want to use a parameter to determine which language should be used to render the output. This use of parameters is considered in more detail in Localizing Stylesheets. Some of the samples listed in Stylesheet Samples demonstrate the use of parameters with stylesheets.

Setting Parameter Values

For each parameter in a stylesheet, Transcoding Publisher attempts to set the value of the parameter for processing by going through the following steps in order:

- Determine whether the stylesheet selector for the stylesheet defines a value for the parameter
- Examine the HTTP request header and response header to determine whether the parameter value is included there
- Query the preference aggregator to determine whether the parameter value is defined as part of a preference profile

If you choose to set a parameter value for use in stylesheet processing, the choice of where to set the value will depend on how the parameter is to be used. For example, define the parameter in the stylesheet selector if you want to limit the parameter's availability to a specific stylesheet or if you want the parameter value available each time the stylesheet is processed. However, if you want the parameter available to any stylesheet that is processed in conjunction with that device, define the parameter in the device profile.

These parameter values, with the exception of those supplied by the HTTP request header, can be set from within the Administration Console. For stylesheet selectors, the parameter values are specified as key/value pairs from the Stylesheet Parameters window. For device profiles, the parameters are treated as device preferences (also key/value pairs) specified from the Advanced Preference Profile Properties window.

Transcoding Publisher Parameters

Transcoding Publisher provides a set of default parameters that can be used with any stylesheet.

Parameter	Description
deviceType	This parameter indicates the kind of device that the stylesheet output is intended for. This parameter contains the value of the deviceType preference in the preference profile for a device, not the name of the profile. This parameter can be used with an <xsl:choose> instruction. For example, a value of I-Mode 501 Device could cause one type of output to be generated, while a value of WML Device could cause another.
markupLanguage	This parameter is set according to the first value for the desiredContentTypes preference in the device profile associated with the stylesheet processing. However, if an explicit markupLanguage preference has been added to the profile, that value will be used instead.
multipleSelectionSupported	This parameter is set according to the multipleSelectionSupported preference in the device profile associated with the stylesheet processing. This parameter indicates whether the markup language used with the device supports lists where you can select more than one element at a time. The default value is true.

The sample described under Using a common intermediate format also illustrates the use of parameters.

Using a common intermediate format

If you need to present XML data on numerous device types, you might need a separate stylesheet for each document and each output type. This requires expertise in all the supported output formats as well as management of large numbers of stylesheets. One solution to the problem of needing many stylesheets to support many output devices is to use a single stylesheet to convert an XML document to HTML, and let WTP transcode the document from HTML to the appropriate format.

If you want to use HTML as a common intermediate format, you will need just one stylesheet, to convert your XML document to HTML. Consider specifying this stylesheet within the document, as described in Specifying stylesheets within a document. If you specify the stylesheet within the document, you do not need to register the stylesheet to have WTP select it for processing.

To enable the use of a common intermediate format, you must modify the Desired Content Types field for the target device. Be sure that the "true" content type for the device is listed first, and add text/html as a second content type. This will enable stylesheets creating HTML to be selected to process documents requested by this device.

WTP includes a set of files in the toolkit\stylesheets\samples\hotels directory, which you can use as an example of:

- Using a stylesheet to convert XML to a common intermediate format
- Using parameters within a stylesheet
- Translating a phrase based on the **lang** variable

To use the sample, register the hotels.xsl stylesheet through the Administration Console and make hotels.xml available on a web server.

The sample demonstrates three different uses of parameters, each to accomplish a different result. The common intermediate format

depends on the **screenCapability** parameter, which is automatically provided by WTP. Possible values of the parameter are high, intermediate, and low. You can tailor content based on these three values. In the sample, this parameter is used to choose between a table display for high- and intermediate-capability screens and a list for low-capability screens.

The second use of parameters is in altering the display based on which hotels the user wants to see. If the **country** parameter is passed on the URL, the stylesheet will display hotels from the selected country, if any. For example:

```
http:\\webserver\hotels.xml?country=France
```

Another use of parameters relies on the **lang** parameter that is also provided by WTP. The sample uses the built-in capabilities of WTP to translate one phrase to another based on the input language, which might be set by a browser.

For more information about using HTML as an intermediate format, see Spinning your XML for screens of all sizes on developerWorks.

Localizing Stylesheets

In addition to its primary function as a transformation mechanism, a stylesheet can also be used to generate localized content, so that a different language can be used to accommodate an international audience. The extent to which you exploit stylesheets to present localized content can be as elaborate as providing translated versions of all your content or as simple as providing translations for commonly accessed elements, such as form labels.

Transcoding Publisher includes several files that illustrate how you might implement localized stylesheets, either with all languages in a single file or with each language having its own file. For example, the translate.xsl file, located in the <install_path>/etc/stylesheets directory, illustrates how you might access translated content, where each language is in a separate XML file.

References to this stylesheet template would be given as:

```
<xsl:call-template name="translate">
  <xsl:with-param name="word" select="'enter'"/>
</xsl:call-template>
```

In this case the stylesheet is being asked to return a translated value for a <word> element with an ID of enter. The stylesheet determines the value of the language parameter (Lang) and then builds the filename of the XML file containing the content, which for this case would be translations_en.xml. The translations_en.xml file contains the following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<translations>
  <word id="address">address</word>
  <word id="button">button</word>
  <word id="check">check</word>
  <word id="enter">enter</word>
  <word id="find">find</word>
  <word id="language">language</word>
  <word id="location">location</word>
  <word id="name">name</word>
  <word id="press">press</word>
  <word id="search">search</word>
  <word id="submit">submit</word>
  <word id="telephone">telephone</word>
</translations>
```

The stylesheet outputs the string "enter," because the language returned is English. Had the language been set for French, however, the stylesheet would consult a file called translations_fr.xml and output "entrer." Note that although translate.xsl defines a default value for the Lang parameter, this parameter could also be set through other means, as described in Setting Parameter Values.

Transcoding Publisher includes a number of these translation XML files in the <install_path>etc/stylesheets directory, along with the translate.xsl stylesheet. The translate_sf.xsl stylesheets and translations_sf.xml file are also included as an example of how to perform similar language selection from a single file containing the translated strings for all the languages.

Although you are free to add your own <word> elements to the XML translation files that come with Transcoding Publisher, keep in mind that future releases of the product might add elements to the files. You should be prepared to migrate any additions you make to the newer versions of the files. Of course you can also choose to create your own separate XML files that follow a similar scheme and would be used in a similar fashion.

Transcoding Publisher Extension Elements

An extension element is an instruction that is not defined by the XSLT standard but that looks and behaves like an XSLT instruction. Extension elements enable you to augment the function available with XSLT.

Transcoding Publisher includes two element extensions designed to simplify the XML coding required to access a translated string from a stylesheet: <nls:trans> and <nls:trans_sf>. While these extensions perform a similar function, they differ in that <nls:trans> is used when the translated strings for each language are in a separate file, while <nls:trans_SF> is used when all translated strings for all languages are in a single file.

Where without these extensions, you might use:

```
<xsl:call-template name="translate">
  <xsl:with-param name="word" select="'enter'"/>
</xsl:call-template>
```

You can use instead:

```
<nls:trans id="'enter'"/>
```

However, to use these extensions, you must add some additional information to the stylesheet declaration (see highlighted lines in example below):

```
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:lxslt="http://xml.apache.org/xslt"
  xmlns:nls="com.ibm.transform.textengine.mutator.stylesheet.ext.NLS"
  extension-element-prefixes="nls"
  version="1.0"/>

<xsl:include href="stylesheets/translate.xsl"/>

  <xsl:template match="/">
    .
    .
    .
    <nls:trans id="'button'"/>
    .
    .
  </xsl:template>
</xsl:stylesheet>
```

Note that the inclusion of translate.xsl with the <xsl:include> instruction provides support for the use of <nls:trans>. If you include translate_sf.xsl, you would specify <nls:trans_sf> instead, if you wanted to make use of the extension. In both translate.xsl and translate_sf.xsl, the parameters for the translation file and language are given default values. However, if you wanted to override either of these values in the stylesheet above, you would specify the new value prior to the <xsl:include> instruction with something like <xsl:param name="Lang" select="'sp'"/>.

Setting the lang parameter

The lang parameter is used to specify the language of the transformed output. When Transcoding Publisher performs stylesheet processing, it attempts to set the value of lang in the following sequence:

- The preference aggregator is queried to determine whether lang might have been set directly as part of a preference profile.

- The Accept-Language header in the HTTP request header is examined for a value. If more than one language has been specified in the header, only the first one is used to set the lang parameter.
- The defaultLanguage preference is checked. Typically used with device profiles, the defaultLanguage preference provides a way to give the Accept-Language header precedence in the selection process.
- Finally, if the lang parameter is still not specified, the default settings in the stylesheet files themselves are used.

Stylesheet Samples

Transcoding Publisher includes several sample stylesheets that demonstrate how XML data can be transcoded for different types of devices. The following files are located in the <install_path>/toolkit/stylesheets/samples directory:

- **FlightInfo.xml:** An XML file that describes airline flight information, such as the carrier, flight number, departure and arrival times, and so forth.
- **FlightInfoForNet_IE.xsl:** A stylesheet that defines how the XML data is transformed for an HTML browser.
- **FlightInfoForPalm.xsl:** A stylesheet that defines how the XML data is transformed for a Palm Pilot device.
- **FlightInfoForWinCE.xsl:** A stylesheet that defines how the XML data is transformed for a device running Windows CE.
- **FlightInfoForWML.xsl:** A stylesheet that defines how the XML data is transformed for a WML device, such as a smart phone
- **TestParamWithTranslation.xsl:** A stylesheet that tests for a parameter and uses a translation file. Apply it to TestParam.xml.
- **hotels/hotels.xsl:** A set of stylesheets that show the use of parameters and a common intermediate format. See Using a common intermediate format for information about this set of samples.

Viewing the FlightInfo Samples

To see the results when you transcode FlightInfo.xml with any of the supplied stylesheets, you will need to request it through Transcoding Publisher. If you are running Transcoding Publisher as a network proxy, you can put the XML file on an accessible Web server and then request it through the proxy as you normally would.

Alternately, you can configure Transcoding Publisher to deliver the file through a servlet. To do this, you can use the TranscodingSamples servlet, which can run either as a MEGlet in Transcoding Publisher or a servlet in WebSphere Application Server. The source code for this servlet, TranscodingSamples.java, is located in the toolkit/misc directory. Because TranscodingSamples delivers files from the server's local file system, it is not installed by default in either environment for security reasons. Moreover, after it is installed and enabled, TranscodingSamples will only deliver files that are listed in the toolkit/RequestableSamples.prop file. The version of this file included with Transcoding Publisher contains a listing for FlightInfo.xml. You can add additional lines for your own files as you need them.

Using Transcoding Publisher as a Proxy

To use TranscodingSamples in a proxy environment, do the following:

1. Register the TranscodingSamples transcoder with the Transcoding Publisher Administration Console. The JAR file for this step is located in toolkit/misc/TranscodingSamples.jar. Refer to the online help in the console for specific instructions on registering transcoders.
2. With your browser configured to use Transcoding Publisher, retrieve the following URL: http://[hostname]/servlet/TranscodingSamples?name=FlightInfo.xml ([hostname] is actually ignored by the proxy, but some value should be included to ensure that correct URL syntax is observed).

Using WebSphere Application Server

To use TranscodingSamples as a filter in a WebSphere Application Server enabled for transcoding, do the following:

1. Make TranscodingSamples accessible to the Web application that you are working with by doing one of the following:
    - Copy the TranscodingSamples.class file to the Web application's servlets directory.
    - Add the directory containing the TranscodingSamples.class file (for example, /opt/IBMTrans/toolkit/misc) to the Web application's classpath.

2. Define the servlet to the Web application using the WebSphere Administrative Console and specify the following initialization parameters:
    - InstallPath: Transcoding Publisher's installation directory (for example, /opt/IBMTrans).
    - LogRequests: Specify true to generate a log of requested files. The default value is false.
3. Restart the Web application if you changed its classpath in step 1, or start the Web application if it is not already running.
4. Retrieve the following URL: http://[hostname]/[web-path]/TranscodingSamples?name=FlightInfo.xml (where [hostname] is the machine on which WebSphere Application Server is installed and [web-path] indicates the Web path of the Web application).

Where to find more stylesheet information

This section includes notes about various techniques you can use to get the most out of stylesheet processing. Each note will introduce a technique and link to a site with detailed information.

If you are just getting started with stylesheet creation, visit an introductory tutorial, which includes a glossary of key terms. Then follow these tips to improve your XSLT coding five ways.

You can use one stylesheet to create others out of stylesheet building blocks. Learn about a component-based approach to building stylesheets. For more information about using stylesheet components, and generating different tags for different situations, and dynamically loading content into a stylesheet, see The Mod Squad: Reusable stylesheet components that enable a single stylesheet for content tailoring.

For more tips and examples of working with stylesheets, look at the WTP 3.5 Redbook. This book describes Version 3.5 of WTP, so the version numbers for Xalan and Xerxes are wrong for WTP 4.0, but the concepts are valid and useful.

JavaBean Transcoders

Some of the transcoding capabilities of Transcoding Publisher have also been packaged as separate JavaBean components that can be used with other applications, such as Java servlets or JavaServer Pages. The JavaBean transcoders are a way of segregating the basic transcoding functions of Transcoding Publisher and making them available in a scaled-down version that, while lacking some of the more complex interactions of the transcoding server, has the advantage of less processing overhead and easier implementation.

Transcoding Publisher includes these JavaBean transcoders:

- The **HttpPreferenceAggregatorBean** performs the function of the WTP preference aggregator, resolving preferences as described in the the *Administrator's Guide*. The Transform Tool uses the HttpPreferenceAggregatorBean transcoder to resolve preferences; use it as an example of how to use this transcoder.
- The **ImageTranscoderBean** converts images to different formats and performs other image transformations, such as scaling and color reduction.
- The **HtmlReducerBean** simplifies HTML documents in various ways, including replacing images with text links and stripping out objects or features that might not be supported by the target device.
- The **ImodeBean** converts HTML data into the Compact HTML used with the i-mode service. To use this JavaBean transcoder, the i-mode transcoder must be enabled.
- The **HdmlBean** converts HTML data into Handheld Device Markup Language (HDML). To use this JavaBean transcoder, the HDML transcoder must be enabled.
- The **VoiceXMLBean** converts HTML data into Voice eXtensible Markup Language (VoiceXML). To use this JavaBean transcoder, the VoiceXML transcoder must be enabled.
- The **PalmBean** converts HTML data into data suitable for Palm devices. To use this JavaBean transcoder, the Palm transcoder must be enabled.
- The **WmlBean** converts HTML data into Wireless Markup Language (WML).
- The **XmlHandlerBean** transforms XML data from one dialect of XML to another, based on the application of XML stylesheets.
- The **HtmlDomBean** creates a DOM from an HTML document.

JavaBean Samples

To demonstrate how the JavaBean transcoders can be used, this toolkit includes two samples:

- Transform Tool
- ImageTranscoder JavaServer Page

Transform Tool Sample

The first sample, TransformTool.java, is also the source code for one of the developer tools supplied with Transcoding Publisher. The Transform Tool is a quick way to preview how information will be transcoded, with original and transcoded content displayed side by side. To perform its conversions, the Transform Tool uses the JavaBean transcoders . TransformTool.java is located in <*install_path*>/toolkit/beansamples/TransformTool.

ImageTranscoder Sample

The second sample is an example of using the JavaBean transcoders with JavaServer Pages (JSP) technology. The ImageTranscoder sample demonstrates how to incorporate the ImageTranscoderBean in a JSP page that renders an image and then allows you to transcode it based on several different output devices. A sample JSP page is included for both JSP 1.0 and later (ImageTranscoder.jsp) and JSP 0.91 (ImageTranscoder91.jsp). These files are located in <*install_path*>/toolkit/beansamples/jsp, along with an image file used in both JSP pages.

When viewed in a browser, the sample page shows an image for transcoding. After you select the target device, click the **Transcode** button to transform the image. To see the transcoded version of the image, be sure to reload the page in the browser.

JSP Version Notes

There are a few differences in syntax between JSP 1.0 and JSP 0.91 that you will need to consider, depending on which version you are using.

When declaring the usage of an instance of a JavaBean component:

JSP 1.0:	Use the jsp:useBean tag: <jsp:usebean id='itb' class='com.ibm.transform.bean.ImageTranscoderBean'/>
JSP 0.91:	Use the Bean tag: <bean name="itb" type="com.ibm.transform.bean.ImageTranscoderBean" .....>

When specifying JSP directives:

JSP 1.0:	The page directive now collects several attributes. <%@ page language = "java" %> <%@ page import = "java.io.*, com.ibm.transform.bean.ImageTransformBean"%>
JSP 0.91:	<%@ language = "Java" %> <%@ import = "java.io.*, com.ibm.transform.bean.ImageTransformBean"%>

To declare class-wide variables for the servlet class:

JSP 1.0:

