SUN BLUEPRINTS
[illegible]
APPROVED

Sun™ ONE Portal Server 3.0 Rewriter Configuration and Management Guide

Rob Baker, Sun ONE Portal Server

Sun BluePrints™ OnLine—July 2002

`http://www.sun.com/blueprints`

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
650 960-1300

Part No. 816-7206-10
Revision 1.0, 06/04/02
Edition: July 2002

Sun ONE Portal Server 3.0 Rewriter Configuration and Management Guide

How can a network administrator provide secure remote access to portal users who need to download and interact with web documents and web applications that are behind a strict firewall? There are three widely used technologies for providing remote users access and interaction with web documents residing behind a strict firewall:

- Virtual private networks (VPNs)
- Secure reverse proxies
- URL rewriting

The first and most inflexible solution is to use a VPN. VPNs have two major advantages over the other two solutions in that no internal content needs to be modified, nor does the solution administrator need any inherent knowledge about the contents of the web documents for the documents to be viewable through the VPN connection. One major disadvantage, leading to the inflexibility of a VPN connection, is that all network traffic must be directed through the VPN to ensure its complete security. This can result in bottlenecks that could be avoided. Additionally, VPNs typically require client-side software to be installed. This is not feasible in the case of a nomadic user who needs to access privileged data through an Internet kiosk or similar means of connecting to the secure network.

The second solution is to use a secure reverse proxy. Like a VPN, the reverse proxy does not require internal content to be modified. The major disadvantage of using a reverse proxy is that every URL used to retrieve a document or access a web application must have explicit mappings that reside on the proxy. This means that embedded URLs must also have a URL mapping for the proxy to work successfully.

The third solution, and the primary focus of this document, is URL rewriting. The basic premise of URL rewriting is that browser requests always come back to a single location (gateway) when the request is for internal content, and the request goes directly to the public content server. Otherwise, sometimes referred to as *VPN-on-demand*, URL rewriting does not unnecessarily put stress on the network. It provides the needed security only when accessing potentially sensitive internal web sites or downloading sensitive information.

Intended Audience

This guide builds on, and in some cases reiterates, what is presented in Chapter 8 of the *Sun™ ONE Portal Server Administration Guide*. Specifically, it addresses real-world deployment scenarios and rewrite-by-example conventions.

This guide is targeted at Sun™ ONE Portal Server administrators. You are expected to be somewhat familiar with Portal Server terminology and have an extensive understanding of web application development and deployment.

You are also expected to be familiar with HTML SPEC. 4.0 tag syntax, JavaScript™ conventions, and the client-server relationship. Knowledge of HTTP and OOP programming is helpful as well. Programming examples are provided; however, the audience is expected to know what the examples actually do. In some cases, code snippets are provided. The audience must be able to understand how that code snippet relates to the larger context that the example is meant to illustrate.

Note – All iPlanet™ products have been rebranded as Sun ONE products.

Deployment Assumptions

There are several deployment assumptions made throughout this guide. The first is that the gateway and Portal Server components have been installed on physically different machines. While not a requirement, the gateway typically sits in the DMZ (demilitarized zone) with a firewall on either side, while the Portal Server Platform and Profile server sit in the corporate Intranet. The machine names referring to the respective components are `ips-server` and `ips-gateway` to avoid any ambiguity. Typically, the gateway also has the HTTP proxy enabled, but this guide makes no such distinction or assumption as to the presence and/or use of the proxy.

A variety of release streams are discussed throughout this document. This guide concentrates mostly on post Service Pack 3.0 (SP3) releases. Differences between SP3 and previous SPs are also briefly discussed where appropriate.

The last assumption is that the gateway is running with encryption enabled. As a result, all of the rewritten examples will have the `https://` protocol identifier preceding the gateway name.

How the Gateway Works

The mainstay of the Sun ONE Portal Server gateway component is the ability to present content from backend web servers and application servers through a single interface to a remote end user in a secure fashion. This can be done in one of two ways using the gateway component.

The first is by using a netlet connection and tunneling the data to the client. The netlet is usually used for tunneling fixed-port TCP/IP protocols (such as telnet and IMAP) to specialized applications running on the remote client. The second is by redirecting content requests through the gateway by using the bookmark provider on the Portal Server desktop or by specifying the URL to the internal resource with the gateway URL prepended in the location field of the web browser. The second method is used primarily for securely accessing Intranet web content.

Gateway Details

To present Intranet content to a remote user, the gateway must first know what URLs are contained in the content itself. For HTML, this is a relatively easy task because the gateway knows what HTML tags and tag attributes represent URLs. One example would be that of the *Anchor* tag with an `HREF` attribute. If the `HREF` value is an absolute URL, the gateway prepends its own URL to the beginning of the original URL so that if an end user selects that particular link, the request would go directly to the gateway component where it would then be retrieved by the gateway on behalf of the browser.

The following is an example of a URL:

```
<A HREF="http://www.internal.iplanet.com">
```

The above URL would become the following:

```
<A HREF="https://ips-gateway.iplanet.com/http://www.internal.iplanet.com">
```

This result is known as URL rewriting or URL translation. Using the same example, if the HREF value was relative to the server root, the gateway would first resolve the server URL based on the HTTP header information. It would then concatenate the the gateway URL and the absolute Intranet URL to make the final result.

The following is an example of a relative URL:

```
<A HREF="/pages/page2.html">
```

After passing through the gateway, the above would become the following:

```
<A HREF="https://ips-gateway.iplanet.com/http://www.internal.iplanet.com/pages
/page2.html">
```

If no prepended path information is given for a URL, the gateway does one of the following to expand the relative path into an absolute URL:

- It uses a BASE tag if one exists in the document.
- It attempts to resolve the absolute URL by using the host and path information from the HTTP request header.
- It does nothing, and the browser uses the URL in the location field as the URL to resolve the relative path when the page is rendered.

The third possibility is described in more detail in "Rewriter Verses Browser" on page 6 and applies more to pre-SP3 deployments. In addition, Portal Server releases earlier than SP3 (with the exception of SP2 Hot Patch 4) ignored the BASE tag altogether. This behavior also assumes that the HREF rule is still listed in the `Rewrite HTML Attributes` field of the gateway profile (which is there by default, out-of-box). How to add, remove, and view rules in the gateway profile is discussed in "Adding and Removing Rewriter Rules" on page 8.

With this basic model in mind, there needs to be some way to tell the gateway that a particular string or piece of code actually represents a URL. The purpose of the rewriter is to provide a human computer interface for providing context where the gateway sees only syntax. By adding rules to different sections of the gateway profile, the way in which the gateway parses, interprets, and modifies the result-set is changed.

Rule entries are simply a list of substring matches or regular expressions that the gateway uses to determine if a string, or portion of scripted code, needs to be rewritten. The rule entries are stored in LDAP and are part of the gateway profile. In SP3, the gateway can rewrite most of the HTML 4.0 tags and tag attributes. The exception is the STYLE attribute that can contain a background URL parameter. The JavaScript code is dealt with using a variety of methods described in "Rewriting JavaScript Content" on page 39. As of SP3, Form Input Data and Java Applet parameters can be rewritten with the appropriate gateway profile configuration. SP3 Hot Patch 1 adds the ability to rewrite XML data, inlined CSS, imported CSS, and imported JavaScript code (to some extent). SP3 Hot Patch 3 includes case insensitive rewriting of the background-URL CSS function within a STYLE tag. This release also adds the ability to use wildcards with the JavaScript content, which is particularly useful for JavaScript document object arrays.

Rewriter Verses Netlet

As stated earlier, the Netlet is typically used to provide secure remote access to specialized, fixed-port TCP/IP applications that talk to their own client application residing on the end-user machine. The netlet does this by establishing a secure tunnel between the client and server, using a preconfigured local port that communicates directly with the gateway. Telnet, Citrix, and IMAP are just a few of the programs and protocols that leverage Netlet functionality.

The netlet is not used for web surfing because of the difficulty in configuring it to actually work in that manner. The browser would have to be configured so that all HTTP requests are redirected to the local host port on which the Netlet is listening. There would also have to be a web proxy configured on the corporate side that knows how to handle the incoming netlet HTTP requests. Those requests might be for content outside of the corporate Intranet and have to be handed off to another proxy. While this extends the Portal Server to include more VPN-like functionality, it would be difficult to implement, put undue strain on the gateway, and require client customizations that may not be feasible, depending on the client type, the end-user location, and the intended Portal Server audience.

Take, for example, a business-to-business portal that provides a parts ordering interface. The company providing the parts interface would not be able to dictate web browser configuration requirements to the parts ordering company. The netlet would be used in this case if the parts ordering interface was a TCP/IP application that had a separate client application to interact with it, rather than a Web-based interface.

In contrast to the netlet, the rewriter allows remote access through the use of a Netscape Navigator™ 4.x or Internet Explorer 5.x browser. The rewriter uses similar functionality of a full-featured reverse web proxy, with the added benefit of rewriting the URLs so that there is no browser configuration required to make sure requests for Intranet content are routed back to the gateway. This prevents the browser from trying to make direct requests to content that is not available outside of the corporate firewall.

Concepts of the Rewriter

The flexibility of the netlet and the uniqueness of the rewriter is what differentiates the Sun ONE Portal Server from most, if not all, other Portal Server offerings. Understanding how the rewriter works and what necessitates rule configuration is essential to a successful and secure Portal Server deployment. Experimenting with the rewriter before the Portal Server is moved into production will also reduce last minute problems that could be avoided.

Rewriter Verses Browser

Because the rewriter is one of the most highly used and sought after features in the Portal Server, it is constantly undergoing modifications and enhancements. Many Portal Server administrators found out that this content that worked in the SP2, no longer worked in SP3, and newer, versions. One major reason for this difference in behavior was the administrator's reliance (knowingly or not) on the browser to handle the resolution of relative URLs by using the location field as a `BASE` tag equivalent when the page was rendered.

This was more evident where the gateway component had been deployed as employee-facing secure remote access to an existing Intranet portal. In this particular case, an employee would log in to the gateway and be redirected, not to the Portal Server Desktop, but to their own home page or corporate Intranet portal. The location field in the URL would still have the gateway address prepended to it, and the Portal Server session would remain active, as long as requests continued to be made through the gateway. Relative URLs in the redirected page would be resolved by the location field as long as there was not a `BASE` tag present in the document.

Where this browser relative path resolution took place actually represented inadequacies in the rewriter itself. It meant that there was content that the rewriter was missing or not interpreting correctly. This was usually acceptable in cases where the browser would be able to help out, but prior to SP3, there were large gaps in rewritten content, such as in the case of imported JavaScript or imported CSS. For instance, in SP2, the `SRC` attribute in the `SCRIPT` tag would be rewritten so that the browser would be able to correctly retrieve the JavaScript content, but the JavaScript content itself was not rewritten according to the rules specified in the gateway profile. While some people were seeing favorable results where the browser would handle relative URLs, other people had problems where variables in imported JavaScript content were not being rewritten correctly.

Administrators who deployed the gateway in front of the Sun ONE Portal Server desktop in SP2 found that in SP3, with the new rewriter functionalities, the browser location field no longer could be used for resolution of relative URLs. The reason is that the URLs on the Portal Server desktop are derived from scraped channels or custom providers and cannot be resolved using the document location that contains the gateway URL, server URL, and `DesktopServlet` in the path.

For instance, consider the following URL:

```
https://ips-gateway/http://ips-server:8080/DesktopServlet
```

This URL would not work for a relative URL of `../` for a page scraped from an internal site, other than `ips-server` because the incorrect server would be referenced by the relative path resolution of the browser.

Relative URLs, which are not rewritten to absolute URLs, should be avoided for the following reasons:

- The Portal Server wants to be sure that requests to get internal content will always come back to the gateway component.
- Absolute URLs are used to determine if the URLs need to be rewritten based on their domain and/or subdomain value.
- Absolute URLs are compared to the profile of the user to see whether or not they have permission to retrieve the specified content.
- Absolute URLs avoid situations where the browser may resolve a relative path to the incorrect fully qualified path.

Gateway Profile

The gateway profile is a component stored in LDAP with attributes and attribute values used by the gateway for initialization and runtime purposes. Contained in the gateway profile are attributes that are used specifically by the rewriter to determine what content, if any, should be rewritten. To look at the contents of the gateway profile, you can use the `ipsadmin` command to dump the component into an XML file to view it. Otherwise, you can access the Administration Console, select Gateway Management and Manage Gateway Profile to view the gateway profile. Both of these methods are explained in detail in “Adding and Removing Rewriter Rules” on page 8.

While the gateway profile contains entries for a variety of aspects related to the gateway operation, only those fields and/or attributes directly related to the rewriter will be discussed. Two settings to be aware of that control the overall behavior of the rewriter are the `Rewrite All URLs Enabled` checkbox and the `DNS Domains and SubDomains` list. These settings are not mutually exclusive as the checkbox will override any entries for Domains and SubDomains if checked. The remainder of the document assumes that `Rewrite All URLs Enabled` is checked.

It is worth noting that if `Rewrite All URLs Enabled` is not checked, any content you wish to have rewritten must have its server domain and subdomain (if one exists) entered in the DNS Domains and SubDomains list. If there is not a subdomain associated with the domain, be sure to put a vertical bar after the domain name. For instance, if you wanted to rewrite every URL that contains `iplanet.com`, you would add an `iplanet.com|` entry to the DNS Domains and SubDomains list. If you want to rewrite only certain subdomains within that domain, you would add an `iplanet.com|`*internal* entry (*internal*, in this case, signifies a fictitious subdomain name).

Rule Interpretation by the Gateway

Rules are a means of informing the gateway how to determine if content that passes through it contains a URL that needs to be rewritten. The rules are either strings, or, in some cases, strings containing wildcards, that are used as regular expressions. Each rewriter attribute and/or field in the gateway profile has an associated environment where the rule will apply, and a top down order in which rules will be compared one by one to content within that document or document section to see if there are any URLs which require translation.

One way an environment can be determined is by an HTML tag such as the `SCRIPT` tag for JavaScript code or the `STYLE` tag in the case of CSS. Within each environment, there are different data constructs that may require differing syntactical interpretation by the gateway. The JavaScript language, for instance, contains functions that can take function parameters. If one of these function parameters happened to be a URL, a rule would have to be added to the gateway Profile under the Rewrite JavaScript Function Parameters section that would determine the function name and parameter that requires rewriting.

Environments can also be determined by MIME types so that when the content is retrieved by the gateway, it is compared to the appropriate subset of rules and rule values. Imported JavaScript code would contain a MIME type of application/x-javascript extracted from the browser `GET` request so that when the gateway retrieves the content, it knows what the environment used to rewrite it.

Adding and Removing Rewriter Rules

This section describes how to add, remove, and view rewriter rules through both the administration interface and the command line interface. Maintaining the rewriter rule set and understanding how the gateway interprets content based on the values contained within the different rewriter attributes in the case of the CLI and fields in the case of the administration console are essential to successfully deploying the gateway in front of complex web applications.

▼ To Access and Modify the Gateway Profile Using the Administrative GUI

1. **Go to** `http://ips-server.iplanet.com:8080/console`**, and log in as the administrator.**

   This is the superuser UNIX™ login by default. Be sure to specify the port number, hostname, and domain relative to your own Portal Server installation.

2. **Select the Gateway Management Link under the Portal Server Services Heading.**

3. **Select the Manage Gateway Profile link.**

   After this page comes up, there will be a long list of component attributes. For details about attributes not related directly to the rewriter, refer to the *Sun ONE Portal Server Administration Guide*.

