Support **Documentation**

NETSCAPE

Navigator Proxy Auto-Config File Format

March 1996

(There are several examples and tips in the end of this document)

The proxy autoconfig file is written in JavaScript. The file must define the function:

```
function FindProxyForURL(url, host)
{
    ...
}
```

which will be called by the Navigator in the following way for every URL that is retrieved by it:

```
ret = FindProxyForURL(url, host);
```

where:

`url`   the full URL being accessed.

`host`   the hostname extracted from the URL. This is only for convenience, it is the exact same string as between `://` and the first `:` or `/` after that. The port number is not included in this parameter. It can be extracted from the URL when necessary.

`ret`   (the return value) a string describing the configuration. The format of this string is defined below.

Saving the Auto-Config File
Setting the MIME Type

1. You should save the JavaScript function to file with a `.pac` filename extension; for example:

   ```
   proxy.pac
   ```

   Note 1: You should save the JavaScript function **by itself**, not embed it in HTML.

   Note 2: The examples in the end of this document are **complete**, there is no additional syntax needed to save it into a file and use it (of course, the JavaScripts have to be edited to reflect your site's domain name and/or subnets).

2. Next, you should configure your server to map the `.pac` filename extension to the MIME type:

   ```
   application/x-ns-proxy-autoconfig
   ```

   If using a Netscape server, edit the `mime.types` file in the `config` directory. If using Apache, CERN or NCSA servers, use the `AddType` directive.

Return Value Format

The JavaScript function returns a single string.

If the string is null, no proxies should be used.

The string can contain any number of the following building blocks, separated by a semicolon:

`DIRECT`
>   Connections should be made directly, without any proxies.

`PROXY` *`host:port`*
>   The specified proxy should be used.

`SOCKS` *`host:port`*
>   The specified SOCKS server should be used.

If there are multiple semicolon-separated settings, the left-most setting will be used, until the Navigator fails to establish the connection to the proxy. In that case the next value will be used, etc.

The Navigator will automatically retry a previously unresponsive proxy after 30 minutes, then after 1 hour from the previous try (always adding an extra 30 minutes).

If all proxies are down, and there was no `DIRECT` option specified, the Navigator will ask if proxies should be temporarily ignored, and direct connections attempted. The Navigator will ask if proxies should be retried after 20 minutes has passed (then the next time 40 minutes from the previous question, always adding 20 minutes).

Examples:

`PROXY w3proxy.netscape.com:8080; PROXY mozilla.netscape.com:8081`
>   Primary proxy is `w3proxy:8080`; if that goes down start using `mozilla:8081` until the primary proxy comes up again.

`PROXY w3proxy.netscape.com:8080; PROXY mozilla.netscape.com:8081; DIRECT`
>   Same as above, but if both proxies go down, automatically start making direct connections. (In the first example above, Netscape will ask user confirmation about making direct connections; in this third case, there is no user intervention.)

`PROXY w3proxy.netscape.com:8080; SOCKS socks:1080`
>   Use SOCKS if the primary proxy goes down.

Predefined Functions and Environment for the JavaScript Function

- Hostname based conditions:
    - isPlainHostName()
    - dnsDomainIs()
    - localHostOrDomainIs()
    - isResolvable()
    - isInNet()
- Related utility functions:
    - dnsResolve()
    - myIpAddress()
    - dnsDomainLevels()
- URL/hostname based conditions:
    - shExpMatch()
- Time based conditions:
    - weekdayRange()
    - dateRange()
    - timeRange()
- There is one associative array already defined (because a JavaScript currently cannot define them on its own):
    - ProxyConfig.bindings

`isPlainHostName(host)`

`host` the hostname from the URL (excluding port number).

True iff there is no domain name in the hostname (no dots).

Examples:

```
isPlainHostName("www")
      is true.
isPlainHostName("www.netscape.com")
      is false.
```

`dnsDomainIs(host, domain)`

`host` is the hostname from the URL.
`domain`
      is the domain name to test the hostname against.

Returns true iff the domain of hostname matches.

Examples:

```
dnsDomainIs("www.netscape.com", ".netscape.com")
      is true.
dnsDomainIs("www", ".netscape.com")
      is false.
dnsDomainIs("www.mcom.com", ".netscape.com")
      is false.
```

`localHostOrDomainIs(host, hostdom)`

`host` the hostname from the URL.
`hostdom`
      fully qualified hostname to match against.