```
<%!
    // code for class-wide variables and methods
%>
```

JSP 0.91:

```
<script runat=server>
    // code for class-wide variables and methods
</script >
```

Using the JavaBean Transcoders

This section describes several considerations to keep in mind when using the JavaBean transcoders provided by Transcoding Publisher, including:

- Defining the proper classpath and path information
- Specifying the input parameters required by the JavaBean transcoders
- Special considerations for Linux users

API documentation for the JavaBean transcoders is also available.

JavaBean transcoders do not have a mechanism to detect dynamic updates from Transcoding Publisher. If you register a new resource while Transcoding Publisher is already running, reinstantiate the JavaBean transcoder to be sure that it has the current preference setting information from Transcoding Publisher before attempting to perform any transcoding. The Transform Tool provided with the toolkit is an example of a program that implements several of the JavaBean transcoders. This example taken from the Transform Tool shows how to reinstantiate JavaBean transcoders:

```
try {

hdmlBean = new HdmlBean();
hdmlBean.setInstallPath(installPath); hdmlBean.setSystemDatabaseDirectory(configDir);
hdmlBean.initialize();

}
catch (Exception e) {

hdmlAdded = false;
System.err.println("HdmlBean bean NOT available");

}
```

The complete source code (TransformTool.java) is available in the \toolkit\beansamples\TransformTool directory.

Defining the Classpath and Path

Before you can use the JavaBean transcoders or compile the related samples, you need to ensure that the following information is added to your classpath (All of the items listed here are relative to the Transcoding Publisher installation directory):

- .\beans\magicbeans.jar
- .\lib\xerces.jar
- .\lib\xalan.jar
- .\lib\log.jar (replaces .\lib\ras.jar)
- .\lib\xpath.jar
- .\lib\js.jar
- .\lib\HTMLParse.zip
- .\lib\NetRexxC.zip
- .\lib\NetRexxR.zip
- .\lib\ibmjndi.jar (must be included *before* jndi.jar)
- .\lib\jndi.jar
- .\lib\bsf.jar
- .\lib\wtpcommon.jar
- .\lib\jt400.jar
- .\lib\wtpserver.jar
- .\lib\WebSphereResources.jar

This list has changed since WTP 3.5, so you must update applications that you created using the JavaBean transcoders from a previous WTP release.

For an example of how the classpath should be updated, you can look at TransformTool.bat (located in the Transcoding Publisher installation directory), which is used to launch the Transform Tool.

Note: To use either the ImodeBean or the HdmlBean, you must also add the .\addedPlugins and .\addedPluginJars directories to the beginning of your classpath.

In addition to modifying your classpath, you must also update your path to include the Transcoding Publisher bin directory. Using TransformTool.bat as an example, you could specify:

```
set path=.\bin;%path%;
```

Setting the Classpath in WebSphere Application Server

If you are using WebSphere Application Server 3.*x* (3.02 or 3.5, for example), this classpath information must be added to the application server classpath (with a command-line argument) and *not* to the reloadable classpath specified in an individual Web application. Modifying the application server classpath is necessary for both performance reasons and for ensuring that all resources required to run the JavaBean transcoders are found correctly.

Specifying Input Parameters

When running in the larger context of Transcoding Publisher, transcoders have access to mechanisms that can determine which transformations need to be performed and whether more than one transcoder should be involved in a given transaction. However, the JavaBean transcoders do not have similar

support. Due in part to their lightweight nature, the JavaBean transcoders require that information about the transcoding environment be explicitly provided.

The following types of information must be specified:

- Property file location
- HTTP header information, such as the user-agent value

Property Files

Each JavaBean transcoder relies on the property files supplied with Transcoding Publisher to provide preference settings that indicate how a transformation should be performed for a particular device. Information about which XML stylesheets are registered is also included in the property files.

To define the location of the property files to the JavaBean transcoder, you can use the setInstallPath() and setSystemDatabaseDirectory() methods, as in the following example taken from TransformTool.java:

```
imageTranscoderBean = new ImageTranscoderBean();
imageTranscoderBean.setInstallPath(".");
imageTranscoderBean.setSystemDatabaseDirectory("etc");
imageTranscoderBean.initialize();
```

Note: The Transform Tool runs out of the Transcoding Publisher installation directory, so in this case, the shorthand approach of identifying the installation directory (".") is sufficient. If your application resides in another path, be sure to specify the complete path for Transcoding Publisher (for example, setInstallPath("C:\\Program Files\\IBMTrans\\")). Because the property files are always installed in the same path relative to the installation directory, the use of setSystemDatabaseDirectory("etc") should always be appropriate, as long as the Transcoding Publisher directory is correctly specified.

HTTP Header Information

In addition to knowing where the property files reside, the JavaBean transcoder also needs to know certain aspects of the HTTP header information, including the user-agent value associated with the target device and the MIME type of the content that is being passed to the JavaBean transcoder for transcoding. This information is stored in an HttpPreferenceBundle object, in a series of key-value pairs. To facilitate creating and populating an HttpPreferenceBundle object, Transcoding Publisher provides a special method called generatePreferenceBundle().

Note: The ImageTranscoderBean does not provide a generatePreferenceBundle() method. If you are using ImageTranscoderBean, you must manually build an HttpPreferenceBundle object:

```
HttpPreferenceBundle image_pb = new HttpPreferenceBundle();
image_pb.setContentType("image/gif");
image_pb.setPreference("supportedImages", "[gif, jpg]");
```

Information used in the transformations, such as the user-agent value, is defined in the device and network preference profiles. As part of your code, you also need to indicate which profiles are to be associated with the JavaBean transcoder. This is accomplished with the setDeviceSource() and

setNetworkType() methods.

Although these convenience methods take care of specifying most of the information used in the transformation, you will still need to set the MIME type of the content that the JavaBean transcoder is going to process. You do this with the setContentType() method.

The following example shows how you might set this information to use WmlBean:

```
HttpPreferenceBundle html_pb = WmlBean.generatePreferenceBundle();
html_pb.setDeviceSource("WML-Device");
html_pb.setNetworkType("wireless");
html_pb.setContentType("text/html");
```

After setting these values, you can invoke the transcoding operation with the service() method, passing in the HttpPreferenceBundle object you created along with the content:

```
wmlBean = new WmlBean();
transcodePage = wmlBean.service(html_pb,
                new String(transcodeContentByteArray, "8859-1"));
```

where transcodeContentByteArray contains the original data to be transcoded and 8859-1 indicates the character encoding used.

Special transcoders

Some of the transcoders provided with WTP require special configuration so that they can interact with other servers such as WebSphere Voice Server or WebSphere Translation Server. This chapter describes the configuration you must perform in order to use the

- Voice Transcoder
- Machine Translation Transcoder
- Palm Transcoder

Using the VoiceXML Transcoder

The VoiceXML Transcoder is a plugin for WebSphere Transcoding Publisher (WTP) that transcodes Web content to the format needed for VoiceXML capable browsers. This transcoder transcodes HTML input into VoiceXML that supports the VoiceXML 1.0 Standard. This output can be converted to speech by a voice browser such as the one provided with IBM WebSphere Voice Server.

The VoiceXML transcoder provides the ability to transcode existing HTML content into a limited VoiceXML version. Since VoiceXML is an audio only based medium, and HTML is both a visual and audio based medium, there will exist some HTML Web pages that the transcoder cannot adapt to VoiceXML. However, for many HTML pages, we are able to effectively adapt to VoiceXML by providing supplemental navigational components for traversing the transcoded Web page. Navigational elements may also be disabled and the transcoder can be used in conjunction with annotation to provide a transcoded HTML document as a subpage of an existing VoiceXML application.

For the best experience with this transcoder, it is recommended that you read the style guidelines that are available for download at the Transcoding Publisher Library page.

Profile Parameters

You can use VoiceXML-Device.prop unchanged as your device profile or customize it to describe the capabilities of your voice browser. This section describes how the parameters affect transcoding. #version = 1.0 #Mon Mar 26 11:13:29 EST 2001 desiredContentTypes=[text/x-vxml] propagateFirstTableRowData=true disposeImages=true textLinksPreferredToImages=true deviceType=VoiceXML\ Device deviceRule=(User_Agent%e*Voice*) convertTablesToUnorderedLists=false ConfigurableProperties=disposeImages{bool}\ convertTablesToUnorderedLists{bool}\ fixedImageScale{bool}\ imageScaleFactor{itext}\ desiredContentTypes{text}

- *deviceRule* set to handle the device profile
- *propagateFirstTableRowData* controls the table mutator
- *disposeImages* makes sure images are removed
- *textLinksPreferredToImages* changes links that are images to text, if possible
- *deviceType* device that is expected
- *convertTablesToUnorderedLists* changes tables to unordered lists if true.

Using the Machine Translation Transcoder

The machine translation transcoder enables Transcoding Publisher to work with WebSphere Translation Server (WTS). With these two products, you can provide Web pages that can be translated into multiple languages in real time and then transcoded in various ways to match the capabilities of different devices.

For example, a German-speaking user with a mobile phone enabled for Wireless Application Protocol (WAP) can request an HTML document written in English. The machine translation transcoder, in conjunction with WTS, first translates the content of the Web page from English into German. Transcoding Publisher then converts the translated document from

HTML into Wireless Markup Language (WML) for display on the WAP-enabled phone. Web pages that might not otherwise be accessible to users that speak different languages can now be available in a variety of formats, all without the expense of professional translation or the maintenance of device-specific source files.

When you install WTP, the Machine Translation transcoder is not registered. Before you can use it, you must register it with the Administration Console. After you register the transcoder, you need to give Transcoding Publisher access to the WTS interfaces. To do this, copy the wts.jar file from the directory where you installed WTS to the lib subdirectory of the Transcoding Publisher installation directory.

Read these sections before using this transcoder:

- Setting the language headers
- Setting the translation subject
- Updating settings with XML configuration
- Advanced Topics

Setting the Language Headers

Before any document can be translated with WTS, there are two parameters that must be determined:

- The original language of the document to be translated (Content-Language header)
- The language for the document to be translated into (Accept-Language header)

WTS supports the following language combinations for translation:

Source Language	Target Language
English	German
English	Spanish
English	French
English	Italian
English	Japanese
English	Korean
English	Simplified Chinese
English	Traditional Chinese
German	English
Spanish	English
French	English
Italian	English

For these languages, the following language strings are used:

- English: en
- German: de
- Spanish: es
- French: fr
- Italian: it
- Japanese: jp
- Korean: ko
- Simplified Chinese: zh-cn
- Traditional Chinese: zh-tw

Content-Language Header

To specify the original language of a document, a Web server can set the *Content-Language* header of the document in the headers for the HTTP session. However, because many Web servers do not set the Content-Language header themselves, Transcoding Publisher allows the value to be set in several different ways.

1) Through the HTTP header sent by the Web server
Transcoding Publisher keeps track of the content language information with a key/value pair called contentLanguage. By using one of the methods below to set the Content-Language header, the contentLanguage key is set within Transcoding Publisher.

2) In a device or network profile
The Content-Language header can also be set in specific device and network profiles. In this way, you can set the language for every request coming from a specific type of device. To do this, you will need to specify an additional key/value pair in the profile to serve as a correlator. Using the Administration Console, open the Advanced Preference Profile Properties for the profile and add a key/value pair to specify the appropriate language, such as: Content-Language = en

3) Through the machine translation transcoder's default settings
Transcoding Publisher administrators also have the option of setting a default value for the Content-Language header in the machine translation transcoder's settings.

Accept-Language Header

Most Web browsers allow the user to specify the languages in which they want content to be displayed. This information is translated into an *Accept-Language* header value by the browser and is then sent to Web servers as an HTTP header.

Because many pervasive devices do not provide a mechanism for setting these values, Transcoding Publisher enables you to set the Accept-Language header in several ways.

1) Through the HTTP header sent by the browser
As described above, the user's browser can specify which language or languages that the client prefers to receive.

2) In a device or network profile
The Accept-Language header can also be set in specific device and network profiles. In this way, you can set the language for every request coming from a specific type of device. To do this, you will need to specify an additional key/value pair in the profile to serve as a correlator. Using the Administration Console, open the Advanced Preference Profile Properties for the profile and add a key/value pair to specify the appropriate language, such as: Accept-Language = es

3) Through the machine translation transcoder's default settings
Transcoding Publisher administrators also have the option of setting a default value for the Accept-Language header in the machine translation transcoder's settings.

Setting the Translation Subject

The WebSphere Translation Server allows users to set a subject value for each document that it translates. The subject value is used to determine the meanings of words that have multiple definitions depending on how they are used. For details on how WTS uses subjects to aid in translation, refer to the WebSphere Translation Server documentation.

Transcoding Publisher enables you to specify the subject value in the following ways:

1) Through the machine translation transcoder's default settings
Transcoding Publisher administrators also have the option of setting default subject values in the machine translation transcoder's settings. When you set subjects values in the default settings, those values are used for all documents, whereas setting the values with a request editor (as described above) gives you the flexibility to set the subject values on a document-by-document basis.

Setting the machine translation transcoder's default settings can be done with Transcoding Publisher's XML configuration facility, described below.

Note: Use care when setting subject values. While providing subject information can improve the quality of the translation,

providing superfluous subjects will only degrade the translation server's performance.

Updating Settings with XML Configuration

After registering the machine translation transcoder with Transcoding Publisher, you can use the XML configuration facility to update the transcoder's configuration.

The following settings can be configured (shown with their default values):

servers=server1:1098, server2:1099
DefaultContentLanguage=en
DefaultAcceptLanguage=en
DefaultSubject=

These values can be encoded in an XML file as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <plugins>
    <ibm>
      <MachineTranslationTranscoder>
        <servers>server1.yourcompany.com:1098,server2.yourcompany.com:1099</servers>
        <DefaultContentLanguage>en</DefaultContentLanguage>
        <DefaultAcceptLanguage>en</DefaultAcceptLanguage>
        <DefaultSubject></DefaultSubject>
      </MachineTranslationTranscoder>
    </ibm>
  </plugins>
</root>
```

The values control the behavior of the transcoder:

- **servers:** comma-delimited list of translation servers with the format **`hostname:port.`**
- **DefaultContentLanguage:** abbrevation of the language that the transcoder will assume as the source language for any document whose source-language cannot be determined. This language should be supported on at least one of the translation servers specified in the <servers> element.
- **DefaultAcceptLanguage:** abbreviation of the language that the transcoder will assume all clients prefer to receive if no other Accept-Language header information is provided.
- **DefaultSubject:** comma-delimited list of keywords that the translation servers understand as subjects to be used in providing more accurate translations (see the WebSphere Translation Server documentation for more information).