   All gateway component attributes directly related to the rewriter contain Rewrite as the first word in the field name. Rules can be added by typing the rule information in the text field just below the option list of the field to be changed and selecting the Add button. The new value is not added to the gateway profile until the Submit button at the bottom of the page has been clicked.

   To delete a rule, first select it, then select the Delete button just bellow the text input field. Again, the changes are not actually made to the profile until the Submit button has been clicked.

   Do not mix and match additions and deletions prior to a submittal so that you are sure that the profile is updated correctly. Also, you should verify that your changes have been made by going back to the gateway profile page.

▼ To Access and Modify the Gateway Profile Using the Command Line Interface

The gateway profile can be modified and viewed by importing and exporting the `iwtGateway` component using the `ipsadmin` command.

1. **Dump the component for viewing and editing to an XML file by typing the following commands:**

```
root@ips-server: PATH=$PATH:/opt/SUNWips/bin;export PATH
root@ips-server: ipsadmin get component iwtGateway > /tmp/iwtGateway.xml
```

where `/opt` is the Portal Server install directory.

The rewriter-specific gateway attribute fields that correspond to what is seen on the administration GUI can be seen by typing the following command:

```
root@ips-server: grep "desc=\"Rewrite" /tmp/iwtGateway.xml
   desc="Rewrite JavaScript Function Parameters In JavaScript"
   desc="Rewrite JavaScript Variables Function"
   desc="Rewrite Form Input Tags List"
   desc="Rewrite JavaScript Function Parameters In HTML"
   desc="Rewrite JavaScript Function Parameters"
   desc="Rewrite JavaScript Function Parameters Function"
   desc="Rewrite HTML Attributes"
   desc="Rewrite Applet/Object Parameter Values List"
   desc="Rewrite JavaScript System Variables Function"
   desc="Rewrite JavaScript Variables In JavaScript"
   desc="Rewrite HTML Attributes Containing JavaScript"
   desc="Rewrite Attribute value of XML document"
   desc="Rewrite JavaScript Variables In URLs"
   desc="Rewrite Text data of XML document"
   desc="Rewrite JavaScript Variables In HTML"
   desc="Rewrite All URLs Enabled"
```

Each of these entries is the description field for the actual attribute name, which might be something like `iwtGateway-JavaScriptVariables`.

2. **To add a value to an attribute, copy everything between the attribute tags including the tags themselves into a new XML file.**

```
root@ips-server: /usr/bin/cat <<EOF >/tmp/iwtGateway-
JavaScriptVariables.xml
<iwt:Att name="iwtGateway-JavaScriptVariables"
   type="stringlist"
   desc="Rewrite JavaScript Variables In URLs"
   idx="a5"
   userConfigurable="false"
   >
   <Val>g_szBaseURL</Val>
   <Val>g_szVirtualRoot</Val>
   <Val>szViewClassURL</Val>
```

```
   <Val>g_szExWebDir</Val>
   <Val>g_szPublicFolderUrl</Val>
   <Val>g_szUserBase</Val>
   <Val>imgsrc</Val>
   <Val>location.href</Val>
   <Val>_fr.location</Val>
   <Val>mf.location</Val>
   <Val>parent.location</Val>
   <Val>self.location</Val>
   <Val>lnk</Val>
   <Val>tabURL</Val>
   <Val>document.location.href</Val>
   <Val>window.status</Val>
   <Val>window.location.href</Val>
   <Wperm>ADMIN</Wperm>
   <Rperm>ADMIN</Rperm>
   <Rperm>OWNER</Rperm>
  </iwt:Att>
EOF
```

3. **Edit the XML file, and add the value or values you want.**

4. **Type the** `ipsadmin` **command to update the gateway profile with the new attribute values.**

```
root@ips-server: ipsadmin change component iwtGateway
/tmp/iwtGateway-JavaScriptVariables.xml
Operation completed successfully.
```

5. **Verify that the new value was imported successfully by typing the following command:**

```
root@ips-server: ipsadmin get component iwtGateway | grep
window.location.href
<Val>window.location.href</Val>
```

Note – Multiple rules with the same name cannot be specified for the same environment and interpretation. For example, a variable called `my_URL` cannot be present in both the Rewrite JavaScript Variables in URLs section and the Rewrite JavaScript Variables Function section of the gateway profile. Only the first instance of the rule encountered by the gateway will be used.

After updating the gateway profile, it is a good idea to keep a backup of the profile data. You can do this by using the `ipsadmin` command (as in the following example) to dump the gateway profile and wrap component tags around the result. The component tags are required in the event that the entire component has to be deleted and re-imported.

```
root@ips-server: cp /etc/opt/SUNWips/xml/iwtGateway.xml
/etc/opt/SUNWips/xml/iwtGateway.xml.bak
root@ips-server: /usr/bin/cat <<EOF >/tmp/iwtGateway.xml
<iwt:Component name="iwtGateway"
   ver="1.0"
   desc="Gateway Profile"
   resB="iwtGateway"
   idx="">
EOF
root@ips-server: ipsadmin get component iwtGateway >> /tmp/
iwtGateway.xml
root@ips-server: /usr/bin/sed '$d' /tmp/iwtGateway.xml >
/etc/opt/SUNWips/xml/iwtGateway.xml
root@ips-server: /usr/bin/cat <<EOF >>/etc/opt/SUNWips/xml/
iwtGateway.xml
</iwt:Component>
EOF
```

Follow the same guidelines to remove a value from an attribute list. If the gateway profile becomes corrupted for any reason, you might be able to delete the component entirely and re-import it using one of the backed up `iwtGateway.xml` files, as in the following example.

```
root@ips-server: ipsadmin delete component iwtGateway
Operation completed successfully.
root@ips-server: ipsadmin get component iwtGateway.xml
Profile get failed. More info: Unable to get attribute or privilege value from
data store
root@ips-server: ipsadmin -chkxml /etc/opt/SUNWips/xml/iwtGateway.xml
Operation completed successfully.
root@ips-server: ipsadmin -import /etc/opt/SUNWips/xml/iwtGateway.xml
Operation completed successfully.
root@ips-server: ipsadmin get component iwtGateway > /dev/null 2>&1
root@ips-server: if [ $? -eq 0 ]; then { echo "Success"; } else { echo
"Failure"; } fi
Success
```

Note – The gateway component must be restarted after any modifications to its profile data.

Methodology for Rule Extraction

Rule extraction from Web applications that will be accessed through the gateway can require persistence. Depending on how the application is written, its integration with the gateway usually will fall in to one of three categories:

- Integration out-of-box
- Integration through profile configuration
- Integration with special attention required

Integration Out-Of-Box (Category 1)

This category usually applies to web applications that are delivered to the browser in purely HTML or HTML and some other languages inlined that do not reference any URLs. Its integration is usually straightforward and requires little or no administrative intervention. This content tends to be static in nature, contain absolute URLs, and be well-formed. Well-formed here means that the entire content is syntactically correct and basic formatting practices are in place. These pages do not have Forms, Java Applets, or imported scripts that require special attention. The `IMG SRC`, `A HREF`, `FORM ACTION`, `APPLET CODEBASE`, and `JAVASCRIPT SRC` tag attributes are just a few that are handled by the gateway out-of-box.

Integration Through Profile Configuration (Category 2)

This category applies to web applications, including those in Category 1, that are increasingly complex. Category 2 content would contain URLs in:

- `FORM INPUT` tags
- Applet parameters
- JavaScript event handlers
- A multitude of content types such as CSS, JavaScript code, and XML
- Dynamically created content

Some of this content is handled out-of-box, while other content requires special considerations.

Integration With Special Attention Required (Category 3)

This category content includes dynamically created URLs on the client-side, complex scripts that have URLs in function parameters, URLs built in several steps or in multiple locations in the code using string concatenation, URLs contained in fractured JavaScript, URLs hidden in nested function calls, integration with unknown third party applications, and URLs contained in code that has passed through an obfuscator.

Category 2 content makes up the bulk of what most people would expect to see pass through the gateway. For applications that are being created specifically for use with the gateway, there are often a multitude of content-based workarounds that can be put in place if it is difficult, or not possible, to create a rule that will match a specific URL. Look for best programming practices in later sections for information on possible workarounds for different corner cases or content types.

The important thing to keep in mind is that the only thing to worry about content-wise is where URLs are referenced. URLs that are not correctly rewritten can manifest themselves in a variety of ways. The applications may misbehave when certain buttons are selected, forms are submitted, or other actions occur, such as a `mouseOver`. The browser may return an error message saying that a particular server is either down or inaccessible when a link has been selected. Users may be mysteriously redirected back to the Portal Server login page, even though they did not log out or have their session terminated. Images may show up broken, Applets may not download completely or run correctly. Navigation bars may not work correctly. Any of these could be signs that the gateway requires additional configuration to work with the application.

For testing purposes, you should ensure that the browser cannot make a direct connection to any server through the gateway. Otherwise, when the Portal Server is moved into production. there may be a number of issues that arise because the browser is no longer able to talk directly to internal content. One way to determine if this is a problem is to snoop the connection between the client and the content server. There should be no direct communication between the two when accessing the content server through the gateway component. If there is, then it is likely that a URL has been overlooked and has not been rewritten.

What Rules Are Necessary?

There are a number of ways to go about investigating what rules may be necessary for the correct rewriting of URLs. Some understanding of the web application will help in figuring out how to rewrite parts of the content correctly.

You should answer the following questions before diving straight into the Web application source code:

Does the Document Have Frames?

If the answer is yes, then you must ensure that you start looking at the source code for the correct frame. You will also want to see if there is a `SCRIPT` tag in the parent document that initializes top-level JavaScript variables that could include URLs or hint at how URLs will be used throughout the application. Keeping this in mind, you would also want to look at the individual frames to see if they make references to `parent.*` or `top.*` that might reference URLs (like `parent.location.href`).

For example, when a specific frame-enhanced page is accessed through the gateway, the page does not render properly if it is resized, and none of the tabs can be selected at the top of the page.

The following is the parent document source code:

```
<HTML>
<FRAMESET ROWS="75,*">
<FRAME SRC="index_top.html" NAME="tabs">
<FRAMESET COLS="134,*"
onResize="options.location.href='index_left.html';
content.location.href='index_right.html';">
<FRAME SRC="index_left.html" NAME="options">
<FRAME SRC="index_right.html" NAME="content">
</FRAMESET>
</FRAMESET>
</HTML>
```

The JavaScript event handler `onResize` executes if the browser window proportions are changed. The `options.location.href` and `content.location.href` JavaScript document objects refer to the URLs of the frame names `options` and `content`. The `FRAME SRC` attributes will be rewritten automatically because all `SRC` attributes in HTML tags will be rewritten by the gateway out-of-box. Because part of the problem is that the page does not render properly when the window is resized, it could be that both `content.location.href` and `options.location.href` are not in the gateway profile under the Rewrite JavaScript in URLs section. `onResize` will also have to be listed in the Rewrite HTML Attributes Containing JavaScript list.

For details about either rewriter attribute see "Rewriting HTML Attributes" on page 28 and "Rewriting JavaScript Content" on page 39. When this page is parsed by the gateway, it will go tag by tag and compare the attribute names to those in the Rewrite HTML Attributes section of the gateway profile. If something is matched, it will attempt to translate the attribute value as a raw URL. If the HTML attribute name appears in the Rewrite HTML Attributes Containing JavaScript list, the gateway will attempt to translate the JavaScript contents so that it can be resolved into a rewritten URL. In this case, the `onResize` attribute value contains two JavaScript variable assignments that are raw URLs. After `onResize`, `content.location.href` and `options.location.href` have been added to the appropriate gateway profile sections, this entire page should be rewritten correctly.

The other half of the problem in this example has to do with the tab links not working. Because this is a frame document, the source for the top frame containing the source code definitions for the tabs will have to be consulted. In this case, that would be index_top.html:

```
<HTML>
<HEAD>
<SCRIPT>
  <!--
     function openTab(id, link) {
       parent.content.location =  link;
     }
  //-->
</SCRIPT>
</HEAD>
<BODY>
<A HREF="javascript:location.reload();" onClick="openTab(0,
  'http://www.iplanet.com');">
  <IMG SRC="images/LeftTab.gif">
  iPlanet Home
<IMG SRC="images/RightTab.gif">
</A>
</BODY>
</HTML>
```

In this particular instance, having parent.content.location in the Rewrite JavaScript Variables in URLs section will not correctly rewrite the value of link in the openTab function body. The reason for this is that link is not a raw URL, so the gateway does not know how to rewrite it. The value of link will not be defined until the Anchor link is selected in the browser and the string value http://www.iplanet.com is passed to the openTab function.

There are two ways to handle this. One is to add openTab:,y to the Rewrite JavaScript Function Parameters section of the gateway profile. This would prepend the gateway URL to the second parameter of the openTab function call within the Anchor tag. The other option is to add parent.content.location to the Rewrite JavaScript Variables Function section of the gateway profile. This will insert a function called iplanet within the SCRIPT tag and change link on the right side of the variable assignment to iplanet(link).

Starting in SP4 Hot Patch 1, the iplanet function definition occurs in the document HEAD element in its own SCRIPT element, instead of being placed multiple times throughout the document.

This will result in the link URL being rewritten by the client at runtime using the browser JavaScript engine. Because parent.content.location would have already been added to the gateway profile in the Rewrite JavaScript Variables in

URLs section to correctly rewrite the parent document, the better option might be to rewrite the `openTab` function parameter. Otherwise, `parent.content.location` could be moved from the Rewrite JavaScript Variables in URLs section to the Rewrite JavaScript Variables Function, which would change the variable assignment in the parent document to:
`content.location.href=iplanet('index_right.html');`

If the openTab function call looked like:

```
openTab(0, top.location.href)
```

then `openTab:,y` would have to be added to the Rewrite JavaScript Function Parameters Function section of the gateway profile. This option is only applicable starting in the SP4 Hot Patch 1 release. This avoids the problem where the `iplanet` function definition would be placed within the HTML tag body.

If optimization is the goal where there may be limited compute power on the client, reducing or eliminating the number of times the client has to resolve URLs using the `iplanet` function is a good idea. If flexibility is the goal, specifically where the same variable name is used in a variety of contexts, using the `iplanet` function to dynamically resolve the URLs is a good idea.

Does the Web Application Create Content Dynamically?

Special considerations may have to be made if content that is accessed through the gateway is generated dynamically by CGIs, servlets, or JavaServer Pages™ technology. Rule extraction is fundamentally the same for dynamic content as it is for static content, except that care needs to be taken in any direct manipulation of the HTTP headers. Also, the original application source code may have to be referred to, or even modified, to more easily determine where URLs might reside and how to best have the gateway handle them.

One thing to make sure with applications that create content dynamically is that the appropriate Content-Type header is set. Otherwise, the gateway may incorrectly rewrite the content, or not rewrite it at all. In a Perl application for example, the Content-Type is usually the first thing added to the response, and it generally looks something like this:

```
print "Content-Type: text/html\n\n";
```

The content-type HTTP header tells the gateway which environment to use to rewrite the content that will follow. Currently, the only content-types that are rewritten by the gateway are `text/html`, `text/htm`, `application/x-javascript`, `text/css`, `text/xml`, and `text/x-component`. These entries can be seen by selecting the Show Advanced Options button at the bottom of the gateway component profile using the administration console.

The content-types can then be seen in the MIME Type Translator Class section of the gateway profile. Adding additional content-types here will work only if the content does not contain special tagging conventions or if it is plain text. URLs outside of the tags themselves will not be rewritten, with the one exception being the first string that begins with a protocol identifier after the SPAN tag. This was done for compatibility purposes with Microsoft Exchange's Web interface.

