

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

There are no translations available.

I have spent seven years designing, deploying and supporting proxies and corresponding PAC files in many Stream Sites. In those years, I have learned a number of hard lessons and gradually refined my PAC files to become (what I feel to be) well organized and highly efficient. I always had to create my PAC file structures by hand and never had any documents to refer to beyond the basic docs provided by a few vendors. One of the things that frustrated me most was the lack of good, practical advice online. There is a lot of basic material out there, but nothing that was written with a practical, engineering-focused mindset.

This guide is an attempt to wrap up many of the things that I've learned to make it easier for others responsible for designing, deploying and maintaining PAC files.

The content of this site is designed to expand upon the original guide published by Netscape and provide some more advanced examples and some fairly critical warnings. It is assumed that the serious reader has first read through the Netscape document and understands the basic functions listed there before trying to process the content here. There are a lot of functions listed that I don't use, simply because I haven't ever had the need. (i.e. time-based calculations), but I do use a lot of them, plus a few other tips and tricks from JavaScript that I've picked up over the years.

JavaScript is, of course, a complex programming language. It is tremendously flexible and can provide a lot of functionality. I am, however, NOT a JavaScript developer and won't ever be. The target audience of this guide are engineers like myself with little or no formal development background. If you're a JavaScript developer you might find some useful tricks and tips here on interaction with proxies but you probably won't be very interested in the sections that are more focused on the language itself.

Table of Content

- [PAC files - An Introduction](#)
- [Why should I use a PAC file?](#)

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

- [PAC files and Mobile Computing](#)
- [Designing a PAC file](#)
- [PAC file Basics](#)
- [Details on selected Functions](#)
- [PAC File Tricks and Tips](#)
- [Lessons Learned](#)
- [Testing and Troubleshooting](#)
- [Optimizing a PAC file](#)
- [Simple PAC file](#)
- [Simple PAC file with loads of Balancing](#)
- [Complex PAC file](#)
- References

PAC files - An Introduction

A Proxy Automatic Configuration (PAC) file is that acts as a URL/site routing control file for a web browser. Simply put: The PAC controls how the browser retrieves an object— directly, bypassing the proxy or via a proxy. The PAC can also return different proxy servers based on conditions found in the file.

PAC file are most commonly used in corporate environments with complex web environments. Chances are, if your company has a proxy it has a PAC file. PAC files can also be used in the home to block specific unwanted web content - Banner pages, malware, etc.

PAC files are written in the JavaScript programming language. Many common JavaScript functions can be used when developing a PAC file.

PAC files are the heart and soul of a large corporate network. They control the browser. A well-written and carefully-planned PAC file can be simple and efficient, yet offer tremendous flexibility. An organically-grown PAC file, made of a number of thrown-together statements might work, but can result in less efficient processing and significantly more network bandwidth for distribution.

Why should I use a PAC file?

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

Using a PAC file is a very good way to manage a complex proxy infrastructure and how it integrates with the desktop. Most companies start with simple browser exceptions and a hard-set proxy server and rapidly outgrow it. Corporate networks are large, complex entities that have complex traffic flows between local networks, directly-connected vendor/partner networks and the Internet. Often DNS domains are split between internal and Internet hosts which require complex routing and acquisition of other companies can add another entire level of complexity.

With a PAC file, you have full if/and/or logic and the ability to structure your queries in whatever order you want. This becomes especially important if you have hosts in your local domain that must be accessed via your proxy. For example, `www.company.com` is on the Internet and must traverse the proxy while the rest of "company.com" is internal and doesn't need to be proxied. This is nearly impossible to fix with manual browser settings which send all traffic to the proxy and only permit exceptions which are sent browser direct. The first time you get a host in a domain that needs to be proxied, everything else in that domain becomes an exception and becomes impossible to manage without using a PAC file.