To apply changes to the machine translation transcoder's configuration, create an XML file similar to the one above and use the XML configuration facility to update Transcoding Publisher. You can create a template XML file by exporting the machine translation transcoder's configuration with the **ExportResources -node /plugins/ibm/MachineTranslationTranscoder** command. This command will export the resources into a file named WTPresources.xml. Make desired changes to the configuration, such as adding or removing servers or changing the language settings, and save the file. To import the changed configuration use the **ImportConfig -node /plugins/ibm/MachineTranslationTranscoder** command.

After updating Transcoding Publisher, you must restart the Transcoding Publisher server.

Advanced Topics

Using Multiple WebSphere Translation Servers

To improve the performance of the machine translation transcoder, it is recommended that you configure the transcoder to use multiple translation servers.

To use multiple servers, create an XML configuration file similar to the one above. Use the file to list the IP addresses of all of your WebSphere Translation Servers. These values must be valid TCP/IP hostnames or IP addresses. Specifying a port is optional. Note that the Remote Method Invocation (RMI) process through which Transcoding Publisher and WebSphere Translation Server communicate uses port 1099 as a default.

So, for example

<servers>192.168.1.100:1099,192.168.1.101:1099,
192.168.1.103:1099,192.168.1.104:1099</servers>

is equivalent to

<servers>192.168.1.100,192.168.1.101,192.168.1.103,
192.168.1.104</servers>

but

<servers>192.168.1.100:1098,192.168.1.101:1098,
192.168.1.103:1098,192.168.1.104:1098</servers>

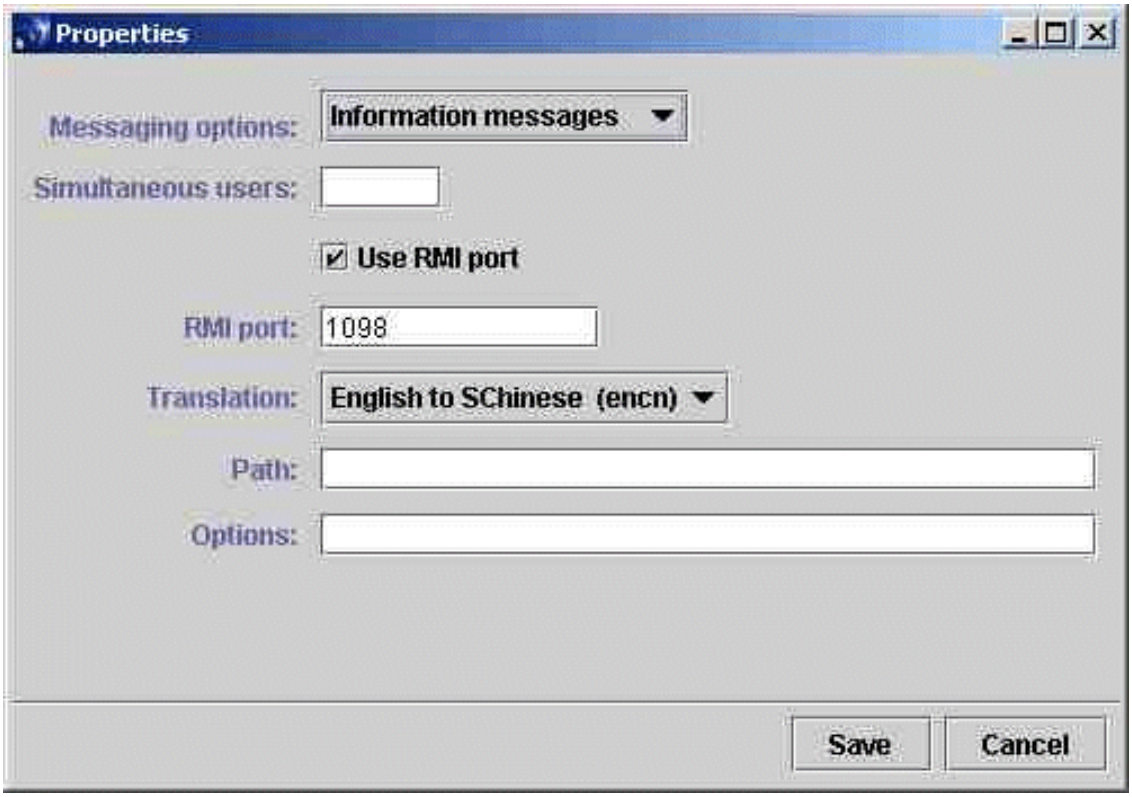
is not equivalent.

You do not need to specify what languages are installed on each server; the transcoder will poll each server specified when it starts. The transcoder will only use the languages that the servers can accept and will not use a server if the server does not respond when the transcoder is starting. If a translation server goes down, the server will not be used again until the transcoder is restarted.

Troubleshooting the RMI communication link

The WebSphere Translation Server and Transcoding Publisher both use RMI and start an RMI registry to communicate between different Java Virtual Machines. If both Transcoding Publisher and a translation server are started on the same machine and attempt to use the same RMI port, there will be a conflict, and only one of them will be able to start an RMI registry.

To avoid this conflict, configure WebSphere Translation Server to use a port other than 1099. You can do this through the WebSphere Translation Server's interface before you start the server, as shown below.



Changing the Priority of the Machine Translation Transcoder

Various Transcoding Publisher transcoders require different input to do their jobs. Some transcoders (such as the annotation transcoders and most of the HTML transcoders) require the generation of a Document Object Model (DOM) representation of the document being transcoded.

By default, the machine translation transcoder is assigned a priority to ensure that it runs after the annotation transcoder (if the annotation transcoder is enabled). This is done to prevent WebSphere Translation Server from translating portions of a document that are subject to being removed by a later annotation step.

However, this method has tradeoffs in terms of performance. If the HTML DOM generator is enabled, the DOM is generated for the annotation step, serialized for translation, and then recreated so that the next transcoder can use it. If annotation is only removing a small amount of text from most Web pages that are being transcoded, a performance gain can be achieved by raising the priority of the machine translation transcoder so that it runs before the HTML DOM generator.

The priority should be set higher than the priority of the HTML DOM generator. This can be determined by running the Request Viewer and viewing the settings of the External HTML DOM Generator. If, for example, the priority of the External HTML DOM Generator is set to 92, you could use the XML configuration file below to set the priority of the machine translation transcoder to 95.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <plugins>
    <ibm>
      <MachineTranslationTranscoder>
        <priority>95</priority>
      </MachineTranslationTranscoder>
    </ibm>
  </plugins>
</root>
```

See Adding Transcoders for more information about priority values for transcoders and how they are used.

Using the Palm Transcoder

The Palm Transcoder for WebSphere Transcoding Publisher (WTP) enables you to use existing enterprise Web sites for mobile use, displaying Web sites on a PalmOS device without requiring the user to install a Web browser. Instead, you will use the Palm Web Clipping Application format to request transcoded Web sites.

To use the Palm Transcoder, a Web Clipping Application (PQA) must be loaded on the device. We envision that customers will develop their own Web Clipping Applications. **wtpdemo.htm** is intended as a sample for how to design a PQA for the PalmOS.

1. Copy the HTML sample below (under Sample PQA Page) to a file called **wtpdemo.htm**.
2. Edit the **BASE HREF** line in **wtpdemo.htm** to point to the WebSphere Transcoding Publisher server.
3. Using the Palm PQABuilder application from Palm.com, build **wtpdemo.PQA** from **wtpdemo.htm**.
4. Optionally, if you intend to enter a URL on the PalmOS device, as opposed to including a specific URL in the PQA file, download the **PalmRedirector.jar.download** file to the WTP installation root folder.
5. Rename **PalmRedirector.jar.download** to **PalmRedirector.jar**.
6. If you want documents to be fragmented, modify the appropriate device profile to set wantFragmenting=true and set HTMLMaximumPageSize to the correct size.
7. Register **PalmRedirector.jar** with the Administration Console by clicking **Register->Transcoder**.
8. Edit the **BASE HREF** line in **wtpdemo.htm** to point to the WebSphere Transcoding Publisher server where the **PalmInternetRedirector** plugin is installed.

The Palm.Net transcoder allows a Palm user to view Web sites without installing a Web browser on the Palm device. Instead, Web sites are transcoded using a WTP plugin and Palm.Net services to the Web Clipping Application format that can be viewed directly using the built-in support in PalmOS 3.5 and later. Although this solution can be used for general purpose Web "surfing", the expected use is to make it easier to enable existing enterprise Web sites for mobile access. A customer who chooses WTP for mobile device support can then support PalmVII devices and other devices connected to the Internet with the Palm Mobile Internet Kit (MIK) with no additional work.

Palm.Net requires that Web pages be designed to certain criteria specified in the Palm Web Clipping Developers Guide, available from Palm.com. Pages must use a defined subset of HTML with a few extra tags and attributes. This transcoder allows any HTML pages to be used, and ensures that the Palm guidelines are followed.

Scenario

This section describes the parts of the solution and how they are connected.

Palm device

The device must be a Palm VII with PalmOS 3.5 or higher, or a supported compatible device with the Palm Mobile Internet Kit installed, and Palm.Net services must be activated.

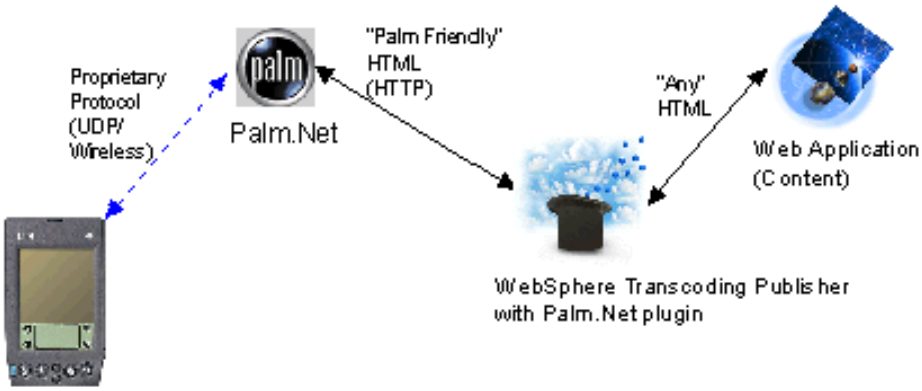
If you don't have a Palm VII or other PalmOS device with the Palm Mobile Internet Kit installed, you may use a PalmOS emulator. To use an emulator with Palm.Net, the proxy setting in the Wireless Preferences page is set to point to Palm.Net. To set the proxy address, open the prefs application and choose wireless from the drop-down. See the Palm Developer Knowledge Base for details. The proxy is currently 206.112.114.81. Check for proxy server address changes at the Palm Web Clipping Development site.

You will need a PQA application installed on the PalmOS device. You should develop this application to meet your specific needs. A sample is included, with guidelines on how to customize it. At a minimum, this application needs to be configured with the address of the WTP host that does the transcoding. It is used to access the starting page for Web browsing. The starting page can be indicated in several ways, but must be coded in the PQA. Several starting pages can be included, but

each must resolve to a server where WTP is installed.

Alternatively (for testing) a Palm Emulator could be used. This requires a 3.5 ROM, and must be installed on a TCP/IP network so it can communicate with the Palm.Net service. See Network Configuration for more details.

Network Configuration



The Palm Device must be configured as described above. The proxy address is the address of the Palm.Net service, not of the WTP transcoding server.

There is no way to specify a proxy server for Palm.Net, so WTP must be configured as a reverse proxy or WebSphere Application Server (WAS) plugin (not a proxy). Of course, applications that access WTP function using the Java Bean interface can also be used.

The communications between the Palm device and Palm.Net do not use standard HTTP, but rather a highly optimized protocol that uses UDP port 5002. If there are any firewalls between the Palm (or Palm Emulator) and Palm.Net, they must be configured to allow this protocol to go through.

In addition, the content host must be accessible from the Internet, so that Palm.Net can forward the requests there. This connection uses standard HTTP protocols.

Sample PQA Page

A PQA is an application that is built using HTML files in accordance with the rules in the Palm Web Clipping Applications document. The HTML and image files are then compiled into an application (with file extension .pqa) using the Palm PQA Builder tool. A PQA application can be installed on a Palm or Palm emulator using the Palm Install Tool provided with the Palm Desktop.

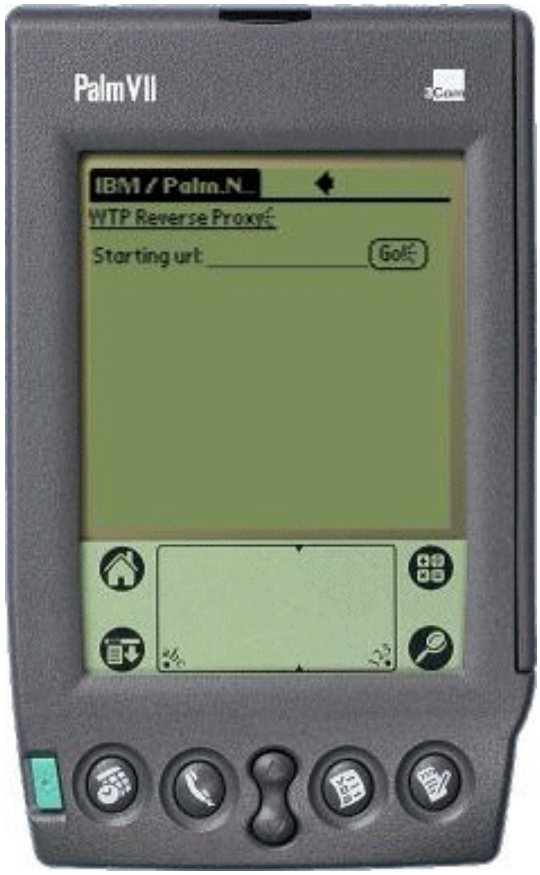
Copy the HTML sample below to a file called **wtpdemo.htm**. You can then build the .PQA application from the source HTML using the Palm.com PQABuilder. The source HTML for this application is displayed here:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
```

```
<TITLE>IBM / Palm.Net Transcoder</TITLE>
<META http-equiv=Content-Type content="text/html; charset=windows-1252">
<META name="palmcomputingplatform" content="true">
<META name="historylisttext" content="web browser &time &date">
<META name="palmlauncherversion" content="1.0">

<!-- Be sure to modify the following line to point to the
     WebSphere Transcoding Publisher server where the
     PalmInternetRedirector plugin is installed.  -->

<BASE href="http://127.0.0.1">
</HEAD>
<BODY> <A href="/">WTP</a>
<br>
<FORM name="redirect" action="/PalmInternetRedirector" method="get">
<Label for="target">Starting url:</Label>
<INPUT type="text" id="target" name="target">
<INPUT type="submit" value="Go!">
</FORM>
<BR>
</BODY>
</HTML>
```



- The palmcomputingplatform META tag must be present.
- The BASE tag defines the address of the host where WTP is installed and the PalmTranscoder plugin is installed and enabled.
- The "WTP" anchor creates a link to the WTP host. This sample uses the BASE tag above to define the address once for

all links, but you could put a fully-qualified URI here. WTP can be deployed as a Reverse Proxy or as a WebSphere Application Server filter.