The following is an example:

```
<SPAN>http://www.iplanet.com</SPAN>
```

There are some other things to keep in mind for integrating applications with the gateway in general. One is to not have an explicit dependence on the content-length header. It is obvious that after the gateway rewrites source code, the content-length will also be different from what the web application originally set it as. This fact can be overlooked fairly easily. A problem with how the content-length header is being manipulated might manifest itself as a page truncation.

When integrating with a JavaServer Pages application, be sure that all of the tags are resolved by the JavaServer Pages engine prior to going through the gateway. Depending on what the tag looks like, the gateway may attempt to rewrite its attributes, making a broken tag, otherwise invisible to the end-user, become visible.

The JavaScript content that passes through the gateway should be syntactically correct. If it is not, the problem can manifest itself by misplacing the iplanet function, or incorrectly parsing the SCRIPT tag and corrupting the page output. This page corruption sometimes shows two SCRIPT closing tags with no opening tag and may even move the entire SCRIPT block to a different location in the page source. Another often overlooked issue with the JavaScript application integration is the closing comment to hide the JavaScript code from non-compliant browsers. Unlike its HTML equivalent, the closing comment should have two leading slashes in front of it.

The following is an example:

```
<SCRIPT>
   <!-- Hide from non-compliant browsers

   // -->
</SCRIPT>
```

One of the difficulties with dynamically created content is that rule extraction tends to be more difficult because URLs are also created dynamically. Also, some applications will attempt to prevent the end-user from being able to view the source of the web application. This is usually done through trickery by trapping the right mouse event or through code obfuscation. Code obfuscation may make the task of rule extraction difficult, if not impossible, and should be avoided for applications that pass content through the gateway.

If, for instance, the code was not only generated dynamically, but obfuscated dynamically as well, the variable names would never be the same, and thus, reliable string matching would not be possible. Even if the obfuscation was only run once, there is a risk of local variables being obfuscated to the same name, which might have unpredictable results. If for some reason the source code cannot be viewed using the browser, the rewritten content can also be viewed by setting the `ips.debug` option in the gateway `/etc/opt/SUNWips/platform.conf` file to Message and restarting the gateway.

The `/var/opt/SUNWips/debug/iwtGateway` file will contain the document source after page translation. For a busy gateway, you may have to use `vi` or your favorite editor to search for the browser `GET` request and for the translated response, or just search for log entries beginning with: `HTMLTranslator:Begin:`

As mentioned previously, it may be easier in some cases to extract rules from the web application's source code, if it is available. The reason for this is that applications generally contain functional blocks. For example, if the end-user is experiencing problems with a navigation button accessing a Perl application through the gateway, and the Perl program contains a subroutine called `buildNavBar`, that may be a better place to start than searching the document view source or debug logs. Sometimes the opposite may be true because the browser source may be a great deal less complex than the web application source. This might be the case if you have a for loop in your web application that is creating a JavaScript for loop block that is dynamically creating image URLs to be used for `mouseOvers`. The web application might also contain variables that only have meaning within the web application and might not ever be seen by the gateway.

Automated Extraction Techniques

Extracting rules by hand is not always a straightforward experience. Attempting to automate that process may prove to be difficult as well, depending on the level of complexity of content that will be accessed through the gateway. For a sidebar on differing levels of complexity, refer to “CASE Studies: How to Configure the Gateway to Rewrite a Web-Based JavaScript Navigation Bar” on page 69 about rewriting a JavaScript web navigation bar.

An ideal companion for the gateway administrator might be an automated recommendation engine that works as a Web crawler and would mine out possible URLs that might need special consideration and make judgements as to how they might best be handled in the gateway profile. Better yet, would be to have the recommendation engine automatically add the rule to the gateway profile when it is at least 90 percent sure that the rule is not only needed but that it will also not regress any other rules or cause other content to be incorrectly handled.

Unfortunately, such a tool is not available today, so gateway administrators must rely on their own skills (and scripting abilities) to find URLs in the content. Luckily, as the document object model has continued to catch on, more and more developers are starting to manipulate document objects directly. These object references can usually be matched by a regular expression and tend to be assignments with a raw URL on the on the right side of the variable assignment.

The following is an example:

```
document.images["IMG"+imgNum].src = "../../images/img"+imgNum+".gif";
```

This example uses a predefined JavaScript array and is useful for a couple of reasons. One is that any reference to an array with a `SRC` property will likely have to be rewritten. The same is true for `HREF`. Also, `document.images["IMG"+imgNum].src` cannot be added to the gateway profile because the brackets are not understood. In SP3 Hot Patch 3 and SP4 Hot Patch 1, functionality was added to be able to use wildcards with these kinds of rules so that not only could they work for array references, but also to reduce the total number of rules that need to be added to the gateway profile. This rule optimization is particularly beneficial when gateway logging is enabled and when there are many similar rules defined.

For the example above, a rule like `document.images*.src` could be added to the Rewrite JavaScript Variables in URLs section of the gateway profile. Because it is known that there are other arrays that contain `SRC` properties that are also document arrays, the rule can be revised to `document*.src`.

However, there are also window objects in the object hierarchy, such as `window.frames`, that also have `SRC` attributes. Both the document and window objects can have this as a placeholder for the actual object name. Some objects have an `HREF` property as well, so two rules can be added that would account for a great deal of content that uses the JavaScript object hierarchy directly and/or the JavaScript predefined arrays to access the object hierarchy. These rules would be: `*.src` and `*.href`

Because these are the most generalized rules, they should occur before other rules to improve the performance of the rewriter.

The other thing to notice from the example is that the right side of the assignment begins with `../../images/`. This would be a relative URL to the images directory that contains the prepended path information.

Because there is a string literal as the first portion of right side that contains prepended path information, it is considered a raw URL—meaning that it is directly resolvable by the gateway. If the assignment instead looked like the following:

```
document.images["IMG"+imgNum].src = imgURL + imgNum + ".gif";
```

then the image `SRC` URL would not be rewritten because the gateway does not understand what `imgURL` is, and `imgURL` could also change at runtime because it is a variable. Also, as of SP3 Hot Patch 3, JavaScript wildcarding works only for rules that are added to the Rewrite JavaScript Variables in URLs section of the gateway profile. To rewrite the second case then, `imgURL` would have to be added to the gateway profile in either the Rewrite JavaScript Variables in URLs section or the Rewrite JavaScript Variables Function section, depending on its usage in other areas of the application.

The last thing this example demonstrates is that if there are no rules added to the gateway profile for this example and the page is still accessed by redirecting through the gateway, the image will still be handled correctly. This behavior was alluded to earlier when comparing the rewriter to the browser where the browser may actually resolve relative URLs using the location field as though it were actually a `BASE` tag. In fact, if you set the cache size to a nonzero value large enough to cache the page, then view the source using the Netscape Navigator browser, the `BASE` tag will be included as part of the source. This is one reason why it is also important to determine rule extraction with the browser cache set to a size of zero and to check for updated pages every time. Otherwise, you may not see any JavaScript content when you view the source because it has already been rendered once and cached by the browser.

This is especially true if the web application contains numerous JavaScript `document.write` calls. If this example were included as a URL scraped channel on the desktop or if the JavaScript portion of the example was imported using a `SCRIPT SRC` attribute, then the relative URLs would not be handled correctly. The scraped channel would not work because the `BASE` equivalent would always have a path of `DesktopServlet`, and the host would likely be incorrect as well. There is a fix available for the later in SP4 Hot Patch 1 for the Internet Explorer browser. The Netscape Navigator browser does not send a document referrer header, so it is not possible for the gateway to determine the parent document URL to use to resolve relative links in the imported JavaScript code.

The term raw URL has been referred to throughout this document without much explanation other than what can be derived from the examples given. Understanding what a raw URL is will help in determining what rule to use and in what section of the gateway profile it should reside. Raw URLs are any string that can be clearly identified as a URL. Raw URLs have relevance when rewriting HTML attributes, `FORM INPUT` tags, and `APPLET` and/or `OBJECT` parameters, but it is most useful to differentiate a raw URL in JavaScript content where a variable assignment occurs. As such, the following is a good rule of thumb for determining raw URLs in JavaScript content. A raw URL in JavaScript content must follow these conventions:

1. Is a string literal enclosed by matching single or double quotes.
2. Usually, but not always, contains prepended path information.
   - Prepended path information can be relative or absolute.
   - The prepended path information must all be in the first string literal after the variable assignment.
   - If no prepended path information is provided, the FQD + path to the parent document is used as the BASE equivalent.
3. It is not built on separate lines by using a concatenation operator.

   Examples of JavaScript variable assignments that are raw URLs:

```
var myURL = "http://www.sun.com/" + prodPath + "solaris"; -
```

The above is a fully qualified prepended path without path remainder.

```
img = "../../images/myimg.gif";
```

The above is a relative prepended path with path remainder.

```
newImg = "../../" + "images/newimg.gif";
```

The above is a relative prepended path with no path remainder.

```
URL = 'images/' + imgNum + '.gif';
```

The above has no prepended path with no path remainder.

The following are examples of JavaScript variable assignments that are not raw URLs:

```
var offImg = "../" + "../" + "images/off.gif";
```

The above is a prepended path that is split.

```
var mouseOverImg = up2dir + "images/moseover.gif";
```

In the above example, `up2dir` is a variable.

```
surfToNewPage += '?param1=val&' + param2Name + '=val2';
```

The above example contains multiple assignments using the += operator.

Typically, JavaScript variable assignments that contain raw URLs are added to the Rewrite JavaScript Variables in URLs section of the gateway profile.

One of the best ways to automate rule extraction is by doing string matching directly on the content that will be accessed through the gateway. If the content is stored locally or if the gateway logging has been set to Message and has been accessed already using the browser, you might be able to use the `grep`(1) command to find content in pages that contain URL references. This is a simple approach, but it may prove to be more powerful than you initially think.

The following is an example of how to find URL variable assignments in imported JavaScript content:

```
$ find ./htdocs -name '*.js' -exec grep '\= \"http' {} >> /tmp/
jsAssignmts.txt \;
$ cat /tmp/jsAssignmts.txt
    var url = "http://www.iplanet.com/bugsplat/show_bug.cgi?id="
+ bug_id;
    var url = "https://www.iplanet.com/cgi-bin/gx.cgi/
AppLogic+WebCall.CaseDet
ails?case_id=" + case_id;
var theWebCallURL = "https://www.iplanet.com:443";
    var url = "http://www.iplanet.com/bugsplat/show_bug.cgi?id="
+ bug_id;
    var url = "https://www.iplanet.com:443/WebCall/wait.html";
```

Because `url` is in the gateway profile by default and all of the right-side values are raw URLs, the only rule to be added is `theWebCallURL`, which will go in the Rewrite JavaScript in URLs section. Note that only the assignment operator is being matched for files with a `js` extension and that have a protocol identifier wrapped in double quotes, one space after the assignment in this particular example.

The following is an example of how to look at the onClick JavaScript event handlers to see if any content requires gateway profile entries:

```
$ find ./htdocs \( -name '*.htm' -o -name '*.html' \) -exec grep -i
"onClick\=" {} \;
document.write('ONCLICK="parent.tabSet--" ONMOUSEOVER="status=\'Back\';
return true;">');
document.write('ONCLICK="parent.tabSet++" ONMOUSEOVER="status=\'More\';
return true;">');
document.write('<TD ROWSPAN="2"><A HREF="javascript:location.reload()"
TARGET="_self" ONCLICK="openTab('+id+', \''+link+'\');" ');
document.write('<TD ROWSPAN="2"><A HREF="javascript:location.reload()"
TARGET="_self" ONCLICK="openTab('+id+', \''+link+'\')" ');
document.write('ONCLICK="openTab('+id+', \''+link+'\');
setDirAccess('+id+');" ONMOUSEOVER="status=\''+name+'\'; return true;"><FONT
SIZE="2" ');
document.write('ONCLICK="openTab('+id+', \''+link+'\');
setDirAccess('+id+');" ONMOUSEOVER="status=\''+name+'\'; return true;"><FONT
SIZE="2" ');
<INPUT TYPE ="button" VALUE="open Window" onclick="openWin()">
<INPUT TYPE=BUTTON VALUE="Build BugList"
onClick="location.href='http://www.iplanet.com.com/bugsplat/
buglist.cgi?bug_id=344836...'">
```

Many of the onClick event handler values do not relate to URLs at all, so they can be ignored. One entry contains location.href, which would be handled by the *.href rule suggested earlier in this section. One other thing to be concerned about is the second parameter to the openTab function. Because link is a JavaScript variable instead of raw URL, it will have to be handled either where link is first initialized to a URL value or within the body of openTab itself. Because the value is now known, you can find out where it is initialized by typing:

```
$ find ./htdocs \( -name '*.htm' -o -name '*.html' \) -exec grep
"link \=" {} \;
```

In this case, nothing is returned, which indicates that link is probably used only in the context of a function parameter to the openTab function, or possibly other functions as well. So the only way to determine how to rewrite link is by looking at the source code for the openTab function definition. If openTab did nothing more than have a function call to open another window to the link URL, then the web application source code would have to be modified to allow for rule creation.

The following is an example of the source code:

```
function openTab (id, link) {
  window.open(link,
,"displayWindow","menubar=yes,location=yes,status=yes");
}
```

Even though `window.open` is in the gateway profile, out-of-box (see "Out-Of-Box Rule Set" on page 26), `link` still cannot be resolved using syntax interpretation alone. This can be resolved by moving `window.open:y` from Rewrite JavaScript Function Parameters to the Rewrite JavaScript Function Parameters Function section of the gateway profile that will wrap the first `window.open` parameter in the `iplanet` function so that it is rewritten dynamically by the client at runtime. One thing to watch out for when doing this is that other pages are not regressed with this rule change.

One example would be if the `window.open` method were called within a JavaScript event handler in some other content. The gateway would then attempt to insert the entire `iplanet` function definition within (inline) the HTML tag itself. This problem will manifest itself in SP3 Hot Patch 3 by outputing the JavaScript code to the visible portion of the web page in the browser. With this in mind it is good to keep any values for entries in the Rewrite HTML Attributes Containing JavaScript out of either the Rewrite JavaScript Variables Function or Rewrite JavaScript Function Parameters Function sections to avoid the `iplanet` function definition appearing in the HTML tag itself.

The SP4 Hot Patch 1 release addresses this problem by moving the `iplanet` function definition to the document `HEAD` element.

Out-Of-Box Rule Set

The out-of-box rule set is responsible for rewriting the HTML 4.0 tag set and integrating content containing some of the most commonly used URL naming conventions and referencing techniques. All tag `HREF` and `SRC` attributes are rewritten out-of-box. One HTML attribute that is not currently handled out-of-box, or otherwise, is the `STYLE` attribute. The `STYLE` attribute is used as a way to insert CSS within an HTML tag, similar to how JavaScript event handlers are used to insert JavaScript function calls within an HTML tag. CSS contains a `URL` function to specify the background URL that looks like:
`background-image:URL(../../img/background.jpg)`

Fortunately, this should only affect the BODY tag when it contains inserted CSS using the STYLE attribute. The following is an example:

```
<BODY STYLE="background-image:
  url(../../img/background.jpg);background-repeat:repeat;width:770px">
```

Because the STYLE attribute is not currently handled by the rewriter, this content would have to be changed to move the BODY definition to inlined CSS between the STYLE tags instead. The following is an example:

```
<HEAD>
<STYLE
BODY { background-image:
  URL(../../img/background.jpg);background-repeat:repeat;width:770px}
</STYLE>
</HEAD>
```

The CSS can also be extracted and imported by the page, instead of defining the background image URL using the STYLE attribute. The Lotus iNotes application, for example, uses this feature to create *skins* for the user interface look and feel.

