Url Rewrite Filter 3.2.0

Manual

Community support is available at urlrewrite google group.

Read **examples of usage** and a sample of the ant task report. If you have feedback, or conf you want to share with the world email me. If you have any suggestions/examples for this manual please post them to the group.

Install

1. Download the zip (or tar.gz) and extract it into your context's directory ie, so that urlrewrite.xml goes into the WEB-INF directory.
2. Add the following to your WEB-INF/web.xml (add it near the top above your servlet mappings (if you have any)): (see filter parameters for more options)

```
<filter>
    <filter-name>UrlRewriteFilter</filter-name>
    <filter-class>org.tuckey.web.filters.urlrewrite.UrlRewriteFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>UrlRewriteFilter</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

3. Add your own configuration to the WEB-INF/urlrewrite.xml that was created.
4. Restart the context.

You can visit http://127.0.0.1:8080/rewrite-status (or whatever the address of your local webapp and context) to see output (note: this page is only viewable from localhost).

Filter Parameters

There are a few advanced filter parameters for enabling conf file reloading etc. There are self-explanatory.

```
<filter>
    <filter-name>UrlRewriteFilter</filter-name>
    <filter-class>org.tuckey.web.filters.urlrewrite.UrlRewriteFilter</filter-class>

    <!-- set the amount of seconds the conf file will be checked for reload
    can be a valid integer (0 denotes check every time,
    -1 denotes no reload check, default -1) -->
    <init-param>
        <param-name>confReloadCheckInterval</param-name>
        <param-value>60</param-value>
    </init-param>

    <!-- if you need to the conf file path can be changed
    it is specified as a path relative to the root of your context
    (default /WEB-INF/urlrewrite.xml) -->
    <init-param>
        <param-name>confPath</param-name>
        <param-value>/WEB-INF/urlrewrite.xml</param-value>
    </init-param>

    <!-- sets up log level (will be logged to context log)
    can be: TRACE, DEBUG, INFO (default), WARN, ERROR, FATAL, log4j, commons, slf4j,
    sysout:{level} (ie, sysout:DEBUG)
    if you are having trouble using normal levels use sysout:DEBUG
    (default WARN) -->
    <init-param>
        <param-name>logLevel</param-name>
        <param-value>DEBUG</param-value>
    </init-param>

    <!-- you can change status path so that it does not
    conflict with your installed apps (note, defaults
    to /rewrite-status) note, must start with / -->
    <init-param>
        <param-name>statusPath</param-name>
        <param-value>/status</param-value>
    </init-param>

    <!-- you can disable status page if desired
    can be: true, false (default true) -->
    <init-param>
        <param-name>statusEnabled</param-name>
        <param-value>true</param-value>
    </init-param>

    <!-- you may want to allow more hosts to look at the status page
    statusEnabledOnHosts is a comma delimited list of hosts, * can
    be used as a wildcard (defaults to "localhost, local, 127.0.0.1") -->
```

```
            <init-param>
                <param-name>statusEnabledOnHosts</param-name>
                <param-value>localhost, dev.*.myco.com, *.uat.mycom.com</param-value>
            </init-param>

            <!-- you may want to allow more hosts to look at the status page
            statusEnabledOnHosts is a comma delimited list of hosts, * can
            be used as a wildcard (defaults to "localhost, local, 127.0.0.1") -->
            <init-param>
                <param-name>statusEnabledOnHosts</param-name>
                <param-value>localhost, dev.*.myco.com, *.uat.mycom.com</param-value>
            </init-param>

            <!-- defaults to false. use mod_rewrite style configuration file (if this is true and
            is not specified confPath will be set to /WEB-INF/.htaccess) -->
            <init-param>
                <param-name>modRewriteConf</param-name>
                <param-value>false</param-value>
            </init-param>

            <!-- load mod_rewrite style configuration from this parameter's value.
                    note, Setting this parameter will mean that all other conf parameters are igno
                <init-param>
                    <param-name>modRewriteConfText</param-name>
                    <param-value>
                        RewriteRule ^/~([^/]+)/?(.*) /u/$1/$2 [R]
                        RewriteRule ^/([uge])/([^/]+)$ /$1/$2/ [R]
                    </param-value>
                </init-param>
            -->