- The form shows how to enter a URL on the Palm device. To use this technique, you must have installed and enabled the PalmRedirector WTP plugin, also available in this package. The user can enter a starting URL in the target input field, and tap the Go! button to send this URL to the WTP server. The action on this form must define the address of the WTP server, and the rest of the fields must be coordinated with the PalmRedirector plugin sample, also provided.

To build the PQA,

1. Customize the PQA source, according to the Palm Documentation.
2. Build the PQA, using the Palm PQA Builder.
3. Install the PQA on the device or emulator you will use to access transcoded content.

Sample PalmRedirector Plugin

If you are using a form like the one in the PQA Sample to enter a URL, you need a "Redirector" plugin like this one to receive the URL entered on the PalmOS device and redirect WTP to fetch the desired content.

Here is the Java code for the sample PalmRedirector:

```
/*
   A Transcoding Publisher plugin to redirect a URL entered on a
   form to the requested site.
*/

import java.io.IOException;
import java.util.Properties;

import com.ibm.wbi.protocol.http.*;
import com.ibm.wbi.protocol.http.beans.*;
import com.ibm.wbi.*;
import com.ibm.wbi.RequestEvent;

import com.ibm.transform.preferences.*;


public class PalmRedirector extends HttpPlugin {

   /**
   * The name of the Persistent Section that contains initialization information
   * (WBI Name, WBI Priority, WBI Condition).
   */
   private static final String SETUP_PROPERTIES = "plugins/lpp/PalmRedirector";

   public void initialize() {

       /* Just create the MEG that does the work.... */
       PalmRedirectorEditor re = new PalmRedirectorEditor();

       /* load the MEG's properties from the .prop file.    */
       re.setup(SETUP_PROPERTIES);

       try {
```

```
            addMeg(re);
         } catch (PluginError pe) {
            pe.printStackTrace();
         }
      }
}


class PalmRedirectorEditor extends HttpRequestEditor {

   public void handleRequest( RequestEvent e )
      throws RequestRejectedException, IOException
   {
      HttpRequestHeader hr =
((DocumentInfo)e.getRequestInfo()).getHttpRequestHeader();

      /* Get the data entered in the "target" input field.  If there was nothing,
         www.ibm.com is the default.  (probably not a good default)    */
      Properties data = FormHelper.interpretFormData(e);
      String url = data.getProperty("target", "http://www.ibm.com").trim();

      /* Set the URL in the HTTP request to be what we got from the form. */
      hr.setUrl(url);

//      System.out.println("Palm Redirector handling request for |" + url + "|");
   }
}
```

- The SETUP_PROPERTIES string must match where the .prop file will be installed.
- The target property name must match the name of the input field in the form in the PQA.

Here is the PalmRedirector.prop file that controls when the plugin is invoked.

```
#Properties of translator sample plugin
Class=PalmRedirector
Description=Redirects to the url entered on a Palm form
DescriptiveName=Palm form url Redirector
Major=1
Minor=1
Name=Palm Redirector
Condition=(path=*/PalmInternetRedirector*)
Priority=100
```

- The *Class* parameter must match the class of the plugin, defined in the Java code.
- The *Condition* parameter must match the action in the form in the PQA that is installed in the Palm device.
- The *Priority* parameter must be 100, so this request editor runs before the Text Engine Adjust Header Request Editor that runs at priority 99.

To build the PalmRedirector Plugin:

1. Customize the Java code, if necessary.
2. Compile the Java code, making sure that the classpath includes the required WTP libraries.
3. Customize the .prop file, if necessary.

4. Make the Plugin JAR file for the PalmRedirector:

```
jar -cf PalmRedirector.jar PalmRedirector.class PalmRedirectorEditor.class
lpp\PalmRedirector.prop
```

5. Register PalmRedirector.jar with WTP using the Administration Console.

Developer Tools

To help you extend Transcoding Publisher's capabilities by developing new transcoders, stylesheets, and preference profiles, Transcoding Publisher includes several developer tools:

- The Request Viewer provides access to configured MEGs and MEGlets and enables you to dynamically monitor the flow of requests through the transcoding server.
- The Transform Tool allows you to evaluate how an image or document would be transcoded by displaying a side-by-side comparison of the original data and the transcoded data.
- The Profile Builder steps you through the process of creating a new preference profile or editing an existing profile.
- The Annotation Editor enables you to create external annotation files.
- The XSL Stylesheet Editor enables you to create Extensible Stylesheet Language (XSL) stylesheets.

Request Viewer

The Request Viewer is a tool for monitoring the operation of the transcoding server. You can view which MEGs and MEGlets are registered with the transcoding server, along with the configuration information for the plug-ins.

The Request Viewer is useful as a debugging tool, both during the development of new transcoders, stylesheets, annotators, or preferences, and when you are diagnosing a problem. It enables you to monitor the flow of requests through the server and observe which transcoders are triggered and when they are triggered. For each transaction, the Request Viewer also displays the header and content information as they are manipulated by the transcoders.

Due to the resource requirements of the Request Viewer, IBM recommends that the Request Viewer *not* be used for debugging when working in a production environment, but rather that it be used only in a development environment. Connecting the Request Viewer to a server significantly degrades that server's performance. Do not use the Request Viewer to monitor a WTP server that has tracing turned to high.

Refer to the online help provided with the tool for more detailed information about the tool's features and using it to monitor Transcoding Publisher's request processing.

You can use the Request Viewer to monitor the WTP server on the local machine or a WTP server elsewhere in your network. A server can only be monitored by one Request Viewer at a time, either local or remote. You can have multiple Request Viewers running on the same machine as long as each is monitoring a different server. The Request Viewer can only monitor servers deployed as a proxy or reverse proxy.

Throughout this discussion, we will refer to the WTP server being monitored as the *target server*, and we will refer to the WTP server on the machine where you are running the Request Viewer as the *monitoring server*. The setup of the Request Viewer depends on the characteristics of both the target server and of the monitoring server, although the monitoring server is not involved in the monitoring process and does

not have to be running when you use the Request Viewer.

Preparing to use the Request Viewer

Before you can use the Request Viewer to monitor a WTP server, you must configure each WTP server that you want to be able to monitor remotely. You can always monitor a WTP server from a Request Viewer running on the same machine. For security purposes, each WTP server maintains a list of remote hosts from which a Request Viewer can monitor it. In other words, each target server has a list of potential monitoring servers. The *Administrator's Guide* describes how to work with this list from the Administration Console.

Starting the Request Viewer

To start the Request Viewer, do the following:

Platform	Instructions
Windows NT or Windows 2000	Click **Start->Programs->IBM Transcoding Publisher->Toolkit->Request Viewer**
AIX, Sun Solaris, or Linux	1. At the command prompt, change directory to the Transcoding Publisher installation directory. 2. Type **RequestViewer** at the prompt.

When you start the Request Viewer, its window will be empty until you connect to a server. Before connecting to a server, you must choose the correct RMI Registry server to use for the monitoring process.

Choosing an RMI Registry server

WTP servers use an RMI Registry server to receive notifications of changes from the Administration Console or central directory. When each WTP Server starts, it registers itself with its default RMI registry. If that server uses a central directory for storage of its configuration data, then its RMI Registry server is configured in the Administration Console under Notification Settings. The host name and port number for that server are configurable to avoid conflicts. If the WTP server is configured to use the local file system instead of the central directory, then the RMI registry it uses will be on the local machine. In that case, the RMI registry port number, but not the host name, is configurable via the Administration Console.

The Request Viewer uses the same RMI registry used for dynamic notification to locate and connect to the target WTP Server.

When you connect the Request Viewer to a WTP server, you must decide whether to use the monitoring server's default RMI Registry or specify a different RMI Registry. Follow these steps to choose which RMI Registry to use:

1. To monitor the WTP server on the local machine, use the default RMI Registry.
2. To monitor the WTP server on a remote machine, when the target WTP server has its configuration stored on the local machine (that is, when the target server does not use a server model), use the hostname of the target server for the RMI Registry.
3. To monitor the WTP server on a remote machine, when the target WTP server has its configuration stored in a central directory (that is, when the target server uses a server model), you must use the RMI Registry that is configured as the notification server for the target machine. If the monitoring server also uses a server model (not necessarily the same one) from the same central directory that the target server uses, it will have the same notification server as the target server, so you can use the default RMI Registry. If the monitoring server does not use a server model, or if it uses a server model from a different central directory, then you cannot use the default RMI Registry; you must check the notification settings on the target server.

To begin using the Request Viewer to monitor a WTP server:

1. From the **Actions** menu, click **Connect to Server**. The default server being monitored is the local server.
2. To monitor a remote server, select the **Remote** radio button.
3. If you have monitored the target server previously, its name should appear in the drop-down list in the **Remote Host Selection** field. Otherwise type the host name in the field.
4. Select the RMI Registry that you will use to monitor the target server as described in Choosing an RMI Registry server.

Only one Request Viewer may be connected to a server at a time. If you notice that your server is using more resources than you expect, and find that you cannot open a Request Viewer and connect to your server, search your server's **message log**for an entry which states that a Request Viewer is attached, along with a timestamp and the hostname and/or IP address of the remote viewer. You can view this log from the Administration Console by clicking **Logs->Message->View** to launch the viewer.

Disconnecting the Request Viewer

Disconnect from the server after you are done collecting the required information from it. A connected Request Viewer degrades the performance of the WTP server being monitored.

To disconnect a currently connected Request Viewer, click **Actions->Disconnect from Server**.

You can also sever a target server's connection with a remote Request Viewer. even if you do not have access to the machine running the Request Viewer. From the Administration Console of the target machine, follow these steps to disconnect a remote Request Viewer:

1. Select **Settings->Request Viewer Hosts**.
2. If you want to identify the remote WTP server from which a Request Viewer has connected, look in the transcoder trace file in the IBMTrans/logs directory on the server and find a log entry that reads "remote Request Viewer registered for name: <name>". You can also locate the hostname by viewing the transcoder trace file with the **Message Viewer**, by clicking on **Logs->Trace->View**. In the **Message Viewer**, the message reads "A Request Viewer has connected from hostname ..." When the Request Viewer disconnects, another entry reads "The Request Viewer that was

monitoring this server has disconnected." If you do not see the second entry, then you can assume that a Request Viewer is currently connected to your server.

3. In the Request Viewer Hosts dialog, you can either delete or disable the host that is connected by unchecking the checkbox of that host. You can completely disable all remote viewers by unchecking the "Allow remote hosts to view requests" checkbox.
4. Refresh the server

This will cause the server to re-verify the authorization of the currently connected Request Viewer. Since you just eliminated that authorization, the server will determine that the Request Viewer's host is no longer authorized. It will then immediately detach that Request Viewer. On the host running the Request Viewer, a message will appear that says, "The server that the Request Viewer was monitoring has detached it. A likely cause is that the administrator has revoked this host´s authority to connect to that server."

Remember that when you disable individual remote viewers or all remote viewers, Request Viewers will no longer be able to connect to this server. Re-enable the appropriate Request Viewer hosts before you try to monitor this server again.

If the Request Viewer fails to work as a remote hosted instance, check the RMI registry and see that it is running. If the RMI registry is malfunctioning and you want to open a Request Viewer to monitor the local server, you can do so by entering the RunTranscoding -g command at a command prompt. Starting the Request Viewer in this manner does not use the RMI registry.

In previous versions of WTP, you started the Request Viewer by typing **RunTranscoding -g** at a command prompt. This method is still supported, but is not recommended for normal use of the Request Viewer. When you start the Request Viewer using the **RunTranscoding -g** command, *you are also starting the transcoding server*. If Transcoding Publisher is already running (for example, as a service in the Windows NT environment), the Request Viewer will encounter port conflict errors as it attempts to initialize its own instance of the transcoding server. To avoid this problem, make sure Transcoding Publisher is not currently running before you start the Request Viewer in this manner.

If you start the Request Viewer by typing **RunTranscoding -g**, the Request Viewer does not use RMI, but starts its own internal instance of the WTP server. You cannot use a Request Viewer started in this way to monitor remote WTP servers

Transform Tool

The Transform Tool has a dual role in Transcoding Publisher: as a programming sample of how to use the JavaBean transcoders in an application and as a developer's tool for previewing the effect of various transcoding operations. The Transform Tool can show the output from the active settings on the server. By placing original content and transcoded content side by side, the Transform Tool demonstrates how a particular image or document will be affected by Transcoding Publisher's current settings. The Transform Tool does not duplicate all the function provided by the transcoding server. It uses the JavaBean transcoders to duplicate basic transformations. See the JavaBean Transcoders chapter for a list and descriptions of the JavaBean transcoders.

The Transform Tool bases its transcoding operations on the settings specified in the Transcoding

Publisher preference profiles. If you want to see which preferences have been set and which values are being used, you can open the Administration Console and examine the profiles.

Starting the Transform Tool

To start the Transform Tool, do the following:

Platform	Instructions
Windows NT or Windows 2000	Click **Start->Programs->IBM Transcoding Publisher->Toolkit->Transform Tool**
AIX, Sun Solaris, or Linux	1. At the command prompt, change directory to the Transcoding Publisher installation directory. 2. Enter TransformTool at the prompt.

Because the Transform Tool displays the current settings from the Administration Console, it might be helpful to have the Administration Console running at the same time as the Transform Tool. This enables you to make changes to the preferences in the profiles and then see their effect in the Transform Tool immediately.

Using the Transform Tool

To perform transcoding operations with the Transform Tool, you can do the following:

1. Select the content to be transcoded by choosing an option from the **File** pull-down menu. The Transform Tool enables you to choose from three kinds of data from the **Open** menu choice:
   - **Image** enables you to select a GIF or JPEG image.
   - **HTML** enables you to select an HTML document.
   - **XML** enables you to select an XML document.

   When you have selected the file, its contents will be displayed in the left pane. Although images are rendered as such in the Transform Tool, HTML and XML documents are displayed in their source format, rather than as they might be formatted by a browser.
2. Select a **Target Device** from the drop-down box.