The following is the HTML source fragment:

```
<link rel="stylesheet" type="text/css"
href="https://ips-gateway.iplanet.com/http://inotes.iplanet.com/mail/
username.nsf/iNotes/Proxy/
?OpenDocument&Form=s_StyleSheet&CacheResults&MaxExpires&TimeStamp=
20011223T090041,80Z">
```

The following is the CSS source fragment:

```
.s-logo-bg {
   background-image:url(https://ips-gateway.iplanet.com/
http://inotes.iplanet.com/iNotes/Forms5.nsf
/h_ResourcesByName/iwa.gif/$FILE/iwa.gif?OpenElement&MaxExpires);
background-repeat:no-repeat;
   width:185px;
   height:43px;
   border:none;
}
```

Besides handling the HREF and SRC attribute values out-of-box, the rewriter also has entries to rewrite the APPLET and OBJECT tag CODEBASE attribute and the FORM ACTION, BODY BACKGROUND, and APPLET ARCHIVE attribute values. With SP3 Hot Patch 3 applied, the necessary rules to integrate with the Microsoft Exchange Web interface will also be present, and several new profile attributes and/or sections are added to handle XML data. There are also rules in place to make sure that the Sun ONE Portal Server: Secure Remote Access applications are launched correctly.

Finally, there are rules to handle the Portal Server desktop JavaScript functions for the default out-of-box providers. Some of the profile attributes and/or sections are empty and should only be populated if needed. Refer to the individual attribute and/or section for the rule entry syntax.

Rewriting HTML Attributes

Although most needed HTML tag attributes are already added to the Rewrite HTML Attributes section of the gateway profile, there may be instances where new attributes are supported in HTML tags. For rules added in this profile section to be considered during translation, the MIME type for the content should be text/html and the attribute value must be a raw URL. If the attribute value contains JavaScript content instead, then the attribute name should be added to the Rewrite HTML Attributes Containing JavaScript section.

The following is an example:

```
<A TARGET="content" HREF="iim.jnlp" NAME="CHAT"
onMouseOver=document.images[0].src="images/chat2.gif"
onMouseOut=document.images[0].src="images/chat.gif";>
<IMG ALIGN="MIDDLE"
SRC="images/chat.gif" BORDER="0" ALT=" Chat"></A>
```

In this particular anchor example, there are two tags, A and IMG. The tag attributes are TARGET, HREF, NAME, onMouseOver, onMouseOut, ALIGN, SRC, and ALT. Only HREF, onMouseOver, onMouseOut, and SRC need to be considered for containing potential URLs. The HREF and SRC attributes have already been added to the Rewrite HTML Attributes section of the gateway profile. Their values are both raw URLs, so they will be rewritten correctly.

The two event attributes onMouseOver and onMouseOut have been added to the Rewrite HTML Attributes Containing JavaScript section out-of-box, so their values will attempt to be translated as URLs. This translation will be successful only if a wildcard rule has been added to the Rewrite JavaScript Variables in URLs section of the gateway profile at the SP3 Hot Patch 3 install level.

HTML BASE Tag

It is important to understand the role that the BASE tag plays in how documents are rewritten and what to expect in content that contains a BASE tag. The BASE tag is used by the browser for address completion of relative links. Instead of rewriting the BASE HREF attribute value and leaving the relative URLs alone, the rewriter comments out the BASE tag entirely and rewrites the relative URLs throughout the document by using the translated value of the BASE tag for address completion. The reason for this implementation is that multiple scraped channels can be displayed on the Portal Server desktop and that one uncommented BASE tag would affect any other Portal Server desktop content that might contain its own relative URLs.

Because the Portal Server desktop is essentially an HTML table after it is rendered, there is no way to have multiple BASE tags and have the relative URLs resolved correctly. Similarly, scraped pages that contain CSS content can adversely affect the entire Portal Server desktop if the CSS content contains generalized style definitions for basic HTML elements such as the BODY and TABLE tags.

One other limitation to be aware of is when content contains a BASE tag and an APPLET and/or OBJECT tag that does not contain a CODEBASE attribute. In this particular case, when the BASE tag is commented out, the browser will no longer be able to find the APPLET and/or OBJECT code and/or data because there will not be any prepended path information supplied. In this case, always be sure that a CODEBASE attribute is used for these, and similar tags, when a BASE tag is also used within the same document. The SP4 Hot Patch 1 release handles this case by inserting a CODEBASE attribute if one does not already exist when a BASE tag is present in the document HEAD element. Even though the BASE HREF value can be a fully qualified URL, which includes a resource name, it is recommended to end the HREF value with a directory name and a trailing slash.

The following is an example:

```
<BASE HREF="http://www.iplanet.com/docs/index.html">
<BASE HREF="http://www.iplanet.com/docs/">
```

The first instance is a valid BASE tag. The second instance will be sure to resolve relative URLs throughout the remainder of the document correctly. The SP4 Hot Patch 1 release addresses cases in which the BASE tag contains only the host and port information, but no path information, as in the following example:

```
<BASE HREF="http://www.iplanet.com.80">
```

Best Practices—HTML Programming for Use Through the Gateway

You should use the following best practices:

- Always use CODEBASE attributes for tags that support them, as in the following example:

```
<APPLET CODEBASE="http://www.iplanet.com/java/"
CODE="helloWorld.class">
```

- End BASE HREF attribute URLs with a directory name or a directory name and a following slash, as in the following example:

```
<BASE HREF="http://www.iplanet.com/docs/">
```

- Avoid fractured HTML where attribute values or tag bodies might be defined on multiple lines, as in the following example:

```
document.write("<A HREF=\"\n");
document.write("http://www.iplanet.com\n");
document.write("\">link</A>\n");
```

- Try to maintain well-formed HTML where quotes match up and they are the same type.

  Avoid nested quotes where possible, and use consistency across tag definitions, as in the following example:

```
document.write("<IMG SRC='" + theSrc + "' HEIGHT=80
WIDTH='80'>");
```

Note – Here the gateway will blindly rewrite the SRC attribute without knowing the value of theSrc variable. There may be a fix for this by the time you read this guide, so check with Sun ONE support if you experience this problem and are unable to code around it.

- Specify URLs with prepended path information whenever possible.

Having prepended path information makes it easier for the gateway to figure out address completion. The following is an example:

```
<IMG SRC="../../images/button.gif">
```

- Do not use upper case or mixed case protocol identifiers in your URLs, as in the following:

```
<A HREF="HTTP://content-server.iplanet.com">
```

- Do not attempt to mimic the rewriter behavior by adding the gateway name to the URL prior to passing the content through the gateway.
- Try to avoid setting attribute values to null if the attribute name has been added to the Rewrite HTML Attributes list. Prior to SP3 Hot Patch 3, a value of "" would still be rewritten.

  The following is an example of what to avoid prior to SP3 Hot Patch 3:

```
<FRAMESET cols="20%, 80%">
  <FRAMESET rows="100, 200">
      <FRAME src="">
      <FRAME src="test-txt2.html">
</FRAMESET>
  <FRAME src="test-txt3.html">
</FRAMESET>
```

Note – This is usually done so that JavaScript `write` methods can later be called to create the actual frame page. If this `SRC` attribute is rewritten and accessed using the Netscape Navigator browser, a directory listing may be presented, depending on the web server configuration, but the `write` methods will still execute. With the Internet Explorer browser, if the directory listing is turned off on the web server, an error redirection occurs in the browser, and the JavaScript `write` methods will no longer work. SP3 Hot Patch 3 fixed this inconsistency by simply not rewriting the null `SRC` attribute value. If white space occurs between the quotes, it will not be considered a null attribute value any longer and will be rewritten. So, it is important to ensure that the quotes occur directly next to one another to prevent unwanted rewriting from occurring.

- Avoid using the STYLE attribute with a background URL in HTML tags, as in the following example:

```
<BODY STYLE="background-image:url(../../img/background.jpg);
background-repeat:repeat;width:770px">
```

- Avoid nesting tags of the same type, which may contain content requiring translation, as in the following example:

```
<SPAN STYLE="color:blue; font-weight:bold; font-style:italic">
  <SPAN>
  Inside SPAN tag:
  <BR CLEAR="ALL">
  <A HREF="../../img/after.jpg">
    <IMG SRC="../../img/after.jpg">
  </A>
  </SPAN>
</SPAN>
<BR CLEAR="ALL">
Outside SPAN tag:<BR>
<A HREF="../../img/after.jpg">
  <IMG SRC="../../img/after.jpg">
</A>
```

Note – Prior to SP3 Hot Patch 3, the rewriter would ignore the content between nested SPAN tags.

- Avoid using a SCRIPT tag with a language attribute other than JavaScript, as in the following example:

```
<SCRIPT Language="VBScript">
```

Note – There is currently no functionality in the rewriter to handle any scripted languages other than JavaScript.

- Do not pass gzipped HTML through the gateway to be displayed by the client.

  This HTML could contain URLs that will not be rewritten because the content is in a compressed format when it passes through the gateway.

Rewriting FORM Tag Input

The FORM tag ACTION attribute will be rewritten by the gateway out-of-box. So, the only things to be concerned about when rewriting form data is the INPUT and SELECT tag value attributes and any JavaScript event handlers if they are being used.

There are eight INPUT types that have supporting attributes that can contain URLs, in addition to the OPTION tag value(s) that can have URLs as well. One of these INPUT types, image also has a supporting SRC attribute that will be rewritten by default out-of-the-box. Although the syntax for rewriting FORM INPUT values is different from other profile sections, the basic premise is the same. Only those values that contain URLs need to be considered. For example, it is unlikely that the value of a form INPUT TYPE of PASSWORD would ever be a URL. However, a drop-down menu used as a navigation tool could very well contain one or more URLs.

The general syntax for a rule added to the Rewrite Form INPUT Tags List section is that it contains at least three entries:

- Page and/or object identifier

  This is the actual name of the object including any prepended path information that directly follows the protocol, web server name, and port number in the URL.

  For example, the URL `http://www.iplanet.com/forms/signup.html` page identifier including the path would be: `/forms/signup.html`

- Form name

  This is the name of the form defined using the NAME attribute in the opening FORM tag, as in the following example:

```
<FORM NAME="menuForm">
```

- INPUT or OPTION tag NAME attribute

  This is the name of the INPUT or OPTION tag given using the NAME attribute, as in the following example:

```
<OPTION NAME="destination1" VALUE="http://www.iplanet.com">
```

- URL pattern if the right side of the value attribute assignment is not a raw URL

The following is an example:

```
<INPUT TYPE="CHECKBOX" NAME="check" VALUE="0|http://www.iplanet.com">
```

These three or four fields allow for a more granular way to control how the gateway rewrites form `INPUT` and `OPTION` data. Fields can also have wildcards for more general application.

It is usually best to be as specific as possible when generating `FORM` input rules so that `FORM` data from other pages is not unexpectedly rewritten. Consider the following example:

```
<FORM>
This is a pulldown menu:<br>
<SELECT onChange="document.location.href=this[selectedIndex].value";>
<OPTION VALUE="___" selected>Select Destination</OPTION>
<OPTION VALUE="http://www.sun.com">Sun Home Page</OPTION>
<OPTION VALUE="http://www.sun.com/solaris">Solaris Information</OPTION>
<OPTION VALUE="http://www.iplanet.com">iPlanet Home Page</OPTION>
</SELECT>
</FORM>
```

The `FORM INPUT` data can be rewritten by using the document name with its path (`/forms/signup.html`) in the first field of the rule, and the remaining part represented by a wildcard. A wildcard for all three initial fields can be used, making the rule look like: `* * *` instead of `/forms/nav.html * *`, or The limitations here should be obvious—the rules are too generalized.

However, if the `FORM` has a name, the scope of what form data would be rewritten is narrowed substantially.

```
<FORM NAME="menuForm">
This is a pulldown menu:<br>
<SELECT onChange="document.location.href=this[selectedIndex].value";>
<OPTION VALUE="___" selected>Select Destination</OPTION>
<OPTION VALUE="http://www.sun.com">Sun Home Page</OPTION>
<OPTION VALUE="http://www.sun.com/solaris">Solaris Information</OPTION>
<OPTION VALUE="http://www.iplanet.com">iPlanet Home Page</OPTION>
</SELECT>
</FORM>
```

The rule can now be expressed as: /forms/nav.html menuForm * This means that only FORM INPUT data contained in a file named nav.html, at path /forms, having a FORM NAME of menuForm, will be rewritten. Because most forms have more than one field, it may be necessary to also include NAME attributes for the FORM elements so that all of the FORM contents are not blindly rewritten.

```
<FORM NAME="menuForm">
This is a pulldown menu:<br>
<SELECT NAME="mySelect"
onChange="document.location.href=this[selectedIndex].value";>
<OPTION VALUE="___" selected>Select Destination</OPTION>
<OPTION VALUE="http://www.sun.com">Sun Home Page</OPTION>
<OPTION VALUE="http://www.sun.com/solaris">Solaris Information</OPTION>
<OPTION VALUE="http://www.iplanet.com">iPlanet Home Page</OPTION>
</SELECT>
<INPUT TYPE="HIDDEN" NAME="hidden_code" VALUE="00019283">
<INPUT TYPE="TEXT" NAME="url_field" VALUE="Enter a URL" SIZE="20">
<INPUT TYPE="SUBMIT" VALUE="Go!">
</FORM>
```

If the same rule is used in this case, the values for hidden_code, url_field, and even the SUBMIT button will all be rewritten.

Instead, the rule should be amended so that it looks like: /forms/nav.html menuForm mySelect This ensures that only the URLs in the pull down menu are rewritten.

To rewrite any other elements in the same FORM, another rule would need to be added with that particular element name in place of mySelect.

Many FORM actions involve CGI scripts that parse the query string and evaluate the FORM data. Sometimes hidden field FORM elements are used to temporarily hold data to be sent to the CGI program and may have a mixture of data in its value that might include a URL.

```
<FORM NAME="menuForm">
This is a pulldown menu:<br>
<SELECT NAME="mySelect"
onChange="document.location.href=this[selectedIndex].value";>
<OPTION VALUE="___" selected>Select Destination</OPTION>
<OPTION VALUE="http://www.sun.com">Sun Home Page</OPTION>
<OPTION VALUE="http://www.sun.com/solaris">Solaris Information</OPTION>
<OPTION VALUE="http://www.iplanet.com">iPlanet Home Page</OPTION>
</SELECT>
<INPUT TYPE="HIDDEN" NAME="hidden_code" VALUE="00019283|http://
www.iplanet.com|898239">
<INPUT TYPE="TEXT" NAME="url_field" VALUE="Enter a URL" SIZE="20">
<INPUT TYPE="SUBMIT" VALUE="Go!">
</FORM>
```

The only thing that has changed in this example from the previous one is that the value of the hidden INPUT field named hidden_code now has *pipe-separated* data values that include a URL. Rewriting the second DATA element in the value requires a URL pattern to be used in the rule syntax.

Thus, to rewrite this particular URL, a rule will have to be added that looks like:
`/forms/nav.html menuForm hidden_code *|`

The `*|` pattern indicates to the rewriter that the raw URL will begin after the first pipe symbol in the value of the VALUE attribute. The use of non-white space separators is recommended so that rule creation is made possible.