            <!-- defaults to false. allow conf file to be set by calling /rewrite-status/?conf=/WE
                    designed to be used for testing only
                <init-param>
                    <param-name>allowConfSwapViaHttp</param-name>
                    <param-value>false</param-value>
                </init-param>
            -->

        </filter>

        <filter-mapping>
            <filter-name>UrlRewriteFilter</filter-name>
            <url-pattern>/*</url-pattern>
            <dispatcher>REQUEST</dispatcher>
            <dispatcher>FORWARD</dispatcher>
        </filter-mapping>
```

Note, setting `logLevel` to `log4j` or `commons` will cause the built in loging to call either log4j or commons-logging as if they were the logging framework, obviously you will need to have the jar for log4j or commons-logging in your classpath.

Configuration File WEB-INF/urlrewrite.xml

Configuration is done via a simple XML file that lives in your `WEB-INF` folder. It should be named urlrewrite.xml. It may be helpful to read the UrlRewriteFilter DTD (Document Type Definition). Please also make sure you look at the examples. A simple configuration file looks like:

```
        <?xml version="1.0" encoding="utf-8"?>

        <!DOCTYPE urlrewrite
            PUBLIC "-//tuckey.org//DTD UrlRewrite 3.0//EN"
            "http://tuckey.org/res/dtds/urlrewrite3.0.dtd">

        <urlrewrite>

            <rule>
               <from>^/some/olddir/(.*)$</from>
               <to type="redirect">/very/newdir/$1</to>
            </rule>

            <rule match-type="wildcard">
               <from>/blog/archive/**</from>
               <to type="redirect">/roller/history/$1</to>
            </rule>

        </urlrewrite>
```

The urlrewrite.xml file must have a root element called "urlrewrite" and must contain at least one "rule" element.

A "rule" must contain a "from" and a "to", and can have zero or more "condition" elements and zero or more and/or "set" elements.

When a "rule" is processed against an incoming request, all the "condition" elements must be met, then the "from" will be applied to the request URL and the final URL generated by applying the "to" to the "from" pattern. So long as the rule has matched then the "set" will be run.

When executing a rule the filter will (very simplified) loop over all rules and for each do something like this psuedo code:

```
Pattern.compile(<from> element);
pattern.matcher(request url);
matcher.replaceAll(<to> element);
if ( <condition> elements match && matcher.find() ) {
    handle <set> elements (if any)
    execute <run> elements (if any)
    perform <to> element (if any)
}
```

<urlrewrite> element

The top level element.

Attribute	Possible Value	Explanation
default-match-type (optional)	**regex** (default)	All rules and thier conditions will be processed using the Java Regular Expression engine (unless `match-type` is specified on a rule).
	wildcard	All rules and thier conditions will be processed using the Wildcard Expression engine (unless `match-type` is specified on a rule).
decode-using (optional)	**header,utf8** (default)	When URL is decoded request.getCharacterEncoding() will be used, if that is empty UTF-8 will be used.
	null	Do not decode at all. (note, this means the literal string null e.g. decode-using="null")
	header	Only use request.getCharacterEncoding() to decode.
	[encoding]	Only use a specific character encoding eg, ISO-8859-1. See Java Charset Object for all character encodings.
	header, [encoding]	When URL is decoded request.getCharacterEncoding() will be used, if that is empty a specific character encoding eg, ISO-8859-1. See Java Charset Object for all character encodings.
use-query-string (optional)	**false** (default)	The query string will *not* be appended to the url that the "from" element matches against.
	true	The query string will be appended to the url that the "from" element matches against.
use-context (optional)	**false** (default)	The context path will *not* be added to the url that the "from" element matches against.
	true	The context path will be added to the url that the "from" element matches against.

<rule> element

Zero or more. The basis of a rule.

Attribute	Possible Value	Explanation
enabled (optional)	**true** (default)	Enable this rule.
	false	Disable this rule.
match-type (optional)	**regex** (default)	This rule and it's conditions will be processed using the Java Regular Expression engine.
	wildcard	This rule and it's conditions will be processed using the Wildcard Expression engine.

In the following example requests for `/world/usa/nyc` will be transparently forwarded to `/world.jsp`

```
<rule match-type="regex">
   <from>^/world/([a-z]+)/([a-z]+)$</from>
   <to>/world.jsp</to>
</rule>
```