Is true if the hostname matches exactly the specified hostname, or if there is no domain name part in the hostname, but the unqualified hostname matches.

Examples:

```
localHostOrDomainIs("www.netscape.com", "www.netscape.com")
      is true (exact match).
localHostOrDomainIs("www", "www.netscape.com")
      is true (hostname match, domain not specified).
localHostOrDomainIs("www.mcom.com", "www.netscape.com")
      is false (domain name mismatch).
localHostOrDomainIs("home.netscape.com", "www.netscape.com")
      is false (hostname mismatch).
```

---

isResolvable(host)

`host` is the hostname from the URL.

Tries to resolve the hostname. Returns true if succeeds.

Examples:

```
isResolvable("www.netscape.com")
```
is true (unless DNS fails to resolve it due to a firewall or some other reason).
```
isResolvable("bogus.domain.foobar")
```
is false.

---

isInNet(host, pattern, mask)

`host` a DNS hostname, or IP address. If a hostname is passed, it will be resoved into an IP address by this function.
`pattern`
an IP address pattern in the dot-separated format
`mask` mask for the IP address pattern informing which parts of the IP address should be matched against. 0 means ignore, 255 means match.

True iff the IP address of the host matches the specified IP address pattern.

Pattern and mask specification is done the same way as for SOCKS configuration.

Examples:

```
isInNet(host, "198.95.249.79", "255.255.255.255")
```
is true iff the IP address of host matches exactly 198.95.249.79.
```
isInNet(host, "198.95.0.0", "255.255.0.0")
```
is true iff the IP address of the host matches 198.95.*.*.

---

dnsResolve(host)

`host` hostname to resolve

Resolves the given DNS hostname into an IP address, and returns it in the dot separated format as a string.

Example:

```
dnsResolve("home.netscape.com")
```
returns the string "198.95.249.79".

---

myIpAddress()

Returns the IP address of the host that the Navigator is running on, as a string in the dot-separated integer format.

Example:

```
myIpAddress()
```
would return the string `"198.95.249.79"` if you were running the Navigator on that host.

`dnsDomainLevels(host)`

`host` is the hostname from the URL.

Returns the number (integer) of DNS domain levels (number of dots) in the hostname.

Examples:

```
dnsDomainLevels("www")
```
returns 0.
```
dnsDomainLevels("www.netscape.com")
```
returns 2.

`shExpMatch(str, shexp)`

`str`   is any string to compare (e.g. the URL, or the hostname).
`shexp`
is a shell expression to compare against.

Returns true if the string matches the specified shell expression.

Actually, currently the patterns are *shell expressions*, not regular expressions.

Examples:

```
shExpMatch("http://home.netscape.com/people/ari/index.html", "*/ari/*")
```
is true.
```
shExpMatch("http://home.netscape.com/people/montulli/index.html", "*/ari/*")
```
is false.

`weekdayRange(wd1, wd2, gmt)`

`wd1`   and
`wd2`   are one of the weekday strings:

```
        SUN MON TUE WED THU FRI SAT
```

`gmt`   is either the string: `GMT` or is left out.

Only the first parameter is mandatory. Either the second, the third, or both may be left out.

If only one parameter is present, the function yeilds a true value on the weekday that the parameter represents. If the string `"GMT"` is specified as a second parameter, times are taken to be in GMT, otherwise in local timezone.

If both `wd1` and `wd1` are defined, the condition is true if the current weekday is in between those two weekdays. Bounds are inclusive. If the `"GMT"` parameter is specified, times are taken to be in GMT, otherwise the local timezone is used.

Examples:

```
weekdayRange("MON", "FRI")
```
> true Monday trhough Friday (local timezone).
```
weekdayRange("MON", "FRI", "GMT")
```
> same as above, but GMT timezone.
```
weekdayRange("SAT")
```
> true on Saturdays local time.
```
weekdayRange("SAT", "GMT")
```
> true on Saturdays GMT time.
```
weekdayRange("FRI", "MON")
```
> true Friday through Monday (note, order does matter!).

---

dateRange(day)
dateRange(day1, day2)
dateRange(mon)
dateRange(month1, month2)
dateRange(year)
dateRange(year1, year2)
dateRange(day1, month1, day2, month2)
dateRange(month1, year1, month2, year2)
dateRange(day1, month1, year1, day2, month2, year2)
dateRange(day1, month1, year1, day2, month2, year2, gmt)

`day`   is the day of month between 1 and 31 (as an integer).
`month`
>   is one of the month strings:

>   JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC

`year`   is the full year number, for example 1995 (but **not** 95). Integer.
`gmt`   is either the string `"GMT"`, which makes time comparison occur in GMT timezone; if left unspecified, times are taken to be in the local timezone.