As an aside, you might be tempted to try and rewrite the onChange JavaScript event handler value instead of the FORM data for the drop down menu. But, the right side of the document.location.href assignment is not a raw URL, so the only way to rewrite it would be to add document.location.href to the Rewrite JavaScript Variables Function section of the gateway profile. In actuality, this will not work in Portal Server releases before SP4 Hot Patch 1 because the gateway will attempt to insert the iplanet function body inside of the SELECT tag.

Best Practices—HTML FORM Generation Programming for Use Through the Gateway

You should use the following HTML FORM generation best practices:

- Name all FORM and FORM-related tags that contain URLs.

  This will give you better control from the rewriter as to which FORM data will need to be rewritten, as in the following example:

```
<FORM NAME="myForm">
<INPUT TYPE="HIDDEN" NAME="myURL" VALUE="http://www.iplanet.com">
</FORM>
```

- Avoid mixing contexts within the same SELECT tag.

  In other words, do not make some options URLs while others are bare strings, as in the following example:

```
<SELECT NAME="mySelect"
onChange="document.location.href=this[selectedIndex].value";>
<OPTION VALUE="Destination1" selected>Destination1</OPTION>
<OPTION VALUE="http://www.sun.com">Sun Home Page</OPTION>
<OPTION VALUE="Destination2">Destination2</OPTION>
<OPTION VALUE="http://www.iplanet.com">iPlanet Home Page</OPTION>
</SELECT>
```

- Avoid the use of white space separators in FORM data whose VALUE attribute contains multiple elements, as in the following example:

```
<INPUT TYPE="HIDDEN" NAME="hidden_code"
VALUE="The URL is http://www.iplanet.com">
```

- Avoid using generalized rules that may unintentionally rewrite other FORM data.

  For instance, you should avoid using a rule like: `/index.html * *`
- Do not define FORM tag elements on multiple lines dynamically using JavaScript programming language, as in the following example:

```
document.write("<INPUT TYPE=\"HIDDEN\" ");
document.write("NAME=\"hidden_code\" ");
document.write("VALUE=\"http://www.iplanet.com\">");
```

- Avoid generating entire FORM data from within a SCRIPT element using multiple string concatenations, as in the following example:

```
html+='<form name="myForm" target="myFrame";
html+=action="http://www.iplanet.com/cgi-bin/send_form.pl" method="post">';
html+='<input type="hidden" value="http://www.sun.com"></form>';
```

- Avoid duplicate FORM element names on different pages, except when referring to a URL.

  This will reduce the number of rules required for FORM data, as in the following code snippets.

  - index.html snippet:

```
<INPUT TYPE="HIDDEN" NAME="field1" VALUE="http://www.iplanet.com">
```

  - page2.html snippet:

```
<INPUT TYPE="HIDDEN" NAME="field1" VALUE="9899898">
```

- Avoid using JavaScript content to change FORM element data where a URL is a JavaScript variable, as in the following example:

```
<FORM TARGET="_self"
ACTION="http://www.sun.com/cgi-bin/gen_mail.pl?uid=$UID"
onSubmit="this.MSG.value=top.homePageURL;">
<INPUT TYPE="HIDDEN" NAME="MSG" VALUE="http://www.sun.com">
</FORM>
```

- Try to handle URLs in scripted buttons in the JavaScript function body if the function parameter is not a raw URL, as in the following example:

```
<SCRIPT>
function openPage(url) {
tmpURL = url;
}
</SCRIPT>

...

<INPUT TYPE="BUTTON" NAME="Next" VALUE="-->>Next"
onClick="openPage(document.protocol + document.hostname +
document.port + '/page2.html');">
```

Note – In this case, `tmpURL` could be added to the Rewrite JavaScript Variables Function section of the gateway profile so that the parameter is rewritten properly when the button is depressed.

- Do not create dynamic paths for the page containing the `FORM` if you wish to match `FORM` data using the URL object as one of the specified fields in the rule name.

  The following is an example of what to avoid:

```
http://www.iplanet.com/cgi-bin/forms/988923/create_form.pl
```

  Where part of the path itself is a random number, such as a session ID, that is used internally by the script temporarily residing in the directory.

Note – Using wildcards for the path information will not always work.

Rewriting JavaScript Content

Because JavaScript is a programming language, there can be any number of ways to represent the same functional result. The trick is that if the result contains a URL, the gateway will need to understand that fact. The reasoning is exactly the same as the reason for rewriting HTML: If there is some URL-handling performed using

JavaScript content, then the request would still need to be sent back to the gateway component rather than attempting to directly contact the internal web application server.

For many Sun ONE Portal Server gateway administrators, rewriting JavaScript content will be the most difficult task deploying and maintaining a secure Portal Server installation. The randomness at which URLs can occur throughout scripted code contributes to the difficulty, and the fact that JavaScript content is intertwined with HTML SPEC 4.0 adds to the challenge.

Areas where JavaScript content may need to be rewritten include the obvious `SCRIPT` elements, event handlers within HTML tags, and imported JavaScript content. Specific things to look for are references to window and document object methods, events, and properties that may affect or refer to URLs. Otherwise, obvious areas may include variable assignments, function parameters, and JavaScript object arrays.

Having a good understanding of how to write JavaScript content will be very helpful when trying to mine out URLs contained in it. While the introduction to, and usage of JavaScript content, is beyond the scope of this guide, the JavaScript objects and methods that you will need to be concerned with, as they relate to the rewriter, will be covered in detail. The browser implementation of the document object model is different for different browsers, and each browser offers JavaScript content its own set of capabilities. Those that are for the most part common between the Internet Explorer and Netscape Navigator browsers are discussed in this guide.

Web Browser Document Object Properties

Document object properties are generally manipulated through a JavaScript assignment statement. In most cases, the full object path will be on the left side of the assignment operator, and a raw URL will be on the right side of the assignment operator.

With that known, you can now differentiate when a rule should be added to the Rewrite JavaScript Variables in URLs section of the gateway profile or the Rewrite JavaScript Variables Function section. Generally speaking, when the right side of a variable assignment is a raw URL and the left side is a document object property, then the full object path should be added to the Rewrite JavaScript Variables in URLs section of the gateway profile. While assignments of this type can be handled by either section, having the server rewrite the URL will save client compute resources required to render and/or use the page.

For example, consider the following JavaScript assignment:

```
document.location.href = "http://www.iplanet.com";
```

The right side is clearly a raw URL, so the rule to be added to the Rewrite JavaScript Variables in URLs section is: `document.location.href`

Consider the following assignment:

```
document.location.href=newURL;
```

There are two options to handle the rewriting of this statement. The first is to see if `newURL` can be rewritten elsewhere so that `document.location.href` would not have to be added to the gateway profile for this particular instance. The second is to add `document.location.href` to the Rewrite JavaScript Variables Function section of the gateway profile. As mentioned earlier, adding this rule will result in the entire right side being wrapped in an `iplanet` function call.

```
document.location.href=iplanet(newURL);
```

The `iplanet` function will evaluate its parameter and return a rewritten URL using the browser JavaScript engine. The `iplanet` function body will also be inserted within the `SCRIPT` tags. As of SP3 Hot Patch 3, adding a rule to this section of the gateway profile to handle a JavaScript assignment called from an event handler will result in the `iplanet` function body being defined inside of the HTML tag itself. Thus, the following code cannot be rewritten in versions before SP4 Hot Patch 1.

```
<INPUT TYPE="BUTTON" NAME="button" VALUE="Click"
onClick="document.location.href=newURL;">
```

This particular scenario will only arise if the `newURL` variable has been defined within a script tag in the document `HEAD`, due to JavaScript variable scoping. However, that also means that `newURL` must have an assignment defined in the `HEAD` element that might be rewritten there.

The following are some browser object properties that can contain URLs:

- `document.location.href`—Used to change the URL for the current page.
- `document.location.path`—Used as part of a relative URL.
- `document.location.protocol`—Used to form a URL.
- `document.location.host`—Used to form a URL.
- `document.referrer`—Used for the URL of the document which referred to the current one.
- `document.URL`—Used for the URL of the current document.

Because of the JavaScript object scoping, each of the proceeding properties can also have window prepended to the full property path. The window object itself also has several event handlers that can contain additional JavaScript content. Two of the more well-used window object events are onLoad and onUnload. window events are often located in the BODY tag of the HTML document and execute when the page is first being rendered (as in the case of the onLoad event). Frames can also use onLoad and onUnload events.

When referring to specific frames or other objects in the DOM hierarchy, the object path may differ, which would require either an additional rule or a wildcard rule, if possible. Neither window.location.href nor parent.location.href would be matched by the location.href rule. However using SP3 Hot Patch 3, if the right side is a raw URL, an entry such as *.location.href can be added to the Rewrite JavaScript Variables in URLs section of the gateway profile that will handle both cases with a single rule.

Web Browser Document Object Methods

There is not any one specific way in which to rewrite browser document object method calls that contain URLs. The rule syntax and appropriate section to be considered in the gateway profile depends on the method's parameter(s) and its semantics. For example, the window.open method takes several parameters, but they must be in a specific order, and all of them start with a URL as the first parameter. If the URL is a raw URL, then the function name can be added to the Rewrite JavaScript Function Parameters section of the gateway profile.

The syntax for a rule in this section is funcName:y, where funcName is the function name that is followed by a colon separator and either by a y or a comma. A comma is used to signify multiple parameters, and a y is used to tell the gateway that a particular parameter requires rewriting. This may be easier to understand in practice. Consider a call to window.open in the following example:

```
<HTML>
<HEAD>
<SCRIPT>
function myWin() {
window.open('/channels/stocks_channel.html','Stocks',
'width=300,height=250,directories=no,location=no,menubar=no,
scrollbars=yes,status=no,toolbar=no,resizable=yes');
}
 </SCRIPT>
</HEAD>
<BODY onLoad="myWin()";
</BODY>
</HTML>
```

After this document loads, another with stock information will automatically open. The `onLoad` attribute could be listed in the Rewrite HTML Attributes Containing JavaScript section of the gateway profile, but it is not necessary in this case because the right side of the assignment operator is not a raw URL, but instead, a function call. Inside the body of the `myWin` function, however, there is a call to the browser document object method `window.open`, whose first parameter is a relative URL. So, the following rule would be added to the Rewrite JavaScript Function Parameters section of the gateway profile: `window.open:y`

This particular rule has already been added to the gateway profile, out-of-box, but it is referred to, in this case, to explain when and how to add a rule to this particular section of the gateway profile. If the content is passed through the gateway that contains a call to `window.open`, where the first parameter is not a raw URL, then the `window.open:y` method would have to be moved from the default section of the gateway profile to the Rewrite JavaScript Function Parameters Function section.

Consider the example:

```
<HTML>
<HEAD>
<SCRIPT>
function myWin() {
myURL = '/channels/stocks_channel.html'; window.open(myURL,'Stocks',
'width=300,height=250,directories=no,location=no,
menubar=no,scrollbars=yes,status=no,toolbar=no,resizable=yes'); }

</SCRIPT>
</HEAD>
<BODY onLoad="myWin()";>
</BODY>
</HTML>
```

There are two ways to rewrite the page so that it will function correctly. The first, which has been discussed already, is to add `myURL` to the Rewrite JavaScript Variables in URLs section of the gateway profile. The second is to move `window.open:y` from the Rewrite JavaScript Function Parameters section of the gateway profile to the Rewrite JavaScript Function Parameters Function section. Although the name may be a bit confusing, this latter profile section works similarly to the Rewrite JavaScript Variables Function section, in that an `iplanet` function is defined and the variable, or parameter in this case, is then wrapped within an `iplanet` function call. However, because `window.open` and other browser object methods are often called directly by event handlers, it is better in this case to rewrite only the `myURL` variable to avoid the problem where the `iplanet` function body is inserted in the HTML tag itself, unless you have SP4 Hot Patch 1 installed.

The following are some other browser object methods that can contain URLs:

- `document.assign`—Sets the `document.location.href` property value.
- `document.write`—Used for dynamically creating content using the client computing resources.
- `document.writeln`—Used in the same manner as `document.write` with the added benefit of a line break.

JavaScript Object Arrays

One other way JavaScript code is used to manipulate URLs is through the use of the default JavaScript object arrays. These provide accessor functionality to the JavaScript content so that attribute values can be changed dynamically after the page is rendered in the browser. Object arrays can be used in a variety of ways, including a `mouseOver` for image buttons, preloading content such as images that will be used in a JavaScript animation, or `FORM` field changes or checks.

Most of the arrays containing URLs that you may need to address are anchors, Applets, forms, frames, images, and links. The syntax of the JavaScript object array references includes the full object path, array name, index value, and attribute to change.

The following is an example:

```
function preLoadImages() {
  this[1] = new Image();
  this[1].src = "image1.gif";
  this[2] = new Image();
  this[2].src = "image2.gif";
}
if (document.images) {
  preLoadImages();
}
```

This example may force the page to take a bit longer to download because all of the images are fetched first, even if they are not initially displayed. This is common practice for `mouseOver` events or animations where the usability of the page depends on quick retrieval (in this case, from the browser cache) of the images. The result of this code running is that the image object will be populated and can be accessed using the JavaScript `document.images[index].src` object array. It is important to note that because the rewriter operates using regular expressions, brackets have special meaning and cannot be used in the rule entries.

As previously mentioned, SP3 Hot Patch 3 allows wildcards in the Rewrite JavaScript Variables in URLs section of the gateway profile to correctly handle JavaScript object array references. Thus, the correct rule for the above example would be: `this*.src`

Be sure to include the attribute name `SRC` in the rule to avoid possible problems with the image constructor attempting to be rewritten or similar problems if the image names, or some other attribute value, were also initialized by the `preLoadImages` function. Using wildcards can be even more beneficial when the index is created dynamically.

The following is an example:

```
<HTML>
<HEAD>
<TITLE>Test for rewriting JavaScript Arrays - RFE #4504371 </TITLE>
<SCRIPT>
<!--
  function depress(imgNum){
    if (imgNum == 1) {
      document.images["IMG"+imgNum].src = "../../img/Back_lit.gif";
      liftUp(2);
    }
    else if (imgNum == 2) {
      document.images["IMG"+imgNum].src = "../../img/Forward_lit.gif";
      liftUp(1);
    }
  }

  function liftUp(imgNum) {
    if (imgNum == 1) {
      document.images["IMG"+imgNum].src = "../../img/Back.gif";
    }
    else if (imgNum == 2) {
      document.images["IMG"+imgNum].src = "../../img/Forward.gif";
    }
}
//-->
</SCRIPT>
 </HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
<A HREF="#" onClick="depress(1);"><IMG SRC="../../img/Back.gif"
NAME="IMG1" BORDER="0"></A>
<A HREF="#" onClick="depress(2);"><IMG SRC="../../img/Forward.gif"
NAME="IMG2" BORDER="0"></A>
</BODY>
</HTML>
```

Note – This particular example will not work on a Netscape Navigator 4.x browser in the Solaris™ Operating Environment.

Here again, the only rule required to rewrite the JavaScript images array reference is: `document.images*.src`

The `*.src` can also be used to reduce rule clutter and maintain performance by limiting the number of rules that the gateway has to compare when rewriting JavaScript content. The leading period is still included to avoid accidentally rewriting of other assignments whose left side ends in SRC. Using a wildcard here, as in most cases, can be just as dangerous as it is beneficial. A rule of thumb is that specificity limits unintended consequence at the expense of flexibility.

Specialized JavaScript Variables

There are a few built-in JavaScript variables for which a relative URL is required, rather than translating its value into an absolute URL. `location.pathname` for instance should only specify the path portion of a URL. Normally, if a page containing `location.pathname` is accessed through the gateway, its value would incorrectly contain '`redirect/`', in addition to the protocol and server where the content originated from instead of just the relative path.