```
<rule match-type="wildcard">
   <from>/world/*/*</from>
   <to>/world.jsp</to>
</rule>
```

<outbound-rule> element

Zero or more. This is very similar to a normal rule but it is used for rewriting urls that go through `response.encodeURL()`.

Attribute	Possible Value	Explanation

enabled (optional)	**true** (default)	Enable this rule.
	false	Disable this rule.
encodefirst (optional)	**false** (default)	Run encodeURL() **after** running this outbound rule.
	true	Run encodeURL() **before** running this outbound rule.

May contain "run", "from", "to" and "set" element(s) also. Example:

```
<outbound-rule>
    <from>^/world.jsp?country=([a-z]+)&amp;city=([a-z]+)$</from>
    <to>/world/$1/$2</to>
</outbound-rule>
```

Using the example above JSP's with the code
`<a href="<%= response.encodeURL("/world.jsp?country=usa&amp;city=nyc") %>">nyc</a>`
will output
`<a href="/world/usa/nyc">nyc</a>`

Or JSTL
`<a href="<c:url value="/world.jsp?country=${country}&amp;city=${city}" />">nyc</a>`
will output
`<a href="/world/usa/nyc">nyc</a>`

Note, If you are using JSTL (ie, <c:url) this will work also.

<name> element

An optional element used for documenting the name of the rule. This can be used with rule and outbound-rule. See ant task.

```
<rule>
    <name>World Rule</name>
    <from>^/world/([a-z]+)/([a-z]+)$</from>
    <to>/world.jsp?country=$1&amp;city=$2</to>
</rule>
```

<note> element

A simple optional element used for documentation of the rule. This can be used with rule and outbound-rule. See ant task.