When a browser is set with a PAC file it pulls that file from a central server. This means that the file centrally controlled and managed rather than distributed on multiple PC's or servers. When a change must be made to browser routing, the PAC file can be centrally updated and the browsers will retrieve the new copy the next time they launch. No more managing browser exceptions via group policy or trying to explain to your users how to manually add a browser proxy exception to fix a problem.

Another benefit of a PAC file is for laptops. If you have hard-set a proxy and you connect a laptop into the Internet it won't work until you manually reconfigure the browser. A browser configured with a PAC file should work equally well on the corporate network or directly on the Internet.

PAC files and Mobile Computing

I've seen a lot of questions about using PAC files and proxy settings on laptops / mobile computers which are often used outside of the typical corporate network. How do they behave?

Keep in mind that the default behavior for Internet Explorer (and most other browsers) is to try to load the PAC file as configured in the browser settings. If that PAC file can't be loaded (or contains an error) the browser will default to retrieving everything directly without using a proxy.

When a laptop user is on the corporate network the browser will pull a PAC file and follow the rules laid out in it. When that browser is on another network or the Internet it will not be able to load the configured PAC file and will default to browser direct.

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

(This, of course, assumes that you are following good security practices and not hosting your PAC file on a place where it could be retrieved from the Internet. PAC files contain data about internal IP addresses, traffic flows and hostnames – Not something that you want to be available on the Internet.)

If you are trying to load a PAC file as a file on the PC this, of course, won't work – The PAC file will always be accessible. This is, obviously, not recommended. If you must do this, be sure to check if the workstation's IP address is on one of your corporate subnets before trying to return a proxy, else return direct.

Pulling a PAC file using WPAD works in a very similar manner to a traditionally-configured PAC file. When the laptop is on the local network it will get the PAC from WPAD via DHCP or the default WPAD DNS name. When the machine is on the Internet neither of these methods will work and the browser will default to direct. Just keep in mind that if you plug into another company's network who uses WPAD your machine will download and apply those settings automatically.

Designing a PAC file

Designing a PAC file is like any other project - Start with a high-level plan of how HTTP traffic should flow in your network. Sketch it out on paper or in Notepad before actually trying to write the code. Work on the basic defaults, general rules, the exceptions to those rules and then any special overrides. Once you have those planned, add in the rest of the "trimmings" and you should end up with a design for a well-structured PAC file.

It is very important to remember that PAC files are parsed iteratively from top to bottom, once per host. The first time the browser sees a RETURN statement it stops processing and uses that response. An easy way to ensure that your PAC file will process properly is to design it from the bottom up, starting with the most general and working your way up to the most specific at the top.

When putting together the structure, keep in mind that your PAC file will be expanded as time goes on. Websites and business partners will be added, new domains will be deployed, etc. Make sure in your planning that it will be easy to perform these modifications.

Choosing a Default Action

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

The first question to ask yourself is what the default action should be - Do you want the browser to fetch a request through the proxy or direct? In 99.99% of the companies out there, using a proxy is the default. This, then, becomes the LAST thing that happens in your PAC file.

Defining General Rules

Now that you've worked out the default action you have to start working on the general rules. These are typically rules about domains that are hosted internal to your corporate network or can be accessed directly by your company's PC's. Let's say that you have an internal domain, "company.net" that is used internally. As a general rule, all requests to hosts in the .company.net should be retrieved directly by the browser.

Defining Exceptions

Working your way UP your PAC file design we come to the exceptions. Using the example from above, all hosts in .company.net are retrieved directly, as a general rule. But, let's say you have outsourced your e-mail and "webmail.company.net" is on the Internet. You would need to put in an exception ABOVE the general rules to send "webmail.company.net" via the proxy.

Other common exceptions might include a list of business partner sites which need to be retrieved directly through the browser or other hosts which would normally be sent to the Internet by the default action but actually need to be retrieved directly.