Starting in SP3 Hot Patch 1, there is a special section of the gateway profile, called Rewrite JavaScript System Variables Function, that is set aside for special variables. Similar to how the `iplanet` function is used to dynamically rewrite URLs using the browser's JavaScipt engine, another function called `iplanet_pathname` is used to do the same thing for built-in JavaScript variables whose values need to remain relative URLs.

The following is an example:

```
<HTML>
<HEAD>
<TITLE>JavaScript Test</TITLE>
<SCRIPT LANGUAGE="JAVASCRIPT">
var pathname = window.location.pathname

</SCRIPT>
</HEAD>
<BODY>
<P>This page tests the windows.location.pathname system variable.</P>
</BODY>
</HTML>
```

The above example will be rewritten as:

```
<HTML>
<HEAD>
<TITLE>JavaScript Test</TITLE>
<SCRIPT LANGUAGE="JAVASCRIPT">
 var pathname = iplanet_pathname(window.location.pathname)
function iplanet_pathname(thePath) {
    newPath = thePath.substr( thePath.indexOf( "/",
thePath.lastIndexOf("://") + 3 ))
    return newPath
}
</SCRIPT>
</HEAD>
<BODY>
     <P>This page tests the windows.location.pathname system
variable.</P>
</BODY>
</HTML>
```

Entries in this section of the gateway profile will likely be sparse, but this section can be useful if the content defines its own object rather than using the `window` object.

The following is an example:

```
<HTML>
<HEAD>
<SCRIPT>
function aFunc(myPage){
  var URL=myPage.location.pathname; // contains full URL w/o protocol
  var lowerURL = URL.substring(0, URL.toLowerCase().indexOf(".html")) +
".html";
  return lowerURL;
}
 var newURL = aFunc(self) + "/cgi-bin/aCGI?val1=foo&val2=bar";
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
</BODY>
</HTML>
```

In this case, `myPage.location.pathname` would have to be added to the Rewrite JavaScript System Variables Function section for the expected behavior to occur.

Nested JavaScript Code

Nesting JavaScript code makes the mining out of URLs a bit more difficult and adding the correct rules a bit more challenging. For instance, the `window` object method `setTimeout` takes an expression as a first parameter that can itself be a JavaScript function call.

The following is an example:

```
<HTML>
<HEAD>
<SCRIPT>
function statusMsg(msgURL) {
  window.status = msgURL;
}

</SCRIPT>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
<SCRIPT>
window.setTimeout("statusMsg('http://www.iplanet.com')", 1000);
</SCRIPT>
</BODY>
</HTML>
```

Assume that you could not work around this by rewriting the `window.status` assignment by adding it to the Rewrite JavaScript Variables Function section of the gateway profile. Instead, you are able only to rewrite the `window.setTimeout` statement for the application to work correctly.

In this case, two things have to be done. First, the rewriter needs to know that the first parameter of the `window.setTimeout` function call contains JavaScript content. So, the `window.setTimeout:y` rule must be added to the Rewrite JavaScript Function Parameters in JavaScript section of the gateway profile. Secondly, the rewriter needs to know that the first parameter of the `statusMsg` function is a raw URL, so the `statusMsg:y` rule must be added to the Rewrite JavaScript Function Parameters section.

Consider the following example:

```
<HTML>
<HEAD>
<SCRIPT>
var curURL = 'http://www.iplanet.com';
function statusMsg(msgURL) {
  window.status = msgURL;
}

</SCRIPT>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
<SCRIPT>
window.setTimeout("statusMsg(curURL)", 1000);
</SCRIPT>
</BODY>
</HTML>
```

The `setTimeout` statement can only be rewritten by adding `statusMsg:y` to the Rewrite JavaScript Function Parameters Function section of the gateway profile, starting in the SP4 Hot Patch 1 release. Otherwise, the only way to rewrite this example correctly is to rewrite the `curURL` assignment statement.

Event Handlers

Event handlers are a special kind of HTML tag attribute whose value can contain JavaScript content. Most event handlers begin with the letters *on* and are initiated by different user actions, such as mouse events or keyboard activity. Many event handlers have already been added to the gateway profile, out-of-box, and they can be seen by looking at the Rewrite HTML Attributes Containing JavaScript section of the gateway profile. Basically, by adding a value to this section, it must be a valid HTML attribute whose value contains JavaScript content. The value of the attribute is then translated by the gateway as JavaScript content.

Imported JavaScript Files

Rewriting imported JavaScript files follows the same basic principles as rewriting inserted JavaScript content, except that relative URLs may not be handled correctly. Because the Netscape Navigator browser does not send an HTTP Referrer header in the imported JavaScript request, there is no way for the rewriter to determine the exact URL to be used as the base.

Consider the following HTML source:

```
<HTML>
<HEAD>
<SCRIPT SRC="scripts/test.js"></SCRIPT>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">

</BODY>
</HTML>
```

Here is the JavaScript source:

```
imgsrc = "images/test.jpg";

window.open(imgsrc,'test');
```

If the page was accessed at:

`http://www.iplanet.com/importedjs.html`

And, if `imgsrc` had already been added to the gateway profile, then the `imgsrc` value would be rewritten as:

`https://ips-gateway.iplanet.com/http://www.iplanet.com/scripts/images/test.jpg`

Instead of:

`https://ips-gateway.iplanet.com/http://www.iplanet.com/images/test.jpg`

The reason for this is that the `SCRIPT SRC` attribute value will be rewritten to: `https://ips-gateway.iplanet.com/http://www.iplanet.com/scripts/test.js`

Without having a Referrer header, this value will be used to resolve the relative links in the imported JavaScript file.

There is currently no fix for this limitation from AOL for the 4.*x* or 6.*x* Netscape Navigator browser. However, there is a fix available in SP4 Hot Patch 1 for the Internet Explorer browser that makes use of the Referrer header to resolve the relative links correctly.

Note – If the imported JavaScript content was changed to:

```
imgsrc = "/images/test.jpg";
window.open(imgsrc,'test');
```

where the prepended path information in the `imgsrc` value is relative to the server root, then the statement will be rewritten correctly only because the server where both the image and the script reside is the same.

Imported JavaScript content is used by many web applications to make the page source cleaner and as a way to *hide* intellectual property contained in the JavaScript content. URLs are not always contained in imported JavaScript content, but it is still a good idea to check. One way to view the imported JavaScript source, without having to dig through the browser cache, is to create your own Web page with links that point to the remote JavaScript file.

The following is an example:

```
<HTML>
<BODY>
<A HREF="http://www.iplanet.com/scripts/test.js">test.js</A>
</BODY>
</HTML>
```

Using the Netscape Navigator browser, you can access this page, right-mouse click over the link, and choose Save As. After you have the JavaScript file saved, you can determine what, if anything, will need to be rewritten for the application to work correctly through the gateway. If this is not possible, you can also refer to the `iwtGateway` log file and look for the actual imported JavaScript content to see if it contains URLs. The `ips.debug` value needs to be changed from `error` to `message` before looking for URLs in the gateway log. For performance reasons, it is not recommended to keep the log level set to `message` for an extended period of time in a production environment.

Dynamically Created HTML Blocks

One of the features that makes JavaScript programming language so attractive to web application developers is its ability to manipulate multiple windows and frames dynamically using the client JavaScript engine. The functionality is used for navigation purposes, site maps, form handling, and ad generation.

Consider the following example:

```
<HTML>
<HEAD>
<TITLE>Tests rewriting of dynamically created HTML blocks</
TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
<SCRIPT>
myWindow = window.open("","myWindow");
mySrc = "<HTML>" +
        "<BODY BGCOLOR=#FFFFFF TEXT=#000000>" +
        "<IMG SRC='/images/logo.gif'>";
self.myWindow.document.write(mySrc);
self.myWindow.document.close();

</SCRIPT>
</BODY>
</HTML>
```

This example essentially demonstrates opening a new window and then creating the source necessary for an image to be displayed in the new window. What sets this example apart from other JavaScript rewriting examples given thus far is that the `IMG SRC` attribute is created dynamically by the client rather than passing through the gateway, which would normally translate the `SRC` attribute value. So, the rewriter needs to know that the JavaScript variable `mySrc` contains HTML content that should be rewritten accordingly.

To do this, the `mySrc` rule needs to be added to the Rewrite JavaScript Variables in the HTML section of the gateway profile. So, if pop-up windows in your web application are not working correctly or simply coming up blank, it may be due to the fact that there is an HTML block created dynamically using JavaScript code that is not being rewritten correctly.

Note – If the `window.open:y` rule is present in the gateway profile, the first parameter will be rewritten even if it is null. This may cause problems in the Internet Explorer browser if the directory listing has been turned off on the web server. If possible, it is better to point to an empty HTML file instead. There may be a fix available by the time you read this document. Contact Sun ONE technical support if you experience this problem.

JavaScript Code Used to Create JavaScript Content

Relatively rare cases to be aware of when rewriting JavaScript content are statements whose right side contains variable initializations that could have a right side that is a URL string literal.

The following is an example:

```
<HTML>
<HEAD>
<SCRIPT>
tmpURL = "var address = 'http://www.iplanet.com'";
</SCRIPT>

</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">

</BODY>
</HTML>
```

In this particular case, `tmpURL` becomes a string object which contains `var address = 'http://www.iplanet.com'`. This syntax is useful in multi-windowing applications where JavaScript content in the parent is responsible for writing JavaScript content in the child window as a result of user events. For the URL to be rewritten correctly, a rule address needs to be added to the Rewrite JavaScript Variables in URLs section of the gateway profile, and the `tmpURL` rule needs to be added to the Rewrite JavaScript Variables in JavaScript section.

JavaScript Code Obfuscators

Obfuscation can come in many forms including scramblers, optimizers, and full-on obfuscation, whose only purpose is to make it difficult for the hacker to pick off publicly available JavaScript source embedded in pages. Usage of JavaScript code obfuscation site-wide in an ASP environment, a business-to-business model, or in a variety of other scenarios may make rewriting difficult, if not impossible, depending on the nature of the JavaScript code and how the code obfuscator works. If the obfuscation is done dynamically, then it probably will not be possible to create static rewriter rules that will predictably rewrite embedded URLs.

Different obfuscators contain different features, but to effectively obfuscate the JavaScript code, function names and variables are typically altered. The same code altered twice may contain different names, while other altered code may contain the same names.

The following is an example of the source for page 1:

```
<SCRIPT>
text = "hi there";
</SCRIPT>
```

The following is the source for page 2:

```
<SCRIPT>
url = "http://www.iplanet.com";
</SCRIPT>
```

After being run through JavaScript code obfuscation, both pages may look like the following:

```
<SCRIPT>x002323="hi there";</SCRIPT>

<SCRIPT>x002323="http://www.iplanet.com";</SCRIPT>
```

If JavaScript code obfuscation is a requirement, then URL references must be extracted from the code for obfuscation and handled a different way such as inserting them in a top level frame or importing the raw JavaScript content containing the URL variables. You can use an obfuscator with configurable code generation in which variables and function names can be mapped.

The following is an example of an obfuscator that maps variables to three-letter codes:

```
url -> scf
```

You could then add `scf` to the appropriate section of the gateway profile.

Best Practices—JavaScript Programming for Use Through the Gateway

You should use the following best practices:

- Do not create variable references with dotted object paths whose right side is not a raw URL, as in the following example:

```
this.foo = protocol + server + path + resource;
```

- Avoid deeply nested JavaScript content.

  The more difficult and spaghetti-like the code is, the harder it is to get it to work through the gateway.
- Use absolute URLs or URLs relative to the content server root in imported JavaScript files.

  This is necessary until the Portal Server is certified for use with the 6.0 browsers and until both of them send a Referrer header when requesting a remote JavaScript file. Inlined JavaScript content can also be used, which does not have this limitation.
- Avoid mixing JavaScript variables with HTML blocks that are created dynamically, as in the following example:

```
<HTML>
<HEAD>
<TITLE>Tests rewriting of dynamically created HTML blocks</
TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
<SCRIPT>
var myURL = "http://www.iplanet.com";
myWindow = window.open("","myWindow");
mySrc = "<HTML>" +
        "<BODY BGCOLOR=#FFFFFF TEXT=#000000>" +
        "<A HREF='" +
        myURL +
        "'>link</A>";
self.myWindow.document.write(mySrc);
self.myWindow.document.close();

</SCRIPT>
</BODY>
</HTML>
```

- Avoid defining URLs using multiple string concatenations or in different locations in the code, as in the following example:

```
<SCRIPT>
url = "http://www.iplanet.com";
url += "/scripts/gen_form.pl";
url += "?var1=foo&var2=bar";
url2 = url + url2;
</SCIPT>
```

- Avoid making assumptions about what a URL should look like.

  More specifically, do not assume that in the JavaScript code, a particular URL will be relative in nature.
- Avoid mismatched quotes after assignment statements, as in the following example:

```
<HTML>
<HEAD><TITLE>Test Case for rewriting src tags twice</TITLE></HEAD>
<SCRIPT LANGUAGE="JavaScript">

lnk = "http://www.iplanet.com";

document.write('<img border="0" src="'+lnk+'/images/logo.jpg"
alt="ALT"></A><BR>\n')
document.write('<img border="0" src="/images/logo.jpg" alt="ALT"></A>')

</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Note – In the SP4 Hot Patch 1 release, this limitation is resolved in this particular case, but there may be other similar corner cases that will continue to fail to be rewritten correctly.

- Avoid using JavaScript code obfuscation.
- Use standardized naming conventions for URLs throughout the code, and use them in the same context.

  Notice the inconsistent naming conventions in the following two examples.

The following is in the source for page 1:

```
<SCRIPT>
url = "http://www.iplanet.com"
</SCRIPT>
```

The following is in the source for page 2.

```
<SCRIPT>
tmpURL =  "http://www.iplanet.com";
url = "The URL is" + tmpURL;
</SCRIPT>
```

- Do not over-generalize when specifying wildcards in rules, as in the following examples:

```
*location
*src
```

Note – The `*location` wildcard is more generalized than the `*.location` wildcard.

- Do not attempt to overload JavaScript Function Parameter rules, as in the three rules specified in the Rewrite JavaScript Function Parameters section of the gateway profile.