   All of the device profiles registered with Transcoding Publisher are displayed in the list. An X on the device icon indicates that the profile is currently disabled. Note that to use the i-mode and HDML profiles, you must have enabled the i-mode transcoder and HDML transcoder, respectively. These transcoders are not enabled by default but are required to transcode data for i-mode or HDML devices.
3. Select a **Target Network** from the drop-down box.

   All of the device profiles registered with Transcoding Publisher are displayed in the list. An X on the device icon indicates that the profile is currently disabled.

4. Type the information for a **Target User** in the User field.

   The information you type here must identify a user profile that is either stored in your central directory or supplied by your own programming interface. If the profile is stored in your central directory, the entry should be of the form <name>@<realm>. An active connection to your central directory is required in order to retrieve user information. If the profile is supplied by your own programming interface, the entry should be of the form <name>.

5. Select **File->Transcode** to start the transcoding operation. The transcoded data will be displayed in the right pane. As with the untranscoded content, images are rendered appropriately, and HTML and XML output is shown in its source format.
6. To keep the transcoded output for later use, select **File->Save Transcoded Content**.
7. Select **File->Exit** to close the tool.

Using XML Stylesheets with the Transform Tool

Transcoding Publisher includes several sample XML stylesheets you can use with the Transform Tool to observe how data can be transcoded from XML into HTML and other varieties of XML, such as WML. To convert XML data through the use of stylesheets, you must ensure that each required stylesheet has been registered with Transcoding Publisher. Stylesheets do not need to be registered prior to starting the Transform Tool, because the Transform Tool dynamically makes use of the active settings of Transcoding Publisher.

To register the XML stylesheets, do the following:

1. Start the Administration Console.
2. For each XML stylesheet you want to register, select a folder in which to store the stylesheet (such as the XML Stylesheet Selectors folder) and then use the **Register->XML Stylesheet** option to start the registration wizard. More detailed instructions for using the console are available from the online help.

   The following information should be specified in the appropriate places in the wizard

   - **Location:** The sample files are located in the toolkit\stylesheets\samples directory under the Transcoding Publisher installation directory. There is a sample file for each device type supported by the Transform Tool.
   - **Output content type:** Specify text/html for each stylesheet, with the exception of the one for WML, which takes text/vnd.wap.wml.
   - **Input DTD:** Specify Flights for each stylesheet.

   A sample XML source file has also been provided in toolkit\stylesheets\samples\FlightInfo.xml.
3. Update the transcoding server by selecting **File->Refresh Server**.

The Transform Tool is able to detect dynamic updates from Transcoding Publisher and enables you to see the output from the active server settings. It can be used simultaneously with the Administration Console, so you may change settings in the Administration Console.

Profile Builder

You can use the Profile Builder to create new preference profiles or to modify existing profiles. For information about preference profiles and how WTP uses them, please refer to the Overview section of the *Administrator's Guide*.

To start the Profile Builder:

Platform	Instructions
Windows NT or Windows 2000	Click **Start->Programs->IBM Transcoding Publisher->Toolkit->Profile Builder**
AIX, Sun Solaris, or Linux	1. At the command prompt, change directory to the Transcoding Publisher installation directory. 2. At the prompt, type ProfileBuilder.sh.

The Profile Builder steps you through the process of creating or changing a profile. Each panel provides descriptive text explaining each field and what it is for.

The Profile Builder will display all the preferences available for the selected type of profile. When you hold your pointer over the name of a preference, the Profile Builder will display an explanation of the preference. For each preference, you can specify:

- Whether the preference should be included in this profile. If a preference is not important for this network or device type, do not include it. Then WTP can select a value from a profile where the preference is important. For example, a preference might not be important to a network type; by omitting it from the network profile, you can allow WTP to derive its value from the chosen device profile. Refer to the *Administrator's Guide* for information about how preferences are resolved.
- Whether the preference should be configurable through the Administration Console or only displayed. You might want to include a preference in the profile but not want administrators to be able to change the value you set. If you include a preference in the "Edit preference" group in the Profile Builder, an administrator can change the value of the preference in the Administration Console. Including a preference in the "View only" group will still cause the preference to be displayed in the Administration Console, but an administrator will not be allowed to change the preference's value.
- The value for the preference. The default value is provided, but you can change it to any value appropriate for this network or device type.

Specifying a User-Agent Value

When you create a new device preference profile, it will be selected for use by comparing the user-agent value that you specify with the value in the User-Agent field in the HTTP header. If the information in

the User-Agent field is not sufficient to differentiate between devices for your profile, you can also match against information in the Accept header.

It is not necessary to enter the complete value of a header , such as Mozilla/4.6 [en] (WinNT; U); instead, you can use a wildcard character (*) to match on a selected part of the value, as in Mozilla/4.* or *MSIE 5.*. The wildcard character can be used only at the beginning and end of the value; it cannot be used in the middle.

You can combine several matching clauses with parentheses to construct a Boolean expression. To negate an argument, precede it with an exclamation point (!). The operators AND (&) and OR (|) can be used between clauses.

For example, Transcoding Publisher provides a profile for use with a device that renders information in Handheld Device Markup Language (HDML) format. The following rule is used with this profile:

```
(User_Agent=*UP.Browser/3.0*) |
   (((User_Agent=*UP.Browser/3.1*) |
     (User_Agent=*UP.Browser/4.*) | (User_Agent=*UP/4.*)) &
     (Accept=*text/x-hdml*) & !(Accept=*wml*))
```

Because some devices that use Wireless Markup Language (WML) also use the UP.Browser, matching clauses on the Accept header have been added to ensure that this profile is only applied to requests from devices that accept HDML.

Implementing your changes

When you modify an existing profile and save your changes, the profile is changed immediately, whether it is stored locally or in a server model. However, the WTP server is not immediately notified of the change. When you want your WTP server to use the updated profile, Click **File->Refresh server**.

When you create a new preference profile, store it in a temporary location that is not part of the Transcoding Publisher directory structure. Click **Register->Preference profile**. to register the new profile through the Administration Console. Refer to the Administering section of the *Administrator's Guide* for information about registering new profiles, refreshing the server, and related topics.

Annotation Editor

You can use the Annotation Editor to create annotators, which are files used by WTP to select the content to be kept or removed from a source file.

Starting the Annotation Editor

To start the Annotation Editor, do the following:

Platform	Instructions

Windows NT or Windows 2000	click **Start->Programs->IBM Transcoding Publisher->Toolkit->Annotation Editor.**
AIX, Sun Solaris, or Linux	1. At the command prompt, change directory to the Transcoding Publisher installation directory. 2. Type AnnotationEditor.sh at the prompt.

Using the Annotation Editor

With the Annotation Editor, you can select the content that will be affected by annotation, and how that content will appear:

- To keep the selected content just as it is after transcoding, select **Keep.**
- To remove the content from the document during transcoding, select **Remove.**
- To replace the contents from a portion of the HTML document, select **Replace.**
- To define whether a table is kept or removed, or whether rows or columns are marked as kept or removed, select **Table.**
- To construct more complex annotations than those provided above, an expert user may select **Custom**.

When you have selected the file, its contents will appear in the left pane. In the left pane of the Annotation Editor, you can display either the **Source** of the HTML file that you've opened or created, or a hierarchal **Outline** view that shows the elements that make up the page you're annotating. In the right pane is the **.ann** annotation file you're creating, both with the Source display of the Annotation file along with the hierarchal view of the Annotation.

1. Open an HTML file that contains the elements you want to apply annotation to from the **File>Open HTML** menu. You may open an HTML file that is either on a local machine, or a file that is out on the internet.
2. Open an annotation (**.ann**) file, or create a new annotation file. This is the file that contains annotation that you can apply to any or all of your HTML files.
3. To apply annotation to an element of an HTML page, select the element in the left pane. In the **Outline** view, you may single-click on an element to select it. In the **Source** view, you may click and drag the moust to select an element from the source of the HTML document. Then choose **Annotation>Annotate** from the **File** menu and pick from the options, **Keep**, **Remove**, **Replace**, **Table**, or **Custom**.

If you have previously used the Annotation Editor to work on HTML and Annotation files, and open a previously opened HTML file, the associated annotation file will open. The reverse is also true: opening a previously edited annotation file will cause the associated HTML file to open. This source target check can be disabled by editing the eae.properties file in the com\ibm\transform\toolkit\annotation directory of your WTP installation. Before any changes made in the eae.properties file can take effect, you must save the file and restart the Annotation Editor.

Annotating Tables

Before you apply annotation to a table in your HTML document, you should decide how you want the table to appear after annotation. When you click **Annotation->Annotate->Table** the **Table Annotation Wizard** will open, and you will have to

1. Choose whether the table is kept or removed by default.
2. Select rows to **keep** or **remove**,
3. Select columns to **keep** or **remove**.
4. When you are done, press **Finish.**

The wizard will add keep and remove annotations to the annotater being edited in the right pane of the Annotation Editor. Note that the output annotation elements do not use the **table** tag. This tag is recognized by the editor, but is not used.

Customizing Annotation

This dialog is intended for expert users who want to construct more complex annotations than those that are created by the built-in annotation functions. This dialog provides control over the XPath syntax used to generate the XPath expression for the target attribute, the value of the take-effect attribute, and the annotation content structure. The dialog is divided into two main sections: the **Description Attributes Pane** and the **Description Content Pane**.

Description Attributes Pane
The top section of the dialog is used to enter or modify the values for the attributes of the description tag that will be generated for this custom annotation. A pop-up menu and a dynamic content area are used to select fast-paths and enter any required information particular to a given fast path in order to create common types of XPaths (or as building blocks for more complex XPaths) for use as the value of the target attribute. Another pop-up menu is used to allow the user to specify the value of the take-effect attribute.

Description Content Pane
The bottom section of the dialog is used to enter or modify the content structure of the description tag. The current mechanism for performing this action is to manually enter annotation XML into a text area. If this information is specified, the source is validated and upon successfull validation, appropriate annotation elements are created in the Annotation DOM. If this information is left blank, only the description tag will be generated using the values specified in the Description Attributes Pane.

The dialog also contains an "OK" button and a "Cancel" button that either commits or discards the changes, respectively.

Navigating Your XML Using the Xpath Expression Address Bar

If you choose, you may navigate your XML by using the Xpath Expression Address Bar. This is a text field where you may enter an Xpath expression, and the associated element set will be selected in the target HTML pane. The previous entries in the Xpath Expression Address Bar are retained, and can be used by clicking on the down arrow of the text entry field. More information on Xpath expressions is

available from the W3C recommendation of XML Path Language 1.0.

Using the Attributes Pane

The Attributes Pane of the Annotation Editor is at the bottom of the editor and spans the width of the application. When the editing panes are in outline view and you select an element, the name of the element will appear on the left side of the Attributes Pane, and the values for that element will appear on the right side of the Attributes Pane. When you select an element from the right pane of the .ann file you are working on, the name of the element will appear on the left side of the Attributes Pane, and the value for that element will be editable on the right side of the Attributes Pane.

XSL Stylesheet Editor

The XSL Stylesheet Editor is a graphical tool that helps you develop XSL stylesheets, which you can use to manipulate XML documents. Using the Stylesheet Editor, you can work with an XML document and create a new stylesheet for it, or you can edit existing stylesheets.

The XSL Stylesheet Editor runs only on Windows NT and Windows 2000. To start the Stylesheet Editor, click **Start->Programs->IBM Transcoding Publisher->Toolkit->Stylesheet Editor**

Some of the features that the Stylesheet Editor supports include:

- The ability to view the stylesheet and XML document simultaneously, alongside an XHTML view that approximates what the resulting output would look like
- The ability to view the stylesheet and XML document in both a hierarchical tree view and a text view
- Point and click capability to create and manipulate template rules and element formatting
- Support for advanced users who want to use the XML Path Language (XPath) to identify and work on specific elements in XML documents

For detailed information on using the Stylesheet Editor, refer to the online help provided with the tool.

Using XML configuration to work with resources

WebSphere Transcoding Publisher (WTP) now offers a set of commands that enables you to:

- Preserve some configuration changes when you install a new version of WTP
- Copy and modify WTP resource definitions without using the WTP Administration Console
- Copy WTP resource definitions from one WTP server to another
- Import an existing cocoon.properties file, to enable you to migrate to WTP from the Cocoon application.
- Define your network's RMI Registry server, which is used to notify WTP servers of changes to server models

XML configuration creates an XML-based definition of your WTP resources that can be imported into any WTP server. The resources whose information can be manipulated include the four types of WTP resources defined in the *Administrator's Guide*:

- Annotators
- Preference profiles
- Plugins (transcoders)
- Stylesheets

plus the Setup resource, which includes information about local settings.

XML Configuration performs two basic tasks:

- Copying resource information from a WTP server or server model to an XML file (exporting)
- Copying resource information from an XML file to a WTP server or server model (importing)

XML Configuration provides several variations of each of these two tasks.

Note that XML configuration does not make copies of the resources themselves: transcoders (.jar files), stylesheets (.xsl files), or annotators (.ann files). The one exception is preference profiles, which are copied in their entirety. For the other resources, XML configuration copies stylesheet selectors, annotator selectors, and transcoder properties. This means that if you want to copy resources and their definitions from one machine to another, you must copy the resource files themselves, or make them available from a Web server, before you import their definitions onto a new machine.

The file produced when you export or back up resource definitions can be in either of two formats:

- WTP-dependent format, which consists of XML created directly from WTP definitions (in the /etc directory tree or a server model) without changing their structure or names
- WTP-independent format, which consists of XML in a more readable format, which contains the same information but not necessarily using the same structure and names

When you export information in one of these formats, you must re-import it using a command that supports the same format.

Use the WTP-independent format if you want to:

- Edit the XML file to modify WTP resource definitions.
- Migrate resource definitions from one version of WTP to another.
- Migrate resource definitions from one WTP server or server model to another.

Use the WTP-dependent format for regularly scheduled backups of your WTP resource definitions and settings, or to work with resources that cannot be used with the WTP-independent format. This file can be used only to restore these resource definitions on the same machine, or one with the same WTP version and server model. Remember that these commands do not back up the resources themselves: transcoders (.jar files), stylesheets (.xsl files), or annotators (.ann files).

XML configuration commands

XML configuration provides these commands. Click the command name to see the arguments for that command and other information about its use. You can enter any command with -help or -? to see a summary of the valid arguments.

Command	Function

ExportResources	Exports WTP resources to an XML file in a WTP-independent format, or in a WTP-dependent format if the -Node argument is specified.
ImportResources	Imports WTP resources from an XML file in a WTP-independent format, or in a WTP-dependent format if the -Node argument is specified.
BackupResources	Exports WTP resources to an XML file in a WTP-dependent format. This file can be used only to restore these resource definitions on the same machine or another with the same WTP version (that is, it cannot be used for migrating from one WTP release to another). If the WTP server whose resources are backed up uses a server model, the server to which it is restored must use the same server model.
BackupConfig	Exports WTP resources to an XML file in a WTP-dependent format. Includes Setup resources as well as those exported by BackupResources. This file can be used only to restore these resource definitions on the same machine.
RestoreResources	Restores WTP resources from an XML file created by the BackupResources command.
RestoreConfig	Restores WTP resources from an XML file created by the BackupConfig command.
ImportCocoon	Imports a Cocoon properties file to support stylesheet specifications within XML documents.