>   Even though the above examples don't show, the `"GMT"` parameter can be specified in any of the 9 different call profiles, always as the last parameter.

If only a single value is specified (from each category: day, month, year), the function returns a true value only on days that match that specification. If both values are specified, the result is true between those times, including bounds.

Examples:

```
dateRange(1)
```
> true on the first day of each month, local timezone.
```
dateRange(1, "GMT")
```
> true on the first day of each month, GMT timezone.

```
dateRange(1, 15)
```
    true on the first half of each month.
```
dateRange(24, "DEC")
```
    true on 24th of December each year.
```
dateRange(24, "DEC", 1995)
```
    true on 24th of December, 1995.
```
dateRange("JAN", "MAR")
```
    true on the first quarter of the year.
```
dateRange(1, "JUN", 15, "AUG")
```
    true from June 1st until August 15th, each year (including June 1st and August 15th).
```
dateRange(1, "JUN", 15, 1995, "AUG", 1995)
```
    true from June 1st, 1995, until August 15th, same year.
```
dateRange("OCT", 1995, "MAR", 1996)
```
    true from October 1995 until March 1996 (including the entire month of October 1995 and March 1996).
```
dateRange(1995)
```
    true during the entire year 1995.
```
dateRange(1995, 1997)
```
    true from beginning of year 1995 until the end of year 1997.

---

timeRange(hour)
timeRange(hour1, hour2)
timeRange(hour1, min1, hour2, min2)
timeRange(hour1, min1, sec1, hour2, min2, sec2)
timeRange(hour1, min1, sec1, hour2, min2, sec2, gmt)

`hour` is the hour from 0 to 23. (0 is midnight, 23 is 11 pm.)
`min`   minutes from 0 to 59.
`sec`   seconds from 0 to 59.
`gmt`   either the string `"GMT"` for GMT timezone, or not specified, for local timezone. Again, even though the above list
        doesn't show it, this parameter may be present in each of the different parameter profiles, always as the last parameter.

True during (or between) the specified time(s).

Examples:

```
timerange(12)
```
    true from noon to 1pm.
```
timerange(12, 13)
```
    same as above.
```
timerange(12, "GMT")
```
    true from noon to 1pm, in GMT timezone.
```
timerange(9, 17)
```
    true from 9am to 5pm.
```
timerange(8, 30, 17, 00)
```
    true from 8:30am to 5:00pm.
```
timerange(0, 0, 0, 0, 0, 30)
```
    true between midnight and 30 seconds past midnight.

Example #1: Use proxy for everything except local hosts

This would work in Netscape's environment. All hosts which aren't fully qualified, or the ones that are in local domain, will be connected to directly. Everything else will go through `w3proxy:8080`. If the proxy goes down, connections become automatically direct.

```
function FindProxyForURL(url, host)
{
    if (isPlainHostName(host) ||
        dnsDomainIs(host, ".netscape.com"))
        return "DIRECT";
    else
        return "PROXY w3proxy.netscape.com:8080; DIRECT";
}
```

Note: This is the simplest and most efficient autoconfig file for cases where there's only one proxy.

Example #1b: As above, but use proxy for local servers which are outside the firewall

If there are hosts (such as the main Web server) that belong to the local domain but are outside the firewall, and are only reachable through the proxy server, those exceptions can be handled using the `localHostOrDomainIs()` function:

```
function FindProxyForURL(url, host)
{
    if ((isPlainHostName(host) ||
         dnsDomainIs(host, ".netscape.com")) &&
        !localHostOrDomainIs(host, "www.netscape.com") &&
        !localHostOrDoaminIs(host, "merchant.netscape.com"))

        return "DIRECT";
    else
        return "PROXY w3proxy.netscape.com:8080; DIRECT";
}
```

The above will use the proxy for everything else except local hosts in the `netscape.com` domain, with the further exception that hosts `www.netscape.com` and `merchant.netscape.com` will go through the proxy.

Note the order of the above exceptions for efficiency: localHostOrDomainIs() functions only get executed for URLs that are in local domain, not for every URL. Be careful to note the parentheses around the *or* expression before the *and* expression to achieve the abovementioned efficient behaviour.