```
<rule>
    <name>World Rule</name>
    <note>
        Cleanly redirect world requests to JSP,
        a country and city must be specified.
    </note>
    <from>^/world/([a-z]+)/([a-z]+)$</from>
    <to>/world.jsp</to>
</rule>
```

<condition> element

An element that lets you choose condtions for the rule. Note, all conditions must be met for the rule to be run (unless "next" is set to "or" obvoiusly).

Value can be any Regular Expression.

Attribute	Possible Value	Explanation
type (optional)	**header** (default)	If used, the header name must be specified in the "name" attribute.
	method	The method of the request. GET, POST, HEAD etc.
	port	The port that the web application server is running on.
	time	Current time at the server (this will be the number of seconds since 00:00:00 1970-01-01 UTC otherwise known as unix time). i.e. `(new Date()).getTime()` This can be used for making sure content goes live only at a time you set.
	year	Current year at the server. i.e. `(Calendar.getInstance()).get(Calendar.YEAR)`
	month	Month at the server. January is 0 i.e. `(Calendar.getInstance()).get(Calendar.MONTH)`
	dayofmonth	Day of the month at the server. March first is 1 i.e. `(Calendar.getInstance()).get (Calendar.DAY_OF_MONTH)`

dayofweek	Day of the week at the server. Saturday is 1, Sunday is 7 i.e. `(Calendar.getInstance()).get (Calendar.DAY_OF_WEEK)`
ampm	AM or PM time at the server. i.e. `(Calendar.getInstance()).get(Calendar.AM_PM)`
hourofday	The hour of the day (24 hour clock) at the server. 10pm is 22 i.e. `(Calendar.getInstance()).get (Calendar.HOUR_OF_DAY)`
minute	The minute field of the current time at the server. i.e. `(Calendar.getInstance()).get(Calendar.MINUTE)`
second	The second field of the current time at the server. i.e. `(Calendar.getInstance()).get(Calendar.SECOND)`
millisecond	The millisecond field of the current time at the server. i.e. `(Calendar.getInstance()).get (Calendar.MILLISECOND)`
attribute	Will check the value of a request attribute (don't confuse this with parameter!), `name` must be set when using this type. i.e. `request.getAttribute([name])`
auth-type	Will check the value of a request attribute (don't confuse this with parameter!) i.e. `request.getAuthType()`
character-encoding	The character encoding of the imcoming request. i.e. `request.getCharacterEncoding()`
content-length	The length of the imcoming request (can be useful if you want to deny large requests). i.e. `request.getContentLength()`
content-type	The type of the imcoming request. (this is probably not that useful) i.e. `request.getContentType()`
context-path	The context path of the imcoming request. i.e. `request.getContextPath()`
cookie	The value of a cookie, note, `name` must be specified to use this i.e. `request.getCookies()` the find we the one with [name] specified and check the value.
parameter	A tidier way of checking request parameters than looking for them in the query string. This will check for the parameter in GET or POST, note, `name` must be specified. i.e. `request.getParameter([name])`
path-info	i.e. `request.getPathInfo()`
path-translated	i.e. `request.getPathTranslated()`
protocol	The protocol used to make the request, e.g. HTTP/1.1 i.e. `request.getProtocol()`
query-string	The query string used to make the request (if any), e.g. id=2345&name=bob i.e. `request.getQueryString()`
remote-addr	The IP address of the host making the request, e.g. 123.123.123.12 i.e. `request.getRemoteAddr()`
remote-host	The host name of the host making the request, e.g. 123qw-dsl.att.com (note, this will only work if your app server is configured to lookup host names, most aren't). i.e. `request.getRemoteHost()`
remote-user	The login of the user making this request, if the user has been authenticated, e.g. bobt i.e. `request.getRemoteUser()`
requested-session-id	Returns the session ID specified by the client, e.g. 2344asd234sada4 i.e. `request.getRequestedSessionId()`
requested-session-id-from-cookie	Whether the requested session ID is from a cookie or not i.e. `request.isRequestedSessionIdFromCookie()`
requested-session-id-from-url	Whether the requested session ID is from the URL or not i.e. `request.isRequestedSessionIdFromURL()`
requested-session-id-valid	Whether the requested session ID is valid or not i.e. `request.isRequestedSessionIdValid()`
request-uri	Returns the part of this request's URL from the protocol name up to the query string in the first line of the HTTP request i.e. `request.getRequestURI()`
request-url	Reconstructs the URL the client used to make the request. The returned URL contains a protocol, server name, port number, and server path, but it does not include query

		string parameters. i.e. `request.getRequestURL()`
	session-attribute	(note, name must be set) i.e. `session.getAttribute([name])`
	session-isnew	Weather the session is new or not. i.e. `session.isNew()`
server-name	The host name of the server to which the request was sent (from the host header not the machine name). i.e. `request.getServerName()`	
scheme	The scheme used for the request, e.g. http or https i.e. `request.getScheme()`	
user-in-role	(Note, the value for this cannot be a regular expression) i.e. `request.isUserInRole([value])`	
name (optional)	(can be anything)	If type is header, this specifies the name of the HTTP header used to run the value against.
next (optional)	**and** (default)	The next "rule" **and** this "rule" must match.
	or	The next "rule" **or** this "condition" may match.
operator (optional)	**equal** (default)	Equals. The operator to be used when the condition is run, the regular expression matches or the values are equal.
	notequal	Not equal to. (i.e. request value != condition value). Note, this operator only work with numeric rule types.
	greater	Greater than. (i.e. request value > condition value). Note, this operator only work with numeric rule types.
	less	Less than. (i.e. request value < condition value). Note, this operator only work with numeric rule types.
	greaterorequal	Greater to or equal to. (i.e. request value >= condition value). Note, this operator only work with numeric rule types.
	lessorequal	Less than or equal to. (i.e. request value <= condition value). Note, this operator only work with numeric rule types.

Examples:

```
<condition name="user-agent" operator="notequal">Mozilla/[1-4]</condition>

<condition type="user-in-role" operator="notequal">bigboss</condition>

<condition name="host" operator="notequal">www.example.com</condition>

<condition type="method" next="or">PROPFIND</condition>
<condition type="method">PUT</condition>
```

<from> element

You must always have exactly one from for each rule or outbound-rule. Value can be a regular expression in the Perl5 style. Note, from url's are relative to the context.

Attribute	Possible Value	Explanation
casesensitive (optional)	false (default)	This value will be matched using case insentitive match. ie, "/WellingtoN" will match "/wellington".
	true	This value will be matched using case sentitive match. ie, "/aAa" will NOT match "/aaa".

Example:

```
<from>^/world/([a-z]+)$</from>
```

<to> element

Value can be a regular replacement expression in the Perl5 style.

Attribute	Possible Value	Explanation
type (optional)	forward (default)	Requests matching the "conditions" for this "rule", and the URL in the "from" element will be internally forwarded to the URL specified in the "to" element. Note: In this case the "to" URL must be in the same context as UrlRewriteFilter. This is the same as doing:

		`RequestDispatcher rq = request.getRequestDispatcher([to value]);` `rq.forward(request, response);`
	passthrough	Identical to "forward".
	redirect	Requests matching the "conditions" and the "from" for this rule will be HTTP redirected. This is the same a doing: `HttpServletResponse.sendRedirect([to value]))`
	permanent-redirect	The same as doing: `response.setStatus(HttpServletResponse.SC_MOVED_PERMANENTLY);` `response.setHeader("Location", [to value]);` (note, SC_MOVED_PERMANENTLY is HTTP status code 301)
	temporary-redirect	The same as doing: `response.setStatus(HttpServletResponse.SC_MOVED_TEMPORARILY);` `response.setHeader("Location", [to value]);` (note, SC_MOVED_TEMPORARILY is HTTP status code 302)
	pre-include	
	post-include	
	proxy	The request will be proxied to the full url specified. commons-http and commons-codec must both be in the classpath to use this feature.
last (optional)	false (default)	The rest of the "rules" will be processed if this one succeeds.
	true	No more "rules" will be processed if this one is a match.
encode (optional)	false (default if under rule)	response.encodeURL([to]) will be run on the to url before performing the rewrite.
	true (default if under outbound-rule)	response.encodeURL([to]) will NOT be called.

context (optional)		If your application server is configured to allow "cross context" communication then this attribute can be used to forward (and only forward, not redirect or other "to" types) requests to a named servlet context.
		
		On Tomcat, for instance, the application contexts in the server configuration (server.xml or context.xml) need the option crossContext="true". For instance, the two applications mentioned before ("app" and "forum") have to be defined as:
		
		`<Context docBase="app" path="/app" reloadable="true" crossContext="true"/>`
		`<Context docBase="forum" path="/forum" reloadable="true" crossContext="true"/>`

Note, "to" can be null ie, `<to>null</to>`, this will mean that the request will go no further if the rule is matched (ie, this filter will not call `chain.doFilter`).

If "to" is set to -, no substitution will take place and the request will go on like nothing happened (ie, this filter will call `chain.doFilter`).

```
<to>/world.jsp?country=$1</to>
```

To elements can contain backreferences and variables.

Backreferences

```
%N
```

Provides access to the grouped parts (parentheses) of the pattern from the last matched Condition in the current rule. N must be less than 10 and greater than 0 (i.e. %1, %2, %3 etc).

Variables

```
%{VARIABLE-NAME}
```

Any valid condition type can be used as a variable name. ie, '`%{port}`' will be translated to '`80`', '`%{year}`' to '`2005`', '`%{cookie:myCookie}`' would be translated to '`myCookieValue`' (assuming the user had a cookie named myCookie with the value myCookieValue).

Valid types are condition types, see condition for a full description.

Functions

```
${FUNCTION:PARAMS}
```

Functions can be places in `set` and `to` elements.

name	example	example returns
replace	`${replace:my cat is a blue cat:cat:dog}`	my dog is a blue dog
replaceFirst	`${replaceFirst:my cat is a blue cat:cat:dog}`	my cat is a blue dog

escape	`${escape:a b c}`	a+b+c
unescape	`${unescape:a+b+c}`	a b c
lower	`${lower:Hello World}`	hello world
upper	`${upper:hello}`	HELLO
trim	`${trim: abc def }`	abc def

<set> element

Allows you to set varous things if the rule is matched.

Attribute	Possible Value	Explanation
type (optional)	request (default)	The same as `request.setAttribute([name], [value])` (note, name must be set).
	session	The same as `request.getSesison(true).setAttribute([name], [value])` (note, name must be set).
	response-header	The same as `response.setHeader([name], [value])` (note, name must be set).
	cookie	Value can be in the format "[value][:domain[:lifetime[:path]]]". This sets a cookie on the client's browser. The cookie's name is specified by the name attribute. The domain field is the domain of the cookie, such as '.apache.org',the optional lifetime is the lifetime of the cookie in seconds, and the optional path is the path of the cookie (note, name must be set).
	status	The same as `response.setStatus([value])`
	content-type	The same as `response.setContentType([value])`
	charset	The same as `response.setCharacterEncoding([value])`
	expires	Will set the Expires HTTP header by adding the time specified and current time (this is mod_expires style). Syntax "{num type}*". Units can be (singular or plural); years, months, weeks, days, hours, minutes, seconds. eg, "1 day 2 seconds", "3 hours", "1 year 1 hour"
	locale	The same as `response.setLocale([value])` specify the Locale in the format (valid locales are, zh, zh-CN, zh-CN-southern i.e. "-" separating the language, country and variant (if any)).
	parameter	Enables you to override a `request.getParameter(String)` wuth a custom value
	method	Enables you to override `request.getMethod()` with a custom value
name (optional)	(can be anything)	If type is request, session, response-header, cookie this specifies the name item.

In the following example a request attribute "client" will be set to "AvantGo" or "Samsung SCH-6100", this can be fetched in a servlet or JSP using `request.getAttribute("client")`.