```
openMyWin:,y,,
openMyWin:y,,y,
openMyWin:y
```

Note – Only the first rule will be matched. The individual function definition should determine how many parameters are passed using the `length` method, instead of defining multiple functions with the same name in different pages. This would also eliminate the problem with the argument type and the order.

The following are more examples of overloaded rules:

```
menu.addItem(
  new NavBarMenuItem("Info",
"JavaScript:top.location='http://www.iplanet.com'"));
menu.addItem(
  new NavBarMenuItem("Info","http://www.iplanet.com"));
```

Note – The SP4 Hot Patch 1 release offers a new advanced gateway profile section that handles ambiguous JavaScript function parameters that are either URLs or JavaScript code, as in the example above. For more information, refer to the SP4 Hot Patch 1 release notes that are included with the patch.

Rewriting Applet Parameters

As with dynamic web applications created using JavaServer Pages technology, Perl, or Servlets, it is good practice not to hard code variables or URL references in Java™ applets. One mechanism to provide the applet with the location to resources or values required for the applet to run is to use applet parameters that appear as a part of the `APPLET` or `OBJECT` element using `PARAM` tags.

The `APPLET` and `OBJECT` tags have two attributes related to Java applets that are rewritten out-of-box. The first attribute, `ARCHIVE`, contains a location to a Java JAR archive. The second, `CODEBASE`, is used in conjunction with the `CODE` attribute to determine where the executable Java bytecode resides.

The `CODEBASE` attribute is also used when the Java `getCodeBase()` method is called from the applet.

Note – If an HTML `BASE` tag is included in the page, then the `APPLET` or `OBJECT` tag must contain a `CODEBASE` attribute. Otherwise, the applet may not load correctly because the `BASE` tag will be commented out after passing through the gateway. The executable Java code will attempt to be fetched using the Referrer header, which probably will not work. The SP4 Hot Patch 1 release explicitly adds a `CODEBASE` attribute to the `APPLET` tag if one does not exist and if a `BASE` tag exists in the `HEAD` element of the document.

The applet parameters can be rewritten using rules similar in syntax to FORM data. The rules allow wildcards so that patterns can be appropriately matched. The general syntax for a rule added to the Rewrite Form Input Tags List section of the gateway profile contains at least three entries:

1. Page or object identifier

   Unlike when writing FORM data, this entry includes only the prepended path information to the page containing the APPLET or OBJECT tag.

   The URL `http://www.iplanet.com/applets/welcome.html` page identifier would be: `welcome.html`

2. Class name, including its extension

   This is the value of the CODE attribute, as in the following example:

```
<APPLET CODEBASE="http://www.iplanet.com/applets/"
CODE="myClass.class">
```

3. Parameter name

   This is the name of the PARAM tag given using the NAME attribute, as in the following example:

```
<PARAM NAME="headergraphic" value="/applets/images/banner.gif">
```

A URL pattern if the right side of the value attribute assignment is not a raw URL, as in the following example:

```
<PARAM NAME="headergraphic" value="98234|/applets/images/banner.gif">
```

The following is an example of an APPLET tag with supporting PARAM values:

```
<HTML>
<BODY>
<APPLET CODEBASE="http://www.iplanet.com/applets/" CODE="hello.class">
<PARAM NAME="leftImg"  value="/images/leftImg.gif">
<PARAM NAME="rightImg" value="/images/rightImg.gif">
</APPLET>
</BODY>
</HTML>
```

If the page was accessed from `http://www.iplanet.com/welcome.html`, then the two rules to add to the Rewrite Applet/Object Parameter Values List would be:

```
welcome.html hello.class leftImg
welcome.html hello.class rightImg
```

These rules will handle data that contains separators and URLs, following the same guidelines for wildcards in the URL pattern described in "Rewriting `FORM` Tag Input" on page 33.

Best Practices—Java Programming for Use Through the Gateway

You should use the following best practices:

- Always use a `CODEBASE` attribute with an `APPLET` or `OBJECT` tag.
- Use prepended path information for `PARAM` tag values that contain URLs.

  The following is an example of what to avoid:

```
<HTML>
<BODY>
<APPLET CODEBASE="http://www.iplanet.com/applets" CODE="hello.class">
<PARAM NAME="headergraphic"  value="images/banner.gif">
</APPLET>
</BODY>
</HTML>
```

- Do not add a `CODE` rule to the Rewrite HTML Attributes section of the gateway profile.

  This results in the applet not loading properly because the browser will not be able to find the Java byte code.
- Do not make network connections to hard-coded URLs.

  Instead, they should be passed through applet parameters.

  The following is an example of what to avoid:

```
URLConnection myConn =
(new URL('http://www.iplanet.com')).openConnection();
```

- Do not allow users to specify their own URLs in the applet user interface.

  If this is required for the application to function, then you should consider running it using the netlet, instead of accessing it directly through the gateway component.
- Use PARAM names when specifying rewriter rules to avoid unintentionally rewriting other APPLET PARAM values.

  You should avoid using rules that look like: /path * *
- Define the APPLET or OBJECT opening tag on a single line.

  If not, you risk the PARAM values not being rewritten (for more information, refer to Sun BugID 4647955). The following is an example of what to avoid:

```
<APPLET code=foo.class
codebase="/foo/foo2">
<PARAM NAME=url VALUE="/somedir/some.html">
</APPLET>
```

Note – This case can be handled in the SP4 Hot Patch 1 release by using a new gateway profile section that addresses fractured HTML. However, this type of configuration change should be avoided unless it is absolutely necessary.

Rewriting Cascading Style Sheets

Cascading Style Sheets Level 1 (CSS1) is supported by the gateway, out-of-box. Support for rewriting imported style sheets started in SP3 Hot Patch 1. As of SP3 Hot Patch 3, the `background-image:url()` statement can be case insensitive. The usage of URLs is light in CSS because it seeks to replace image-heavy pages by offering useful styling alternatives.

Every sheet in the cascade will be rewritten where appropriate save for a cascade created using the STYLE attribute, sometimes referred to as an inline sheet. The difference between an internal sheet and an inlined sheet is that the STYLE tag generally contains multiple CSS rules or statements and selector class definitions; whereas, the STYLE attribute usually defines a single multivalued CSS property. Because only the background-image property needs to be rewritten, there is no separate section of the gateway profile specifically for CSS.

Best Practices—CSS Programming for Use Through the Gateway

You should use the following best practices:

- Do not add STYLE tags within the page content.

  While there is no requirement for the STYLE tag to appear within the HEAD element, it cannot occur in the middle of the BODY element, even if the BODY is only implied.

  The following is an example of what to avoid:

```
<HTML>
<HEAD>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
Table 1:
<STYLE>
  P {background-image:url('/images/logo.gif');}
<STYLE>
<TABLE>
  <TR WIDTH="100" HEIGHT="100">
<TD WIDTH="100" HEIGHT="100"><P>Para1</P></TD>
  </TR>
</TABLE>

</BODY>
</HTML>
```

- Avoid creating STYLE tags and content using JavaScript content.

Nesting STYLE tags within SCRIPT tags does not work in the rewriter, prior to the SP4 Hot Patch 1 release. The following is an example of what to avoid:

```
<HTML>
<HEAD>
<SCRIPT>
var styleTags = '<STYLE></STYLE>';

</SCRIPT>

</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">

</BODY>
</HTML>
```

- Avoid adding styles using the STYLE attribute.

  Instead, create the style using the STYLE element within the HEAD element. A selector class can be defined, and the class name can be referred to in the appropriate HTML tag, as in the following example:

```
<HTML>
<HEAD>
<STYLE>
  SPAN.logo {
    background-repeat: no-repeat;
    background-width: 116px;
    background-height: 61px;
    background-image: url(/images/logo.gif);
    background-position:top left;
  }
</STYLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
<SPAN CLASS="logo">Here is your logo</SPAN>
</BODY>
</HTML>
```

The following is an example of what to avoid:

```
<HTML>
<HEAD>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
<SPAN STYLE="background-image: url(/images/logo.gif)">Here is your logo</SPAN>
</BODY>
</HTML>
```

Rewriting XML

Currently, the Internet Explorer 5.5 browser is the only browser certified for use with the Portal Server that supports inline XML tags. The SP3 Hot Patch 1 release added two new gateway profile sections to handle the rewriting of XML data. They are Rewrite Text Data of XML Document and Rewrite Attribute Value of XML Document. The two lists are pre-populated with the entries required for Outlook Web Access 2000 to work through the gateway. See “Exchange” on page 79 for other necessary changes.

Rewriting XML PCDATA

PCDATA is essentially the text between the XML tags. It is defined as #PCDATA in the DTD. To rewrite PCDATA, you need only add the tag name to the Rewrite Text Data of XML Document section of the gateway profile.

The following is an example:

```
<?xml version="1.0"?>
  <mytag>
     http://www.iplanet.com
  </mytag>
</xml>
```

To rewrite this URL, add `mytag` to the Rewrite Text Data of XML Document list. There may be cases where the PCDATA is dependent on the tag attribute values. In these cases, both the tag name and the attribute definition need to be added to the Rewrite Text Data of XML Document profile section.

The following is an example:

```
<?xml version="1.0"?>
  <mytag myattr1="desc">
    iPlanet Home Page
  </mytag>
  < mytag myattr2="href">
     http://www.iplanet.com
   </mytag>
</xml>
```

In this case, it is clear that the `PCDATA` of the `mytag` element containing the attribute `myattr1` should not be translated. To avoid this, the rule should be specified as: `mytag,myattr2=href`

Rewriting XML Tag Attributes

As with HTML attributes, XML attribute values can contain URLs. To rewrite XML attribute values, a rule containing the attribute name needs to be added to the Rewrite Attribute Value of XML Document section of the gateway profile.

The following is an example:

```
<?xml version="1.0"?>
  <mytag desc="iPlanet Home Page"/>
  <mytag url="http://www.iplanet.com"/>
</xml>
```

In this example, the rule `myattr2` would need to be added to the Rewrite Attributes Value of XML Document profile section. To differentiate specific tag names containing the same attribute names from having their values rewritten, specify a rule with the syntax `attrName,tagName` to the Rewrite Attribute Value of XML Document list.

The following is an example:

```
<?xml version="1.0"?>
  <mytag myattr="iPlanet Home Page"/>
  <urltag myattr="http://www.iplanet.com"/>
</xml>
```

To rewrite the correct attribute value, the `myattr,urltag` rule would need to be added to the Rewrite Attribute Value of XML Document profile section.

Best Practices—XML Programming for Use Through the Gateway

You should use the following best practices:

- Try to use unique attribute names specific to URLs.

  You should be sure they are all used in the same context. The following is an example of what to avoid:

```
<?xml version="1.0"?>
  <myTag>
    <tag1 url="The URL is:">myValue</tag1>
    <tag2 url="http://www.iplanet.com">myValue</tag2>
    <tag3 url="http://www.sun.com">myValue</tag3>
</myTag>
```

- Do not generate XML content containing URLs dynamically using JavaScript content.

  There are ways to parse HTML and JavaScript values created using JavaScript write methods, but there is no way to parse XML created in the same way.

  The following is an example of what to avoid:

```
<SCRIPT>
document.write('<mytag url="http://www.iplanet.com"/>\n');
</SCRIPT>
```

- Perform user-agent checks to ensure that the browser supports XML before emitting XML code to be rendered by the browser.

Performance

Performance is an important consideration when deploying a high usage multi-user Portal Server. Detailed maximum throughput in a baseline architecture is beyond the scope of this guide, but gateway tuning parameters are discussed, along with some basic guidelines about creating rewriter rules and their affect on the overall performance of the rewriter.

Tuning Recommendations

This section contains several tables with tuning recommendations.

The following are the `/etc/system` entries:

Entry	Value
`tcp_conn_hash_size`	8192
`rlim_fd_max`	16384
`rlim_fd_cur`	16384

The following are the TCP parameter values that are set on both the Portal Server and gateway components:

Entry	Value
`tcp_time_wait_interval`	60000 (60 seconds)
`tcp_conn_req_max_q`	1024
`tcp_conn_req_max_q0`	4096
`tcp_slow_start_initial`	2

Use `ndd`(1M) to check or change the default TCP parameter values to those specified above, as in the following example:

```
root@ips-gateway: ndd /dev/tcp tcp_time_wait_interval
240000
root@ips-gateway: ndd -set /dev/tcp tcp_time_wait_interval 60000
root@ips-gateway: ndd /dev/tcp tcp_time_wait_interval
60000
```

The following table contains the performance configuration for the Sun ONE Web Server 4.1 for use with Portal Server.

Parameter Name	Location	Value	Remarks
`jvm.minHeapSize`	`jvm12.conf`	134217728	64 Mbytes
`jvm.maxHeapSize`	`jvm12.conf`	805306368	768 Mbytes
	`jvm12.conf`	`"-Xgenconfig:32m, 64m, semispaces:32m, 704m, markcompact"`	The JDK 1.2.2_09 provides better performance and scalability with the `genconfig` and `optimize` options.[1]
	`obj.conf`	`Init fn="cache-init" disable="true"`	Disable Sun ONE Web Server cache for static pages and images.
`RqThrottle`	`magnus.conf`	128	With Web Server SP5, the maximum number of active threads is calculated using the formula `RqThrottle` + `MaxKeepAliveConnections`.[2]
`MaxKeepAliveConnections`	`magnus.conf`	72	

1. The format is: `-Xgenconfig:min0, max0, semispaces:min1, max1, markcompact`

Where `min0` is the minimum size of the young generation, `max0` is its maximum size, `min1` is the min size of the old generation, and `max1` its maximum size. You would typically want to tune the young generation size so that most short-lived objects are not promoted to the old generation. In general, the larger the size of a generation, the longer it takes to collect it. Note also that `max0` + `max1` must be lower than, or equal to, the `jvm.maxHeapSize`. The non-standard options (beginning with `-X`) are not guaranteed to be supported on all Java Virtual Machine implementations and are subject to change without notice in subsequent releases of the Java2 SDK. Although this setting provides better performance in our testing environment, other workload conditions may require a different setting.

2. You may slightly modify the ratio between `RqThrottle` and `MaxKeepAliveConnections`, but keep the sum of the two values around 200 to scale properly. The `perfdump` utility can be used to refine the setting.

More gateway-specific performance tuning is discussed in the SP4 Hot Patch 1 release notes.

Order Importance

Your highest usage rules should appear at the top of the gateway profile section lists. Matches are made using top-down ordering. The first rule to match the syntax will be used. This is the reason JavaScript function parameter rules cannot be overloaded and why the same rule cannot occur in different gateway profile sections for the same rewriting environment and context.

CASE Studies: How to Configure the Gateway to Rewrite a Web-Based JavaScript Navigation Bar

This section contains case studies that are intended to show different approaches to rewriting required for different complexities of JavaScript content created for the purpose of a Navigation Bar. It also underlines the importance of understanding the content that is passed through the gateway so that you know if it needs rewriting and how to create the rules appropriate for doing so.

Case 1: Simple Navigation Bar

In this case, a simple navigation bar is created by coding each image and URL associated with the image in a line-item fashion.

```
...
..
.
<A HREF="http://url1"><IMG SRC="../../images/IMG1.gif"></A>
<A HREF="http://url2"><IMG SRC="../../images/IMG2.gif"></A>
.
..
...
```

The gateway profile already has entries that automatically rewrite HTML tag attributes whose name is either `SRC` or `HREF`.

Prior to SP3 (or possibly SP2), URLs could reference directories higher than the document root and were rewritten accordingly. For instance, if the document root was `http://server/dir1/dir2/dir3` and a URL referenced `../../`, instead of stopping at the document root, the URL would be incorrectly rewritten to: `http://server/dir1`

Case 2: Navigation Bar Using a JavaScript `MouseOver` Event

In this case, a navigation bar is created by using a JavaScript `MouseOver` event so that the image changes, appearing to be depressed.