Log entries will be written to the WTP log file, logs/TranscoderMessages.log, when any import or export command is issued. Entries will be written to the XML configuration trace file, logs/cmdmagictrace.log, whenever an exception is encountered.

Using XML configuration as a migration tool

On Windows NT, Windows 2000, AIX, and Solaris, the WTP V4 installation process will offer you the opportunity to save configuration changes before you install WTP V4, and reimport those changes at the end of the installation process. On Linux, you must perform the migration process manually. You can use this process on the other platforms if you wish.

You can use XML configuration to preserve some configuration changes when you install a new version of WTP. The migration process migrates stylesheet selectors, annotator selectors, transcoder properties, device profiles, and network profiles. It does not migrate transcoders, stylesheets, or annotators.

Changes you have made to existing values will not be preserved. For example, if you have modified a value in a preference profile, that value will not be preserved. Any new values you have added will be preserved. For example, if you added new key/value pairs to device profiles and stylesheets, those key/value pairs will be preserved. The definitions of new resources that you have added will also be preserved.

You can use XML configuration to migrate from WTP version 1.1.2 or 3.5 to WTP version 4.0. Because XML configuration was not included in those releases, you must copy it from the WTP installation CD in order to save the configuration changes you have made.

To migrate your configuration, follow these steps:

1. Copy the file xmlconfig.jar from the instmgr directory on the WTP installation CD to the directory where you currently have WTP installed.
2. Extract the files from the jar file: jar -xvf xmlconfig.jar
3. On Unix platforms, add execute permission to all the scripts: chmod +x *.sh
4. Run **ExportResources** from the WTP installation root directory to back up stylesheets, annotators, device profiles, network profiles, user profiles, and plugins (transcoders) to the **WTPResources.XML** file. A message will be written to the console to confirm that the command was executed successfully.
5. Copy the WTPResources.XML file to another directory outside the WTP directory structure so that it will not be erased when you install WTP V4.
6. Copy any plugins (.jar files), stylesheets (.xsl files) and annotators (.ann files) that you had added to the previous version of WTP to a location outside the WTP directory structure. You must preserve the directory structure in which these files reside so that the import process can locate them.

7. Install WTP Version 4.0.
8. Copy the XML file created in step 2 into the WTP root directory. Copy the resource files (.jar, .xsl, .ann) back into the WTP directory structure.
9. Run **ImportResources** from the WTP installation root directory to import stylesheets, annotators, device profiles, network profiles and user profiles from WTPResources.xml. A message will be written to the console to confirm that the command was executed successfully.

Using XML configuration to copy and modify resources

If you are comfortable with editing XML property files, you can use XML configuration to export existing resource definitions and to re-import them after they have been modified. You can work with resources defined on the local machine or in a server model that you are editing. XML configuration can work with these types of WTP resources:

- Setup (includes basic WTP settings: Firewall, Proxy Port, Reverse Proxy and Server setup. Does not include WAS settings)
- Stylesheet
- Annotator
- Device (preference profile)
- Network (preference profile)
- User (preference profile)
- Plugin (transcoder)

For example, if you have created a group of stylesheets that will be applied in similar but not identical situations, you could:

1. Place the stylesheets (.xsl files) on your WTP server or on a Web server
2. Register one stylesheet through the WTP Administration Console
3. Export your stylesheet resources to an XML file
4. Use an XML or text editor to make several copies of the stylesheet definition and modify each one to represent one of your new stylesheets
5. Import your updated stylesheet resources into WTP

For another example, suppose you have already registered a set of annotators to be applied to a set of documents on a certain Web server, and the documents are moved to a different domain, or even to a new directory on the server. You need to make the same change to each URL used to select one of these annotators. Rather than modifying each annotator in the Administration Console, you could:

1. Export your annotator resources to an XML file
2. Use an XML or text editor to find and replace the necessary strings in the URLs
3. Import your updated annotator resources into WTP

You might find that you need to modify the properties of a transcoder. For example, you might need to add a new translation server to work with the machine translation transcoder. To modify transcoder properties:

1. Export the one transcoder's properties to an XML file:

   ```
   ExportResources -ResourceType "plugin, MachineTranslationTranscoder"  -File "MTT.xml"
   ```
2. Use an XML or text editor to make your updates.
3. Import your updated transcoder properties into WTP:

   ```
   ExportResources  -File "MTT.xml"
   ```

   Since the file contains only the one resource definition that you want to import, you do not have to specify the -ResourceType argument.

Some transcoders cannot be modified in this way. If ExportResources fails when you specify the name of a transcoder, you can work with the WTP-dependent form of the transcoder definition by using the -Node argument. To modify transcoder properties using -Node:

1. Locate the property file for the transcoder in the /etc/plugins directory tree.

2. Export the one transcoder's properties to an XML file using the -Node argument with the transcoder's property file as the value:

   ```
   ExportResources -Node plugins/ibm/TextEngine/XMLHandler  -File "XMLHandler.xml"
   ```

   Do not include the ".prop" file extension.
3. Use an XML or text editor to make your updates
4. Import your updated transcoder properties into WTP:

   ```
   ImportResources -Node "*"  -File "XMLHandler.xml"
   ```

   If you specify -Node on the ExportResources command, you must also specify it on the ImportResources command. Specifying -Node "*" means to import all resources in the file.

Copying your resource definitions to another machine

After you have configured one WTP server, you can use XML configuration to copy the resource definitions to another machine. Remember that XML configuration does not copy the resources themselves (transcoders, annotators, or stylesheets). This means that you must make sure that the resources are available on your target machine in exactly the same location as on your original machine. One way to accomplish this is to place the resources on a Web server instead of on the WTP server. Then the resources can be accessed using HTTP by any WTP server. If you do not want to use a Web server, copy any resource files to the target machine before you import the resource definitions.

To copy resource definitions from one WTP server to another, follow these steps:

1. Install WTP 4.0 on the first server.
2. Register any new resources and configure your WTP resources using the Administration Console and/or XML configuration.
3. Use the ExportResources command or the Administration Console to export your resource definitions to an XML file.
4. Install WTP 4.0 on the second (target) server.
5. If you added any new resources to the first server (annotators, stylesheets, or transcoders), copy any local files (xsl, ann, and jar files) to the target server. Skip this step if the files are located on a Web server.
6. Copy the XML file created in step 3 to the target server.
7. Import the resource definitions from the XML file on the target server. Resolve any errors, such as files not found.

Copying your resource definitions into a server model

If you use server models to store WTP server configurations, you might want to fine tune your configuration on a server with a local configuration before storing it in a server model. If you want to do this, follow these steps:

1. Install WTP 4.0 on your test machine, specifying that its configuration is stored in the local file system.
2. Configure your server's resources as you want them. Test your configuration to be sure it is correct.
3. Use the Administration Console or the ExportResources command to export your resources to an XML file. You cannot import settings into a server model.
4. Move the XML file to a machine with an Administration Console configured to work with your central directory. If you do not have such a machine, you can re-install WTP 4.0 on your test machine, either as a server whose configuration is stored in a server model, or as an Administration Console only. If you plan to re-install, be sure to move your XML file to a location outside the WTP install directory.
5. At an Administration Console configured to work with your central directory, edit the server model that you want to configure. Use the Administration Console or the ImportResources command to import the resources stored in your XML file.
6. If you added any new transcoders, you must register them through the Administration Console on one WTP server that uses this server model after you import their definitions. When you register the transcoder, you will see a message saying that the transcoder already exists and asking if you want to continue. Say yes and complete the registration process. The central directory will make the transcoder available to all the servers using that server model. If you prefer, you can delete the definitions of any new transcoders from your XML file before importing it and add them to the server model by registering them.

XML configuration command arguments

These tables show the arguments that can be used with each XML configuration command.

ExportResources arguments

Arguments for ExportResources		
Argument	**Default value**	**Description**
-File [filename]	WTPResources.XML	Name of the file to which to export the resources
-Append	no	Append the new resources to the specified file. Duplicate resources will be overwritten. If this argument is not specified, the file will be overwritten.
-ResourceType [type, names]	all types	Export the resources of the specified types (stylesheet, annotator, plugin, device, network, or user). The default is to export resources of all these types. If names are specified after the type, resources of that type with those names will be exported.
-Node [node_name]	none	Export only the definitions of the specified resources. Give the path of the resource definition, relative to *IBMTrans*/etc (where *IBMTrans* is the WTP installation directory), such as "plugins/ibm/TextEngine/XMLHandler". This argument causes the file to be created in WTP-dependent format. You must specify -Node when you import resources from this file.
-Comment [text]	none	Add the specified text to the file as a comment.
-Encoding [value]	UTF-8	Encoding to be used for the exported XML. Other available values will depend on your JDK.
-Help	false	Displays help for the command.
-Debug	false	Writes messages to the console as records are written to the output file.

For example, the following two commands produce a file called MyResources.xml with all stylesheet selectors and device profiles: ExportResources -ResourceType stylesheet -File MyResources.xml ExportResources -ResourceType device -append -File MyResources.xml

The following command produces a file called MyResources.xml with the specified two device profiles and two annotator selectors: ExportResources -ResourceType "device, WML-Device, HDML-Device" -ResourceType "annotator, Raleigh, Durham" -File MyResources.xml

ImportResources arguments

Arguments for ImportResources		
Argument	**Default value**	**Description**
-File [filename]	WTPResources.XML	Name of the file from which to import the resources

-ResourceType [type, names]	all types	Import the resources of the specified types (stylesheet, annotator, plugin, device, network, or user). The default is to import all the resources found in the file. If names are specified after the type, resources of that type with those names will be imported.
-Node [node_name]	none	Use this argument if the file was created by ExportResources with -Node specified. Use **-Node "*"** to import all resources in the file.
-Encoding [value]	UTF-8	Encoding to be used for the imported XML. Other available values will depend on your JDK.
-Help	false	Displays help for the command.
-Debug	false	Writes messages to the console as records are written to the output file.

Run ImportResources.bat or ImportResources.sh from the WTP root directory to import the changes made in the previous step to WTP. If the resource already existed, the old values will be overwritten unless you specify the -NoImportOverwrite argument on the command line.

For example, if you created MyResources.xml with the two export commands above, then **ImportResources -File MyResources.xml** would import all the resources in the file, while **ImportResources -ResourceType device -File MyResources.xml** would import only the device profiles contained in the file.

Note: When you import resource definitions, WTP will attempt to verify the existence of all local files referred to by the definitions. For example, when you import stylesheet selectors, WTP will check whether the stylesheet files identified in the selectors exist in the specified locations. If the files are not found, WTP will issue a warning message. Files identified using http:// rather than file:// are not verified.

Executing instructions in the XML document

You can add processing instructions in an XML document before importing it into WTP, and the import process will execute the instructions as it imports the file. The supported instructions are:

Instruction	Function
<?Delete?>	If specified within a resource definition, delete all resources with the SelectorName value defined for that resource. If specified within a folder definition, delete the folder and all resources contained within it. Remove the registration entry for the deleted resources.
<?Refresh Server?>	Issue a command to the WTP server to reload all resources of the type within which this command is included. You only need to include this instruction once per resource type, even if the XML file includes several resources of that type.

BackupResources arguments

The BackupResources command always exports definitions of all resources of all types except Setup. You cannot append to an existing file; if you specify the name of an existing file, it will be overwritten. The default file name is WTPResources_*datetime*.xml, where *datetime* is the system date and time when the command is issued. The default location is *IBMTrans*/backup/resource, where *IBMTrans* is the WTP installation directory.

Arguments for BackupResources		
Argument	**Default value**	**Description**
-File [filename]	*IBMTrans*/backup/resource/ WTPResources_*dt*.XML	Name of the file to which to back up the resource definitions (see explanation above)

-Comment [text]	none	Add the specified text to the file as a comment.
-Encoding [value]	UTF-8	Encoding to be used for the exported XML. Other available values will depend on your JDK.
-Help	false	Displays help for the command.
-Debug	false	Writes messages to the console as records are written to the output file.

BackupConfig arguments

The BackupConfig command always exports definitions of all resources of all types, including Setup. You cannot append to an existing file; if you specify the name of an existing file, it will be overwritten. The default file name is WTPConfig_*datetime*.xml, where *datetime* is the system date and time when the command is issued. The default location is *IBMTrans*/backup/config, where *IBMTrans* is the WTP installation directory.

Arguments for BackupConfig		
Argument	**Default value**	**Description**
-File [filename]	*IBMTrans*/backup/config/ WTPConfig_*dt*.XML	Name of the file to which to export the configuration data (see explanation above)
-Comment [text]	none	Add the specified text to the file as a comment.
-Encoding [value]	UTF-8	Encoding to be used for the exported XML. Other available values will depend on your JDK.
-Help	false	Displays help for the command.
-Debug	false	Writes messages to the console as records are written to the output file.

RestoreResources arguments

Arguments for RestoreResources		
Argument	**Default value**	**Description**
-File [filename]	none	Name of the file from which to restore the resource definitions (required)
-Encoding [value]	UTF-8	Encoding to be used for the XML. Other available values will depend on your JDK.
-Help	false	Displays help for the command.
-Debug	false	Writes messages to the console as records are read from the input file.

RestoreConfig arguments

Arguments for RestoreConfig		
Argument	**Default value**	**Description**

-File [filename]	none	Name of the file from which to restore the configuration data (required)
-Encoding [value]	UTF-8	Encoding to be used for the XML. Other available values will depend on your JDK.
-Help	false	Displays help for the command.
-Debug	false	Writes messages to the console as records are read from the input file.

Importing Cocoon properties

If you have been using the Cocoon application, you can import your cocoon.properties file into WTP:

```
ImportCocoon -File <path>/cocoon.properties
```

See Specifying stylesheets within a document for information about migrating from Cocoon to WTP.

Updating Setup Resources

If you need to update a local setting, such as the RMI Registry used by your WTP server, you can export the setup resource, modify it as needed, and re-import it.