```
<rule>
    <condition name="user-agent">Mozilla/3\.0 (compatible; AvantGo .*)</from>
    <from>.*</from>
    <set name="client">AvantGo</set>
</rule>
<rule>
    <condition name="user-agent">UP\.Browser/3.*SC03 .* </from>
    <from>.*</from>
    <set name="client">Samsung SCH-6100</set>
</rule>
```

It is also possible to use regular replacement expressions as part of the value similar to their usage in <to> elements:

```
<rule>
    <from>/products/(.*)/(.*)/index.html</from>
    <set name="urlrewrite.product.slug">$1</set>
    <set name="urlrewrite.product.id">$2</set>
    <to>/products?slug=$1&id=$2</to>
</rule>
```

<run> element

Allows you to run a method on an object when a rule and it's conditions are matched.

Attribute	Possible value	Explanation
class		The class you want to run a method on. Must be a fully qualified name.
method (optional)	run (default)	The method you want to run, the method must have the parameters (HttpServletRequest, HttpServletResponse) e.g. `run(HttpServletRequest request, HttpServletResponse response)`

		Note, if `init(ServletConfig)` or `destroy()` is found they will be run at when creating or destroying an instance.
neweachtime (optional)	false (default)	One instance for each UrlRewriteFilter instance.
	true	A new instance of the class will be created before running each time set to true.

When the rule in the following example is matched, `WorldServlet.goGet(HttpServletRequest, HttpServletResponse)` will be invoked, the request will then be forwarded to `/world-presentation.jsp`.

```
        <rule>
            <from>^/world/[a-z]+/[a-z]+$</from>
            <run class="com.blah.web.WorldServlet" method="doGet" />
            <to>/world-presentation.jsp</to>
        </rule>
```

Note, you can specify init-param's the same way you would for a servlet.

```
        <run class="com.blah.web.MyServlet" method="doGet">
            <init-param>
                <param-name>someParamName</param-name>
                <param-value>10</param-value>
            </init-param>
        </run>
```

If the method being called throws an Exception the original exception will be re-thrown as if it were the original if it extends RuntimeException (eg, NullPointer), other exceptions are wrapped in a ServletException and thrown so your container can handle them.

<class-rule> element

Allows you to run a method every time a request come in for 100% dynamic rules. See the org.tuckey.web.filters.urlrewrite.sample package for an example.

Attribute	Explanation
class	The class you want to run a method on. Must be a fully qualified name.
method (optional, default matches)	The method you want to run, the method must have the parameters (HttpServletRequest, HttpServletResponse) e.g. `run(HttpServletRequest request, HttpServletResponse response)` Note, if `init(ServletConfig)` or `destroy()` is found they will be run at when creating or destroying an instance.
last (optional, default true	If false more rules will be processed following this rule even if it is matched (so that a better match may be found).

Example:

```
        <class-rule class="com.blah.web.MyRuleClass" />
```

Tips

- When you want to put an "&" in a rule you must enter it as the XML entity "&amp;"
- For simplicity you might want to start all from's with a `^` and end them with a `$`.
  In regular expressions `^` specifies the start of the string and `$` specifies the end.
  ie, a request for `/my/url/path` will NOT match `<from>^/url/$</from>` but it will match `<from>/url/</from>`
- If using `<outbound-rule>` remember all urls in your code must be encoded e.g. `<a href="">my link</a>`
- Regular expressions are complex and a bit tricky at times, read regular expression syntax for Java.
- If you find regular expressions difficult use Wildcards.
- "Context" is important. If you have an app with the context "/myapp" and you request the url "/myapp/somefolder/somepage.jsp", the container tells UrlRewriteFilter that the url is "/somefolder/somepage.jsp". This can be confusing, but basically your rules and conditions should not contain the context path (it will be handled by the container).

Wildcard Matching Engine

The wildcard matching engine can be used instead of regex. It is supported in conditions and rules where `match-type` is set to `wildcard` (or default-match-type is set on the urlrewrite element

e.g. `/big/url/*` will match `/big/url/abc.html` but will NOT match `/big/url/abc/dir/` or `/big/url/abc/`.

`/big/url/**` will match `/big/url/abc.html`, `/big/url/abc/dir/` and `/big/url/abc/`.

You can also use Regular expression style variable replacement, each match of a `*` will be available for use in `to` and `set` elements using simple `$1` `$2` variables.

e.g. `/my/big/url/*` will match `/my/big/url/abc.html` and `$1` will be set to `abc.html`.

Added in 3.0

Ant Task

An Ant task has been written to allow validate the conf file and generation of documentation. You can view a sample.

Paste the following into your build.xml file, then change the `dest` and `conf` to point to the correct places. Note, the urlrewrite jar file will need to be in your classpath.

```
<target name="urlrewrite-doc" depends="compile"
    description="UrlRewriteFilter validation and documenting">

<taskdef name="urlrewritedoc" classpath="lib/urlrewrite-3.2.0.jar"
    classname="org.tuckey.web.filters.urlrewrite.UrlRewriteDocTask" />
<urlrewritedoc
    conf="${build.home}/WEB-INF/urlrewrite.xml"
    dest="urlrewrite-conf-overview.html" />
</target>
```

mod_rewrite Style Configuration

Sample web.xml snippet:

```
<filter>
    <filter-name>UrlRewriteFilter</filter-name>
    <filter-class>org.tuckey.web.filters.urlrewrite.UrlRewriteFilter</filter-class>

    <!-- defaults to false. use mod_rewrite style configuration file (if this is true and
    is not specified confPath will be set to /WEB-INF/.htaccess) -->
    <init-param>
        <param-name>modRewriteConfText</param-name>
        <param-value><![CDATA[

            # redirect mozilla to another area
            RewriteCond  %{HTTP_USER_AGENT}  ^Mozilla.*
            RewriteRule  ^/no-moz-here$                /homepage.max.html  [L]

        ]]></param-value>
    </init-param>

</filter>

<filter-mapping>
    <filter-name>UrlRewriteFilter</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

OR alternately set modRewriteConf to true in filter parameters and add a `WEB-INF/.htaccess` file with your mod_rewrite style configuration in it.

```
<filter>
    <filter-name>UrlRewriteFilter</filter-name>
    <filter-class>org.tuckey.web.filters.urlrewrite.UrlRewriteFilter</filter-class>

    <!-- defaults to false. use mod_rewrite style configuration file (if this is true and
    is not specified confPath will be set to /WEB-INF/.htaccess) -->
    <init-param>
        <param-name>modRewriteConf</param-name>
        <param-value>true</param-value>
    </init-param>

</filter>

<filter-mapping>
    <filter-name>UrlRewriteFilter</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

Sample: WEB-INF/.htaccess

```
# redirect mozilla to another area
RewriteCond  %{HTTP_USER_AGENT}  ^Mozilla.*
RewriteRule  ^/no-moz-here$                /homepage.max.html  [L]
```

Documentation for the original mod_rewrite library mostly applies, differences are documented below.

Attribute	Explanation
RewriteLogLevel	Specified as int, trasnlated as: <= 1 - FATAL, 2 - ERROR, 3 - INFO, 4 - WARN, >= 5 DEBUG
RewriteLog	SYSOUT, SYSERR, log4j, commons (if not set context logging will be used)

RewriteRule	Supported but note: • Proxy flag [P] supported if commons-httpclient and commons-codec in the classpath Certain flags not supported: • chain flag [C] not supported • env flag [E] not supported • next flag [N] not supported • nosubreq flag [NS] not supported • qsappend flag [QSA] not supported • Skip flag [S] not supported
RewriteBase	Not supported
RewriteLock	Not supported
RewriteMap	Not supported
RewriteOptions	Not supported