```
...
..
.
<A HREF="http://url1" NAME="ACTION1"
onMouseOver=document.images[0].src="../../images/IMG1ON.gif"
onMouseOut=document.images[0].src="../../images/IMG1OFF.gif";>
<IMG ALIGN="MIDDLE" SRC="../../images/IMG1ON.gif" BORDER="0" ALT="ACTION1"></
A>
<A HREF="http://url2" NAME="ACTION2"
onMouseOver=document.images[1].src="../../images/IMG2ON.gif"
onMouseOut=document.images[1].src="../../images/IMG2OFF.gif";>
<IMG ALIGN="MIDDLE" SRC="../../images/IMG2ON.gif" BORDER="0" ALT="ACTION2"></
A>
.
..
...
```

The JavaScript `onHandlers` are included in the gateway profile, out-of-box. As of SP3 Hot Patch 3, the URLs in JavaScript content can contain wildcards. Before that, and in SP4, the JavaScript rules can not contain array references for JavaScript document object arrays. With SP3 Hot Patch 3, a rule would have to be added to the Rewrite JavaScript Variables in URLs section of the gateway profile like: `document.images*.src`

Because `document.images*.src` and `onMouseOver` or `onMouseOut` are included in the gateway profile, there is the possibility of something being unintentionally rewritten, depending on how the JavaScript code is formed. If the image statement was instead changed to the following:

```
lnk = "http://nsx.red.iplanet.com/test_cases/foo;
document.write('<IMG ALIGN=\"MIDDLE\" SRC=\"' + lnk +
'../../images/IMG1ON.gif\" BORDER=\"0\" ALT=\"ACTION1\">');
```

then, `lnk` would be added to the gateway profile, and the `SRC` attribute would be automatically rewritten making the URL look like the following:

```
https://GWaddr/http://nsx.red.iplanet.com/images/IMG1ON.gif/
https://GWaddr/http://nsx.red.iplanet.com/images/IMG1ON.gif
```

There is a fix available for this limitation in the SP4 Hot Patch 1 release (see “How to Get Hot Patches” on page 80).

Case 3: Navigation Bar Using a JavaScript Function Call

In this case, a navigation bar is created by using a JavaScript function call with a mouse event handler:

```
<HTML>
<HEAD>
<SCRIPT>
<!--
  function depress(imgNum){
    if (imgNum == 1) {
      document.images["IMG"+imgNum].src = "../../img/Back_lit.gif";
      liftUp(2);
    }
    else if (imgNum == 2) {
      document.images["IMG"+imgNum].src = "../../img/Forward_lit.gif";
      liftUp(1);
    }
  }
   function liftUp(imgNum) {
    if (imgNum == 1) {
      document.images["IMG"+imgNum].src = "../../img/Back.gif";
}
    else if (imgNum == 2) {
      document.images["IMG"+imgNum].src = "../../img/Forward.gif";
    }
  }
//-->
</SCRIPT>

</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">

<A HREF="#" onClick="depress(1);"><IMG SRC="../../img/Back.gif" NAME="IMG1"
BORDER="0"></A>

<A HREF="#" onClick="depress(2);"><IMG SRC="../../img/Forward.gif" NAME="IMG2"
BORDER="0"></A>

</BODY>
</HTML>
```

The same limitation applies with regard to the JavaScript document array. Having JavaScript functions can sometimes increase the flexibility of either a content-based or rule-based workaround. In this particular instance though, SP3 Hot Patch 3 or SP4

Hot Patch 1 release would have to be applied and a rule like `document.images*.src` would have to be added to the Rewrite JavaScript Variables in URLs section of the gateway profile.

You could create a variable called `up2dir` and initialize it to the prepended directory path for the images. Then, add `up2dir` to the gateway profile, and rewrite the relative content, as in the following example:

```
<SCRIPT>
  up2dir = "../../";
  .
  ..
  ...
  document.images["IMG"+imgNum].src = '"' + up2dir  + 'img/Back.gif"';
  ...
  ..
  .
</SCRIPT>
```

Case 4: Navigation Bar Using Imported JavaScript Code

In this case, a navigation bar is created by using imported JavaScript code. The following is an example of the directory hierarchy:

- `./scripts`
- `./scripts/nav.js`
- `./nav.html`
- `./images`
- `./images/nav.jpg`

Consider the following HTML source:

```
<HTML>
<HEAD>
<SCRIPT SRC="scripts/nav.js">

</SCRIPT>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
...
..
..
...
</BODY>
</HTML>
```

The following is the corresponding SCRIPT source:

```
imgsrc = "images/nav.jpg";
document.write('<IMG ALIGN=\"MIDDLE\" SRC=\"' + imgsrc + '\"
BORDER=\"0\" ALT=\"ACTION1\"
NAME=\"ACTION1\">');
```

By default, the Portal Server will rewrite this using the incorrect header information because a Referer is not supplied by the Netscape Navigator browser for imported JavaScript files. The SCRIPT SRC and the CSS SRC attributes were not rewritten prior to SP3.

The image `src` in this case would be rewritten as:

`https://GWhost/http://ContentServer/scripts/images/nav.jpg`

Instead of:

`https://GWhost/http://ContentServer/images/nav.jpg`

The SP4 Hot Patch 1 release rewrites this correctly for the Internet Explorer browser.

If the script is on the same server, you can either modify the SRC URL to contain the prepended path from the document root, or you can create a symbolic link in the scripts directory that will point to the images directory.

Case 5: Navigation Bar Using Arrays of URL Arrays

In this case, a navigation bar is created by using arrays of URL arrays.

```
.
..
...
var tabArray = new Array();
tabArray[0] = new Array();
tabArray[0][1] = 'Calander';
tabArray[0][2] = '../index_left.html';
tabArray[0][3] = 'none';
tabArray[0][4] = 'ics';
tabArray[0][5] = 'Calander calls:Calander bugs';
tabArray[0][6] = 'Home';
tabArray[0][7] = 'home';
.
...
...
var curLinkURL = top.tabArray[parseInt(top.curTab)][7].split(":");
...
..
.
```

Because the document object array occurs on the right side of the variable assignment, you cannot add `curLinkURL` to the gateway profile because the gateway will not understand that `top.tabArray` actually represents a URL of any sort. You cannot add `curLinkURL` to the Rewrite JavaScript Variables Function list because `curLinkURL` is actually an array, rather than a single value.

Again, in this case, you only want to rewrite `tabArray[0][2]`, `tabArray[0][4]`, and `tabArray[0][7]`.

While `tabArray[0][7]` and `tabArray[0][4]` do not look like a URL at first, the JavaScript code that uses the array knows that it represents a relative URL that does not contain any prepended path information. In fact, it is a colon separated list of directories that are URLs. So, the rewriter would have to know that you want to rewrite the two dimensional array at index `[0][7]` after it has been parsed on the colon. Using the syntax alone on the existing JavaScript code, there is no way to have the rewriter do this.

You can insert code between where `curLinkURL` is initialized and where it is used to individually rewrite the array values.

The following is an example:

```
for (i=0; i<curLinkURL.length; i++) {
  tmpURL = curLinkURL[i];
  curLinkURL...
}
```

You could then add `tmpURL` to the Rewrite JavaScript Variables Function section of the gateway profile. `tmpURL` cannot be added to the Rewrite JavaScript Variables in URLs section of the gateway profile because the right side of the variable assignment is not clearly a raw URL, so a domain determination cannot be made. It would not be clear what the value of the right side would be until the code actually executed.

Case 6: Navigation Bar Using Dynamically Generated URLs

In this case, a navigation bar is created by using dynamically generated URLs.

```
 for (var i=0; i<curLinkName.length; i++) {
.
..
...
document.writeln('<A HREF=\"' + curLinkDir + '/' + curLinkURL[i] +
'/index.html\" TARGET=\"content\"');
document.writeln('onClick=\"self.location.href=
  \'index_left.html\'\;parent.curLink='+i+'\;');
...
..
.
}
```

How this would look after it is rewritten entirely depends on what `curLinkDir` is. Without knowing this, it is difficult to presume if any modification is necessary. In this particular case, `curLinkDir` is array element `[0][4]` from the previous step:

```
var curLinkDir = top.tabArray[parseInt(top.curTab)][4];
```

In this case, the gateway will look at the value of the HREF attribute that it sees as a single quote. As discussed previously, prior to SP4 Hot Patch 1, the entire fully qualified URL to the Referrer will be prepended, which would look like the following:

```
<A HREF="https://GW/http://ContentServer/' + curLinkDir + '/' +
curLinkURL[0] + 'index.html\"
....
```

The above would be reduced to the following:

```
<A HREF="https://GW/http://ContentServer/ics/home/index.html
TARGET="content"..
```

For the second statement to be rewritten correctly, `self.location.href` should be added to the gateway profile under Rewrite JavaScript in URLs or possibly Rewrite JavaScript Variables Function so that the second half of the `A` tag is rewritten correctly. `curLink` in this case is an integer, so you do not need to rewrite it.

Case 7: Navigation Bar Using JavaScript Code Obfuscation

In this case, a navigation bar is created by using JavaScript code obfuscation.

```
Bx+=Bm("browse", "Select file(s)");
...
..
.
..
....
function Bm(BD, altTag){var Bn=BD + "Image"; var Bo=BD + ".gif";
var Bp='width=22 height=22 align=absmiddle alt="' + altTag + '"
buttonId="' + BD + '"'; var Bq='<td class="s-form-tool-cell">' +
NH(Md() + "/iNotes/Forms5.nsf" + "/h_ResourcesByName/" + Bo + "/$FILE/" +
Bo + "?OpenElement&MaxExpires&TimeStamp=" + haiku.sFormsTLM,Bn,Bp) + '</td>';
return Bq;}
```

Without having an intimate understanding of what this code does, it is difficult to tell what to rewrite or if anything needs to be rewritten at all. Further, because the variables do not seem to have unique names, it may be dangerous to create rewriting rules for them because they could have unforeseen side effects on other code coming through the gateway.

Additionally, this code incorporates everything previously stated at the other levels of complexity. This is code from a third party vendor that has been obfuscated for public consumption and is emitted on a single line, so disseminating it is quite difficult. Understanding it and writing rules for it is another challenge entirely.

Third Party Application Cookbooks

Sun ONE professional services can be used for integrating Interwoven, Lotus Domino, Screaming Media, SAP, iDSAME, Tarantella, Sun ONE Integration Server, Netbios, Citrix NFUSE, and Open Market. Other integrations mentioned below are strictly rewriter-specific in nature.

▼ To Set Up the Sun ONE Messaging Server 5.0

1. **In the `main.js` file, change the following line:**

```
// \5c -> \

to

//      \5c -> backslash
```

2. **Also in `main.js`, change all of the instances of `msgHREF` to `srcHREF`.**

3. **Turn IP validation off by logging in as the Messaging User and performing the following commands:**

```
$ cd <inst_dir>/server5/msg-<hostname>/
$ ./configutil -o service.http.ipsecurity -v no
$ su root
# ./stop-msg
# ./start-msg
```

▼ To Set Up the Sun ONE Calendar Server 5.0

1. **Add the following to the Rewrite JavaScript Function Parameters list.**

```
load:,y
open:y
loadtab:,y,
```

2. **Add the** `window.top.location.href` **rule to the Rewrite JavaScript Variables in URLs list.**
3. **Add the** `urlstring` **rule to the Rewrite JavaScript Variables Function list.**
4. **Log in as superuser.**
5. **Change directories to the** *inst_dir*`/SUNWics5/bin/config` **directory.**
6. **Edit the** `ics.conf` **file, and change the following line**:

```
service.http.ipsecurity=no
```

7. **Stop, and start the calendar server with the following commands**:

```
# inst_dir/SUNWics5/bin/stop-cal
# inst_dir/SUNWics5/bin/start-cal
```

Exchange

You can run Exchange using the Netlet for Windows 95, 98, SE, ME. Windows 2000, and later releases, has reserved the port that the netlet uses to listen on and that the Outlook client is hard coded to use. Stopping the services on this port will prevent system-wide networking from functioning correctly.

In SP3 Hot Patch 1, the ability to use Outlook Web Access (OWA) instead was added so that remote users could still access their corporate mail without the use of the Outlook client. The rewriter rules necessary for this integration are pre-loaded into the gateway profile when the patch is installed. There is one remaining problem with OWA integration that has to do with the Internet Explorer client.

As described earlier, the `STYLE` attribute is not currently handled by the gateway. OWA uses DHTML defined by the CSS `behavior` attribute. Because the `behavior` attribute has a `URL` and the `STYLE` attribute itself is not handled any differently than other HTML tag attributes, the URL will not be rewritten correctly. Thus, there are some more advanced OWA actions, such as folder renaming, which will not work correctly using the Internet Explorer browser. The entire functionality has been determined to work correctly using the Netscape Communicator client.

In addition to running the Portal Server on at least SP3 Hot Patch 1, NTLM authentication will also need to be disabled on the IIS instance running the Web Exchange interface, as described in the following procedure.

▼ To Disable NTLM on the IIS Instance

1. **From the Control Panel, select the Internet Information Services icon.**
2. **Expand the default website icon.**
3. **Select Exchange (or whatever you have mapped OWA to use).**
4. **Click on the page icon with the right mouse button, and select Properties.**
5. **Choose Directory Security Tab.**
6. **Under Anonymous Access and Authentication Control, choose Edit.**
7. **Deselect Integrated Windows Authentication.**
8. **Select OK.**
9. **Restart the services.**

How to Get Hot Patches

Hot Patches are not publicly available. Therefore, they are given out on an as needed basis to customers with valid support contracts. There are two ways to open a support case to get a Hot Patch. You can call 877-838-7272 to log a case with technical support. Or, you can file a case at the following site:

```
http://cgi.iplanet.com/cgi-bin/cs/ct-newcase.cgi
```

Glossary

- Attribute—Used to identify logical groupings in the `iwtGateway` XML tagged text that maps directly to LDAP entries after the component containing the attributes is imported using the `ipsadmin` command.
- CSS—Stands for cascading style sheets. Used for styling purposes in HTML and XML.
- DMZ—Stands for demilitarized zone. A network term indicating the space behind a packet filtering firewall and outside of the corporate firewall.
- DTD—Stands for document type definition. A document containing the declaration of tag names and attributes that can be used by an accompanying XML file.
- Left side—Stands for left-hand side, and indicates the left-most portion of the assignment operator (=).
- Out-of-box—Refers to a general software configuration that has just been installed using the installer defaults or the state of the system just following installation with no customizations having been made.
- PCDATA—Stands for parsed character data. PCDATA represents text that does not contain special characters such as quotes, <, and >, referred to as markup.
- Raw URL—Refers to any string, string literal, or string object that by itself is clearly identifiable from a syntax perspective as being a URL. Strings that begin with a protocol identifier or prepended path information are usually raw URLs.
- Rewriter—Refers to a logical unit of the Portal Server gateway that is responsible for the translation of URLs so that all browser requests for Intranet content go through the gateway, rather than attempting to contact the Intranet content server directly.
- Right side—Stands for right-hand side, and indicates the right-most portion of the assignment operator (=).
- Rule set—Refers to all of the gateway profile attributes and their values that are related to URL rewriting.
- Section—Refers to the name in the gateway profile Administration page that maps to the corresponding profile attribute description field. A section name might be Rewrite JavaScript Variables in URLs.
- SRAP—Stands for Secure Remote Access Pack. In the 3.0 SP version of the product, SRAP is used interchangeably with the term gateway component.
- XML—Stands for Xtensible Markup Language. Offers users a way to define and control how to render their own tag set.

About the Author

Rob Baker has over five years of experience in Internet server software development, deployment, and usage. For the past three years, he has worked as a CTE-Sustaining Engineer on products including the Solaris ISP server software, Netscape Enterprise server software, the Netscape PublishingXpert software, and now the Sun ONE Portal Server software (formerly iPlanet Portal Server). In his current role as sustaining lead for the Sun ONE Portal Server, he continues to create and follow best practices for large scale Enterprise-level Portal Server deployments, addressing complex issues such as content aggregation, leveraging existing applications, and providing secure remote access to Enterprise resources through VPN-on-demand technology.

Acknowledgements

Thanks goes out to Patrick Petit and Alejandro Medrano for providing the recommendations in "Tuning Recommendations" on page 67.

Ordering Sun Documents

The SunDocs^SM program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is `http://docs.sun.com/`

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at: `http://www.sun.com/blueprints/online.html`