Here is an example of the file created by exporting the setup resource. This example was created on a WTP server configured as a network proxy using a server model called "al" and a socks server as a firewall.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Date and Time of export :Wed Jul 25 16:14:00 EDT 2001-->
<Resources ServerModel="al" Version="magic40">
<Setup>
<RunWTPAs>proxy</RunWTPAs>
<ServerModelToBeUsed>/\al</ServerModelToBeUsed>
<LDAP>
<Server>wtppro</Server>
<User>cn=wtppro</User>
<Port>389</Port>
<EncryptedPassword>DKM-2g-_eHK</EncryptedPassword>
</LDAP>
<RMIRegistry>
<Enable>true</Enable>
<HostName></HostName>
<Port>1099</Port>
</RMIRegistry>
<Configuration>
<Firewall>
<Socks>
<Enable>true</Enable>
<Server>socks</Server>
<Port>1080</Port>
</Socks>
<Proxy>
<Enable>false</Enable>
<Server>your.proxy.server</Server>
<Port>80</Port>
<NoProxy>*.your.local.domain</NoProxy>
</Proxy>
</Firewall>
```

```
<ReverseProxy>
<TranscodingServerHostName></TranscodingServerHostName>
<SingleWebServerHostName></SingleWebServerHostName>
<MultipleWebServerPage></MultipleWebServerPage>
</ReverseProxy>
</Configuration>
</Setup>
</Resources>
```

Data Transformation

This section describes how HTML documents are modified during conversions into other formats. These conversions are automatically performed by Transcoding Publisher when the output content type for a document is specified as WML, HDML, or i-mode's Compact HTML. These conversions are also performed when using the VoiceXML Transcoder or Palm Transcoder to product output for VoiceXML capable browsers or Mobile Internet Kit equipped devices with the Palm.Net service.

Data transformation can also be used in conjunction with custom transcoders called text clippers to provide customized documents for specific types of devices. For more information on document clipping and how data transformation fits into it, read Document Clipping.

Generic WML Transcoding

This section describes the transformations performed in generating a WML document (and DOM) from the original HTML document, as well as which tags are deleted or replaced.

Transformed HTML Tags	
Tag Name	Transformation Description
Anchor (A)	If the BASE attribute was stored from the previous Head element (see description of Head element below), a relative path is prefixed with the base value. If the TARGET attribute is specified on the Anchor element it is removed. Also, if no HREF attribute is specified the entire element is removed.
Body (BODY)	The BODY element should be the first element found under the HTML element and is replaced with a WML card element with a P element as its child. All the children of the BODY element are moved under the created P element.
Bold (B)	If the Bold (B) element is a child of an Anchor element, the children are removed from the node and put in its place. This is because in WML an Anchor is not allowed to have a Bold element as a child. Moving up the children in this way has the effect in the output of looking like the outer tag (in this case the <B> and </B>). If the Bold element has a FORM HTML element as a child, the children are likewise moved up in place of the Bold element.
Break (BR)	If the Break element is a child of a WML or TD element, it is removed. Otherwise, only the attributes are removed.
Comment (COMMENT)	Comment elements are preserved in the DOM form of the document but removed if the document is output as a string from the DOM.
Definition List (DL, DT, DD)	The children are moved up in place of the Definition List element. Break elements are inserted before the children that were moved up, if they do not automatically cause a break.
Font (FONT)	If the FONT element is a child of an Anchor (A) or Strong (STRONG) node, the children are moved up in place of the element. Otherwise, the FONT element is turned into a STRONG element.

Form (FORM)	The FORM element usually contains INPUT and SELECT elements. The FORM element is traversed and if a POST or GET action is found and an INPUT with a VALUE specifier is found, a WML DO element (with an equivalent INPUT element) is created. Similarly, an equivalent SELECT element is created if the HTML SELECT has a VALUE specifier. The following TYPE attributes on the INPUT element are currently supported: "submit," "password," "text," "radio," "hidden," and "checkbox." If TYPE is not specified, "text" is assumed.
Frame (FRAME)	Replace with a link to the target to which the FRAME was pointing.
Frameset (FRAMESET)	Replace with a link to the target to which the FRAMESET was pointing.
Head (HEAD)	If the HEAD element specifies the BASE attribute, this value is saved in the MegContext object for later use in processing the Anchor element.
Heading (H1...H6)	The Heading element is replaced with a Bold element. In addition, Break elements are created and placed before and after the heading.
HTML (HTML)	The HTML element is replaced with a WML element, and the children of the HTML element are moved underneath the new WML element.
Image (IMG)	The Image element is removed if there is no alternate text (ALT attribute) specified. If the "Convert images into image links" preference is true, the image is replaced with a link to the image using its alternate text, if available. Any BORDER and USEMAP attributes are also removed.
List (UL, OL, LI)	The children are moved up in place of the List element. Break elements are inserted before the children that were moved up, if they do not automatically cause a break.
Paragraph (P)	If the Paragraph element is not the child of the newly created CARD element, Break elements are inserted before and after the children of the P element, and the P tags themselves are removed.
Small (SMALL)	If the Small element is a child of an Anchor (A) element, the Small element is moved up in the Anchor's place. Otherwise, the node is unchanged.
Table (TABLE)	The Table element is processed by first moving up the elements of nested tables, which are not allowed in WML. Table Header (TH) elements, which are not supported in WML, are replaced by TD elements. Those tables that are not nested have the elements moved up in place of the table when: ● They contain elements that are not allowed in WML tables. ● They have only one column. ● They have more than 3 columns.

Deleted Tags		
Applet (APPLET)	Layer (LAYER)	Script (SCRIPT)
Area (AREA)	Link (LINK)	Strike (STRIKE)
Base Font (BASEFONT)	Map (MAP)	Style (STYLE)
Background Sound (BGSOUND)	No Frames (NOFRAMES)	Title (TITLE)
Delete (DEL)	Object (OBJECT)	
Horizontal Row (HR)	Strike (S)	

Tags Replaced by Their Children		
Address (ADDRESS)	Floating Frame (IFRAME)	Quoting (Q)
Block Quote (BLOCKQUOTE)	Insert (INS)	Sample (SAMP)
Center (CENTER)	Keyboard (KBD)	Subscript (SUB)
Cite (CITE)	Legend (LEGEND)	Superscript (SUP)
Code (CODE)	Label (LABEL)	Typewriter Text (TT)
Division (DIV)	Map (MAP)	Underline (U)
Fieldset (FIELDSET)	Preformatted (PRE)	

Generic HDML Transcoding

This section describes the transformations performed in generating an HDML document (and DOM) from the original HTML document, as well as which tags are deleted or replaced.

Transformed HTML Tags	
Tag Name	Transformation Description
Anchor (A)	Set the HDML DEST attribute to the contents of the HREF element, and set the TASK attribute to "GO." All other attributes are removed.
Body (BODY)	Replace with a DISPLAY element. All the children of the BODY element are moved under the created DISPLAY element.
Break (BR)	Remove all attributes.
Caption (CAPTION)	Handled as part of converting the associated table into an unordered list. Refer to the TABLE element.
Comment (COMMENT)	Comment elements are preserved in the DOM form of the document but removed if the document is output as a string from the DOM.
Directory List (DIR)	Replace with children and place break (BR) elements between children.
Form (FORM)	The FORM element usually contains INPUT and SELECT elements. The FORM element is traversed and if a POST or GET action is found and an INPUT with a VALUE specifier is found, a HDML ENTRY element is created. Similarly, an equivalent CHOICE element is created if the HTML SELECT has a VALUE specifier. The following TYPE attributes on the INPUT element are currently supported: "submit," "password," "text," "radio," "hidden," and "checkbox." If TYPE is not specified, "text" is assumed.
Frame (FRAME)	Replace with a link to the target to which the FRAME was pointing.
Frameset (FRAMESET)	Replace with a link to the target to which the FRAMESET was pointing.
Head (HEAD)	If the HEAD element specifies the BASE attribute, this value is saved in the MegContext object for later use in processing the Anchor element. The HEAD element and its children are then removed.

HTML (HTML)	The HTML element is replaced with an HDML element, specifying version 3.0. The children of the HTML element are moved underneath the new HDML element.
Image (IMG)	Add alternate text (ALT attribute) if none is specified. If the "Convert images into image links" preference is true, the image is replaced with a link to the image using its alternate text. Only SRC and ALT attributes are supported.
Input (INPUT)	Handled as part of the FORM element.
List (UL, OL, LI)	Replace with children and place break (BR) elements between children.
Menu (MENU)	Replace with children and place break (BR) elements between children.
No Script (NOSCRIPT)	Use in replacing SCRIPT element.
Option (OPTION)	Choice provided by OPTION element converted to CHOICE element of FORM.
Paragraph (P)	Replace with children and place Break (BR) element in front of P.
Selection Menu (SELECT)	Choice provided by SELECT element converted to CHOICE element of FORM.
Span (SPAN)	Replace with children and place Break (BR) element in front of SPAN.
Table (TABLE)	Move up table elements for both main table and any nested tables.

Deleted Tags		
Applet (APPLET)	Horizontal Row (HR)	Object (OBJECT)
Area (AREA)	Searchable Index (ISINDEX)	Parameter (PARAM)
Base Font (BASEFONT)	Layer (LAYER)	Script (SCRIPT)
Background Sound (BGSOUND)	Link (LINK)	Text Area (TEXTAREA)
Button (BUTTON)	Map (MAP)	Strike-through Text (S)
Column (COL)	Meta (META)	Strike-through Text (STRIKE)
Column Group (COLGROUP)	Next ID (NEXTID)	Style (STYLE)
Deleted Text (DEL)	No Frames (NOFRAMES)	Title (TITLE)

Tags Replaced by Their Children		
Abbreviation (ABBR)	Emphasized Text (EM)	Preformatted (PRE)
Acronym (ACRONYM)	Fieldset (FIELDSET)	Sample (SAMP)
Address (ADDRESS)	Font (FONT)	Small Text (SMALL)
Bold Text (B)	Italic Text (I)	Strong Text (STRONG)
Bi-Directional Override (BDO)	Floating Frame (IFRAME)	Subscript (SUB)
Big Text (BIG)	Insert (INS)	Superscript (SUP)
Blinking Text (BLINK)	Keyboard (KBD)	Typewriter Text (TT)

Block Quote (BLOCKQUOTE)	Label (LABEL)	Quoting (Q)
Center (CENTER)	Legend (LEGEND)	Underline (U)
Cite (CITE)	Listing (LISTING)	Variable (VAR)
Code (CODE)	Marquee (MARQUEE)	Example (XMP)
Defining Instance (DFN)	No Script (NOSCRIPT)	
Division (DIV)	Plain Text (PLAINTEXT)	

Generic i-mode Transcoding

This section describes the transformations performed in generating a document (and DOM) for devices using the i-mode service. i-mode devices render their content in a slightly customized version of Compact HTML.

Transformed HTML Tags	
Tag Name	Transformation Description
Anchor (A)	Remove all attributes, including any event attributes, but NAME and HREF. However, if an HREF attribute has a value of "javascript" or "vbscript", the HREF attribute will be removed. "I-Mode" value for ACCESSKEY attribute is supported.
Body (BODY)	If converting to I-mode version 2.0, preserve BGCOLOR, TEXT, and LINK attributes and remove all others. Otherwise, remove all attributes.
Base (BASE)	Remove all attributes except HREF.
Blink (BLINK)	If converting to I-mode version 2.0, do not change. Otherwise, replace with children.
Break (BR)	Remove all attributes except CLEAR.
Caption (CAPTION)	Handled as part of converting the associated table into an unordered list. Refer to the TABLE element.
Center (CENTER)	Do not modify.
Directory List (DD)	Do not modify.
Directory List (DIR)	Replace with UL element.
Directory List (DL, DT)	Remove all attributes.
Division (DIV)	Remove all attributes except ALIGN.
Font Style (FONT)	If converting to I-mode version 2.0, preserve COLOR attribute and remove all others. Otherwise, remove all attributes.
Form (FORM)	Remove all attributes but ACTION, METHOD, and ENCTYPE attributes.
Frame (FRAME)	Replace with a link to the target to which the FRAME was pointing.
Frameset (FRAMESET)	Replace with a link to the target to which the FRAMESET was pointing.
Head (HEAD)	Remove all attributes.
Heading (H1...H6)	Remove all attributes except ALIGN.

Horizontal Row (HR)	Remove all attributes except ALIGN, SIZE, WIDTH, and NOSHADE.
HTML (HTML)	Remove all attributes except VERSION and specify a value of "C-HTML 1.0".
Image (IMG)	Remove all attributes except those allowed by the Compact HTML specification. Currently the USEMAP and ISMAP attributes are removed.
Input (INPUT)	Remove occurrences of TYPE="image" and TYPE="file". If converting to I-mode version 2.0, allow ISTYLE attribute for TYPE="text". ACCESSKEY attribute is supported for use with I-mode
List (LI)	If converting to I-mode version 2.0, remove all attributes but TYPE and VALUE. Otherwise, remove all attributes.
List (OL)	If converting to I-mode version 2.0, remove all attributes but TYPE and START. Otherwise, remove all attributes.
List (UL)	Remove all attributes.
Marquee (MARQUEE)	If converting to I-mode version 2.0, allow only BEHAVIOR, DIRECTION, LOOP, HEIGHT, WIDTH, SCROLLAMOUNT, and SCROLLDELAY attributes. If converting to I-mode version 1.0, replace with its children..
Menu (MENU)	Remove all attributes except COMPACT.
Meta (META)	If converting to I-mode version 2.0, pass element through only if HTTP-EQUIV="Content-Type" and CONTENT includes "SHIFT_JIS". Allow only these two attributes. If converting to I-mode version 1.0, delete element and its children.
No Script (NOSCRIPT)	Use in replacing SCRIPT element.
Option (OPTION)	Replace text with value of VALUE attribute, if specified.
Paragraph (P)	Remove all attributes but ALIGN.
Selection Menu (SELECT)	Remove all attributes but NAME, SIZE, and MULTIPLE. If converting to I-mode version 1.0, remove MULTIPLE attribute as well.
Span (SPAN)	Replace with children and place Break (BR) element in front of SPAN.
Table (TABLE, TBODY, TR, TH, TD, TFOOT, THEAD)	Move up table elements for both main table and any nested tables.
Text Area (TEXTAREA)	Remove all attributes but NAME, ROWS, and COLS. If converting to I-mode version 2.0, ISTYLE attribute is also allowed.
Title (TITLE)	Remove all attributes.

Deleted Tags		
Applet (APPLET)	Deleted Text (DEL)	No Frames (NOFRAMES)
Area (AREA)	Searchable Index (ISINDEX)	Object (OBJECT)
Base Font (BASEFONT)	Label (LABEL)	Parameter (PARAM)
Background Sound (BGSOUND)	Layer (LAYER)	Script (SCRIPT)
Button (BUTTON)	Link (LINK)	Style (STYLE)

Column (COL)	Map (MAP)	Title (TITLE)
Column Group (COLGROUP)	Next ID (NEXTID)	

Tags Replaced by Their Children		
Abbreviation (ABBR)	Emphasized Text (EM)	Strike-through Text (S)
Acronym (ACRONYM)	Fieldset (FIELDSET)	Sample (SAMP)
Address (ADDRESS)	Italic Text (I)	Small Text (SMALL)
Bold text (B)	Floating Frame (IFRAME)	Strike-through Text (STRIKE)