Example #2: Use proxy only if cannot resolve host

This example would work in an environment where internal DNS is set up so that it can only resolve internal host names, and the goal is to use a proxy only for hosts which aren't resolvable:

```
function FindProxyForURL(url, host)
{
    if (isResolvable(host))
        return "DIRECT";
    else
        return "PROXY proxy.mydomain.com:8080";
}
```

The above requires consulting the DNS every time; it can be grouped smartly with other rules so that DNS is consulted only if other rules do not yield a result:

```
function FindProxyForURL(url, host)
{
    if (isPlainHostName(host) ||
        dnsDomainIs(host, ".mydomain.com") ||
        isResolvable(host))
        return "DIRECT";
    else
        return "PROXY proxy.mydomain.com:8080";
}
```

Example #3: Subnet based decisions

In this example all the hosts in a given subnet are connected to directly, others through the proxy.

```
function FindProxyForURL(url, host)
{
    if (isInNet(host, "198.95.0.0", "255.255.0.0"))
        return "DIRECT";
    else
        return "PROXY proxy.mydomain.com:8080";
}
```

Again, use of DNS in the above can be minimized by adding redundant rules in the beginning:

```
function FindProxyForURL(url, host)
{
    if (isPlainHostName(host) ||
        dnsDomainIs(host, ".mydomain.com") ||
        isInNet(host, "198.95.0.0", "255.255.0.0"))
        return "DIRECT";
    else
        return "PROXY proxy.mydomain.com:8080";
}
```

Example #4: Load balancing/routing based on URL patterns

This example is more sophisticated. There are four (4) proxy servers; one of them is a hot stand-by for all of the other ones, so if any of the remaining three goes down, the fourth one will take over.

Furthermore, the three remaining proxy servers share the load based on URL patterns, which makes their caching more effective (there is only one copy of any document on the three servers -- as opposed to one copy on each of them). The load is distributed like this:

Proxy	Purpose
#1	`.com` domain
#2	`.edu` domain
#3	all other domains
#4	hot stand-by

All local accesses are desired to be direct. All proxy servers run on the port 8080 (they wouldn't need to). Note how strings can be concatenated by the + operator in JavaScript.

```
function FindProxyForURL(url, host)
{
    if (isPlainHostName(host) || dnsDomainIs(host, ".mydomain.com"))
        return "DIRECT";
    else if (shExpMatch(host, "*.com"))
        return "PROXY proxy1.mydomain.com:8080; " +
               "PROXY proxy4.mydomain.com:8080";
    else if (shExpMatch(host, "*.edu"))
        return "PROXY proxy2.mydomain.com:8080; " +
               "PROXY proxy4.mydomain.com:8080";
    else
        return "PROXY proxy3.mydomain.com:8080; " +
               "PROXY proxy4.mydomain.com:8080";
}
```

Example #5: Setting a proxy for a specific protocol

Most of the standard JavaScript functionality is available for use in the `FindProxyForURL()` function. As an example, to set different proxies based on the protocol, the `substring()` function can be used:

```
function FindProxyForURL(url, host)
{
    if (url.substring(0, 5) == "http:") {

        return "PROXY http-proxy.mydomain.com:8080";
    }
    else if (url.substring(0, 4) == "ftp:") {

        return "PROXY ftp-proxy.mydomain.com:8080";
    }
    else if (url.substring(0, 7) == "gopher:") {

        return "PROXY gopher-proxy.mydomain.com:8080";
    }
    else if (url.substring(0, 6) == "https:" ||
            url.substring(0, 6) == "snews:") {

        return "PROXY security-proxy.mydomain.com:8080";
    }
    else {

        return "DIRECT";
    }
}
```

Note: The same can be accomplished using the `shExpMatch()` function described earlier; for example:

```
...
if (shExpMatch(url, "http:*")) {
    return "PROXY http-proxy.mydomain.com:8080;
}
...
```

Tips

- The autoconfig file can be output by a CGI script. This is useful e.g. when making the autoconfig file act differently based on the client IP address (the `REMOTE_ADDR` environment variable in CGI).
- Use of `isInNet()`, `isResolvable()` and `dnsResolve()` functions should be carefully considered, as they require DNS server to be consulted (whereas all other autoconfig related functions are mere string matching functions). If a proxy is used, the proxy will perform its own DNS lookup which would double the impact on the DNS server. Most of the time these functions are not necessary to achieve the desired result.

March 1996