Bi-directional Override (BDO)	Insert (INS)	Strong Text (STRONG)
Big Text (BIG)	Keyboard (KBD)	Subscript (SUB)
Block Quote (BLOCKQUOTE)	No Script (NOSCRIPT)	Superscript (SUP)
Center (CENTER)	Option Group (OPTGROUP)	Typewriter Text (TT)
Cite (CITE)	Plain Text (PLAINTEXT)	Underline (U)
Code (CODE)	Preformatted (PRE)	Variable (VAR)
Defining Instance (DFN)	Quotation (Q)	

Using the Voice Transcoder

The VoiceXML Transcoder transcodes Web content to the format needed for VoiceXML capable browsers. This transcoder transcodes HTML input into VoiceXML that supports the VoiceXML 1.0 Standard. This output can be converted to speech by a voice browser such as the one provided with IBM WebSphere Voice Server.

Transformed HTML Tags	
Tag Name	Transformation Description
Abbreviation (ABBR)	Replace with children.
Acronym (ACRONYM)	Replace with children.
Address (ADDRESS)	Replace with children.
Anchor (A)	Store the HREF attribute and link text. Replace with Children. ALL HREF attributes with javascript or vbscript are removed. All event attributes such as ONCLICK or ONBLOCK are removed.
Applet (APPLET)	Remove with children.
Area (AREA)	Remove all attributes.
Body (BODY)	Remove and replace with FORM, with a BLOCK child node.
Bold (B)	Replace with children.

Base (BASE)	Replace with children. If HREF is present, it will be saved.
Basefont (BASEFONT)	Remove with children.
Big (BIG)	Remove with children.
Blink (BLINK)	Replace with children.
Break (BR)	Create a new BLOCK element and move all of the BR children to children of the BLOCK. Now move all of the siblings of the BR to be children of the BLOCK. Replace the BR with the BLOCK.
Button (BUTTON)	Remove with children.
Caption (CAPTION)	Handled as a part of making a table into an unordered list.
Center (CENTER)	Replace with children.
Cite (CITE)	Replace with children.
Code (CODE)	Replace with children.
Column (COL)	Replace with children.
Columns (COLGROUP)	Replace with children.
Delete (DEL)	Remove, with children.
Define (DFN)	Replace with children.
Directory List (DD)	Replace with children.
Directory List (DIR)	Replace with children.
Directory List (DL)	Create a new BLOCK element and move all of the DL children to children of the BLOCK. Now move all of the siblings of the DL to be children of the BLOCK. Replace the DL with the BLOCK.
Directory List (DT)	Create a new BLOCK element and move all of the DT children to children of the BLOCK. Now move all of the siblings of the DT to be children of the BLOCK. Replace the DT with the BLOCK.
Division (DIV)	Replace with children.
Emphasis (EM)	Replace with children.
Fieldset (FIELDSET)	Replace with children.
Font Style (FONT)	Replace with children.
Form (FORM)	Replace with FIELD, and assign ACTION and METHOD appropriately. Then remove the children of this node except for TEXT, INPUT, SELECT, and OPTION nodes. New FILLED and SUBMIT nodes are created to handle the creation of the VoiceXML representation of the HTML form. Because of the limitations with voice recognition in general, text boxes and checkboxes will not work correctly. Recommendations for using annotation or slightly reauthoring the pages will solve some of these problems.
Frame (FRAME)	Replace with a link to the target to which the FRAME was pointing.
Frameset (FRAMESET)	Remove from the page structure.
Head (HEAD)	Remove all attributes.
Heading (H1...H6)	Using the H1....6 tags as a guide to page construction and navigation, we will make a list of the headings to be read as choices of sections of the page that can be visited. The Hn tag will then be replaced by its children.

Horizontal Row (HR)	Remove all attributes.
HTML (HTML)	Replace with a VXML tag.
Keyboard (KBD)	Replace with children.
Italics (I)	Replace with children.
Image (IMG)	Remove all attributes.
Input (INPUT)	Remove TYPE=IMAGE, TYPE=FILE, TYPE=CHECKBOX, TYPE=SUBMIT, TYPE=BUTTON, and TYPE=TEXTBOX unannotated instances. Replace with the option for use with the field structure.
Insert (INS)	Replace with children.
Index Document (ISINDEX)	Remove all attributes.
Label (LABEL)	Remove including children.
List (LI)	Create a new BLOCK element and move all of the LI children to children of the BLOCK. Now move all the siblings of the LI to be children of the BLOCK. Replace the LI with the BLOCK.
List (OL)	Create a new BLOCK element and move all of the OL children to children of the BLOCK. Now move all the siblings of the OL to be children of the BLOCK. Replace the OL with the BLOCK.
List (UL)	Create a new BLOCK element and move all of the UL children to children of the BLOCK. Now move all the siblings of the UL to be children of the BLOCK. Replace the UL with the BLOCK.
Link (LINK)	Remove all attributes.
Listing (LISTING)	Replace with children.
Map (MAP)	Remove with children.
Marquee (MARQUEE)	Replace with children.
Menu (MENU)	Replace with children.
Meta (META)	Remove all attributes.
Next ID (NEXTID)	Remove all attributes.
No Frames (NOFRAMES)	Remove all attributes, with links generated to frames.
No Script (NOSCRIPT)	Replace with children.
Object (OBJECT)	Removeall attributes.
Option Groups (OPTGROUP)	Replace with children.
Option (OPTION)	Change the format of the OPTION tag to the VoiceXML format.
Paragraph (P)	Create a new BLOCK element and move all of the P children to children of the BLOCK. Now move all of the siblings of the P to be children of the BLOCK. Replace the P with the BLOCK.
Plaintext (PLAINTEXT)	Replace with children.
Pre-formatted Text (PRE)	Replace with children.
Parameter (Param)	Remove all attributes.

Quotation (Q)	Replace with children.
Strikethrough (S)	Remove including children.
Sample (SAMP)	Replace with children.
Script (SCRIPT)	Remove with children.
Render text smaller (SMALL)	Replace with children.
Span (SPAN)	Remove with children.
Strikethrough (STRIKE)	Replace with children.
Strong (STRONG)	Replace with children.
Stylesheet rules (STYLE)	Remove, including children.
Subscript (SUB)	Replace with children.
Superscript (SUP)	Replace with children.
Table (TABLE)	Replace with children. The contents of the table will be handled by the Table Mutator and corresponding tag mutator.
Cell (TD)	Handled as a part of the Table Mutator.
Header Cell (TH)	Handled as a part of the Table Mutator.
Text Area (TEXTAREA)	Replace with children.
Table Body (TBODY)	Removed by the Table Mutator.
Header Cell (TH)	Handled as a part of the Table Mutator.
(TR)	Handled as a part of the Table Mutator.
(TT)	Replace with children.
Table Footer Cell (TFOOT)	Handled as a part of the Table Mutator.
Title (TITLE)	Remove with children.
Underline (U)	Replace with children.
Variable Name (VAR)	Replace with children.
Example (XMP)	Replace with children.

Deleted Tags		
Big Text (BIG)	Background Sound (BGSOUND)	Next ID (NEXTID)
Bi-Directional Override (BDO)	Block Quote (BLOCKQUOTE)	No Frames (NOFRAMES)
Base Font (BASEFONT)	Floating Frame (IFRAME)	

Tags Replaced by Their Children		
Abbreviation (ABBR)	Emphasized Text (EM)	Strike-through Text (S)

Acronym (ACRONYM)	Fieldset (FIELDSET)	Sample (SAMP)
Address (ADDRESS)	Font Style (FONT)	Small Text (SMALL)
Bold text (B)	Italic Text (I)	Strike-through Text (STRIKE)
Center (CENTER)	Insert (INS)	Strong Text (STRONG)
Column (COL)	Keyboard (KBD)	Subscript (SUB)
Cite (CITE)	Menu (MENU)	Superscript (SUP)
Code (CODE)	Marquee (MARQUEE)	Text area (TEXTAREA)
Column Group (COLGROUP)	No Script (NOSCRIPT)	Typewriter Text (TT)
Directory List (DD, DIR)	Option Group (OPTGROUP)	Underline (U)
Division (DIV)	Plain Text (PLAINTEXT)	Variable (VAR)
Defining Instance (DFN)	Preformatted (PRE)	Example (XMP)
	Quotation (Q)	

Using the Palm Transcoder

The Palm Transcoder is a plugin for WebSphere Transcoding Publisher (WTP) that transcodes Web content to the format needed for Mobile Internet Kit-equipped Palm devices. This section describes the transformations performed in generating a document (and DOM) for devices using the Palm.Net service. Palm.Net devices render their content in a slightly customized version of Compact HTML. This Compact HTML is a compact form of HTML which is not the regular form of HTML, and is not the same as the compact HTML that i-mode services use.

Transformed HTML Tags	
Tag Name	Transformation Description
Address (ADDRESS)	Do not modify.
Anchor (A)	Remove all attributes, including any event attributes, but NAME, TITLE, and HREF. However, if an HREF attribute has a value of "javascript" or "vbscript", the HREF attribute are removed.
Body (BODY)	Remove all attributes except BGCOLOR and TEXT.
Base (BASE)	Remove all attributes except HREF.
Blink (BLINK)	Replace with children.
Break (BR)	Remove all attributes except CLEAR.
Caption (CAPTION)	Remove all attributes except ALIGN. Handled as a part of making a table into an unordered list.
Center (CENTER)	Do not modify.
Cite (CITE)	Do not modify.
Code (CODE)	Do not modify.

Column (COL)	Remove, including children.
Columns (COLGROUP)	Remove, including children.
Directory List (DD)	Remove all attributes.
Directory List (DIR)	Remove all attributes, convert to unordered list (UL).
Directory List (DL, DT)	Remove all attributes.
Division (DIV)	Remove all attributes except ALIGN.
Emphasis (EM)	Do not modify.
Fieldset (FIELDSET)	Replace with children.
Font Style (FONT)	Preserve COLOR and SIZE attributes and remove all others.
Form (FORM)	Remove all attributes but ACTION, METHOD, and ENCTYPE attributes.
Frame (FRAME)	Replace with a link to the target to which the FRAME was pointing.
Frameset (FRAMESET)	When used in conjunction with FRAME, the FRAME and FRAMESET combinations are replaced with a page containing links to the frames.
Head (HEAD)	Remove all attributes. Insert Palm-specific META elements.
Heading (H1...H6)	Remove all attributes except ALIGN.
Horizontal Row (HR)	Remove all attributes except ALIGN, SIZE, and WIDTH.
HTML (HTML)	Remove all attributes.
Keyboard(KBD)	Do not modify.
Italics (I)	Do not modify.
Inline Frame (IFRAME)	Replace with children.
Image (IMG)	Remove all attributes except SRC, ALT, HEIGHT, WIDTH, and ALIGN. HEIGHT and WIDTH are adjusted according to SCALE.
Input (INPUT)	Remove all attributes except TYPE, NAME, VALUE, CHECKED, SIZE, MAXLENGTH, SRC, and ALIGN.
Label (LABEL)	Replace with children.
Layer (LAYER)	Remove including children.
Legend (LEGEND)	Replace with children and add BRs.
List (LI)	Remove all attributes but TYPE and VALUE.
List (OL)	Remove all attributes but TYPE and START.
List (UL)	Remove all attributes but TYPE and START.
Marquee (MARQUEE)	Replace with children.
Menu (MENU)	Remove all attributes.
Meta (META)	Remove all attributes but NAME, CONTENT, HTTP-EQUIV.
No Script (NOSCRIPT)	Use in replacing SCRIPT element.
Object (OBJECT)	Remove including children.
Option (OPTION)	Remove all attributes but SELECTED and VALUE.

Paragraph (P)	Remove all attributes but ALIGN.
Plaintext (PLAINTEXT)	Do not modify.
Pre-formatted Text (PRE)	Do not modify.
Parameter (Param)	Remove including children.
Quotation (Q)	Replace with children.
Strikethrough (S)	Do not modify
Sample (SAMP)	Do not modify.
Script (SCRIPT)	Remove, including children.
Render text smaller (SMALL)	Do not modify.
Span (SPAN)	Replace with children and insert a break in front.
Strikethrough (STRIKE)	Do not modify.
Strong (STRONG)	Do not modify.
Stylesheet rules (STYLE)	Remove, including children.
Subscript (SUB)	Replace with children.
Superscript (SUP)	Replace with children.
Selection Menu (SELECT)	Remove all attributes but NAME, SIZE, and MULTIPLE.
Span (SPAN)	Replace with children and place Break (BR) element in front of SPAN.
Table (TABLE)	Remove all attributes except ALIGN, WIDTH, BORDER, CELLPADDING, and CELLSPACING. Scale width to 153 pixels, maximum. Convert to list, if preferred. Convert to list if the table is wider than 153 pixels. Convert to list if the table is nested.
Cell (TD)	Remove all attributes except ALIGN, ROWSPAN, COLSPAN, WIDTH, and HEIGHT.
Header Cell (TH)	Remove all attributes except ALIGN, ROWSPAN, WIDTH, COLSPAN, and HEIGHT.
Text Area (TEXTAREA)	Remove all attributes but NAME, ROWS, and COLS.
Title (TITLE)	Remove all attributes.

Deleted Tags		
Applet (APPLET)	Searchable Index (ISINDEX)	Parameter (PARAM)
Area (AREA)	Layer (LAYER)	Script (SCRIPT)
Background Sound (BGSOUND)	Link (LINK)	Style (STYLE)
Button (BUTTON)	Map (MAP)	
Column (COL)	Next ID (NEXTID)	
Column Group (COLGROUP)	No Frames (NOFRAMES)	
Deleted Text (DEL)	Object (OBJECT)	

Tags Replaced by Their Children		
Abbreviation (ABBR)	Floating Frame (IFRAME)	Quotation (Q)
Acronym (ACRONYM)	Insert (INS)	Subscript (SUB)
Bi-Directional Override (BDO)	No Script (NOSCRIPT)	Superscript (SUP)
Fieldset (FIELDSET)	Option Group (OPTGROUP)	Any unrecognized tag

Resources

There are a number of resources available that can help you make the most of Transcoding Publisher's capabilities, whether you are developing custom transcoders or creating your own XSL stylesheets.

Intermediaries and WBI	
Web Intermediaries (WBI) site	Web site for Web Intermediaries technology, an architecture and framework for creating intermediary applications on the Web.
WBI Development Kit	A flexible API for programming intermediary applications on the Web.
Intermediaries: An approach to manipulating information streams	An article in the IBM Systems Journal describing intermediaries and the WBI framework.

Java Servlets and JavaServer Pages	
Sun's Java Servlet page	Web site for Java Servlet technology.
Java Servlet API Specification (Version 2.1)	Specification and documentation for Version 2.1 of the Java Servlet API.
Servlet technical resources page	Articles and tutorials on a number of servlet-related topics, as well as a list of third-party resources such as books and other Web sites.
JavaServer Pages	Web site for JavaServer Pages (JSP) technology.
developerWorks Java technology zone	IBM's developer portal, with a variety of resources, downloads, and news items about Java and servlets.

XML and Stylesheets	
Extensible Markup Language (XML)	W3C Web site for XML, the universal format for structured documents and data on the Web.
Extensible Stylesheet Language (XSL)	W3C Web site for XSL, a language for specifying stylesheets for use in transforming XML documents.
Xerces-Java XML parser	A validating XML parser written in Java and available from the Apache Software Foundation.
Xalan-Java stylesheet processor	An XSL processor available from the Apache Software Foundation.
IBM alphaWorks	IBM's emerging technology Web site, providing a number of tools for XML and XSL development.

developerWorks XML zone	IBM's developer portal, with a articles, news, and downloads related to XML, XSL, and related topics.

Edition notice

Third Edition (August 2001)

This edition applies to Version 4 of IBM WebSphere Transcoding Publisher and to all subsequent releases and modifications until otherwise indicated in new editions.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT,

MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
IBM Corporation
Department TL3B/062
P.O. Box 12195
Research Triangle Park, NC 27709-2195
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

IBM, SecureWay, WebSphere, Everyplace, and WorkPad are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Pentium is a registered trademark of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.