Special Overrides

Occasionally an administrator will run into times when special very specialized routing rules will be required. For example, a trading floor might have some dedicated routing to some of their service providers. Users on the trading floor subnets need to route some requests differently than the others. I consider this an override to all other rules. From a syntax perspective, this would be written logically as "If the client IP is in the special address range and the target is the specific domain, then return the proper value"

Adding the "Trimmings"

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

I've always considered the "trimmings" to be the parts of the PAC file which aren't involved in comparing the host. There's a number of things you can do in a PAC file - Checking the client's IP address to see which proxy it should use, blocking known spyware/adware, localhost overrides, utility informational pop-ups, etc.

Keeping it Simple

When I was a young engineer I had a great boss who gave me an excellent piece of advice - Keep it Simple and Supportable. In PAC files (and many other technology disciplines) there are many ways of accomplishing the same goal. When you build your PAC file you should do your best to build it in a straightforward manner that others can easily understand and support. I've seen some incredibly elegant PAC file code that is tight, clean and highly efficient but it would require a professional programmer to support. Assume that when you write a PAC file it needs to be understandable by all the people on your direct team plus others who might need to read it - Desktop support, server engineering, etc. You might also call this the "hit by the bus" rule - Write your PAC file so that someone else can easily take it over if you get run over by a bus while out to lunch.

Writing the PAC File

Once you have your PAC file designed it's time to start writing it. Before you dig in too deeply, I'd strongly suggest you read the "PAC File Basics" article on this site. This explains some of the basic structures and details you'll need to write a proper PAC file.

There are a number of reference sections on this site to get more information about available functions and a number of pages with tips and tricks. Feel free to copy some of the code from the sample PAC files on this site - That's what they are there for. You can also use one of the examples as a base and modify it to fit your needs.

When you're actually starting to write your code you might consider just doing it in a text editor. There's a web-front end to the Google PAC Tester on this site, in the reference section. This should allow you to test your code via copy-and-paste as you go. I've found this simpler than hosting it on a web server or in a local file and needing to constantly mess with my browser settings. I only do that when I've run into something I can't work out and need to use alert().

If you're starting from scratch, write the code of your PAC file just like you designed it. Start with the basic framework and the default return. Load that in your browser and make sure it sends everything via the proxy. Once you've got that, add in your general rules above it and make sure those work. Don't be shy to throw in some Alert statements to make sure things are processing as you expect them to. Test everything, every step of the way. If something

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

doesn't work, you don't want to have to dig through 15 changes to find the one with the error.

As you're writing, plan for the future. A PAC file can be a very dynamic document. Try to structure things such that it is easy to expand. For example, when defining exceptions for .company.net, consider using one big IF statement with lots of OR statement for the various hosts/subdomains and one RETURN value at the end. Don't use individual IF statements with individual RETURNS. Avoid sprawl, keep it nice and clean.

Gradually all the pieces will fit together and you'll have a complete, working PAC file. Congratulations!

PAC File Basics

This section is designed to provide a bit of background on some of the more technical aspects of writing a PAC File.

As mentioned elsewhere, PAC files are written in JavaScript. JavaScript is a formal, structured programming language which has specific structures that must be followed when writing code. The goal of this section is to make you understand some of those rules and how they should be used.

This is an important document to read, but it's not going to teach you how to write JavaScript or even to develop perfect PAC files. This is a reference that you should review to understand some of the basics to be able to better interpret what you're seeing in some of the examples and other guides.

When I wrote this page I had a difficult time deciding what to call a "PAC File" component or a "JavaScript" component. Since PAC files are written in JavaScript there's nothing here that's unique to a PAC file and is generally useable in JavaScript. You might see me interchange the two terms occasionally.

JavaScript Variables

A variable in JavaScript is simply a way to assign name a value in a way that it can be easily reused later and referenced by the name.

For example, there's a JavaScript function called "myIPAddress()" which always returns the machine's local IP address. You can create a variable called "localIP" that contain the local

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

workstation IP address by using the following command.

```
LocalIP = myIPAddress();
```

Now, each time you need to check the machine's IP address you can use the variable "localIP" instead of having to call the myIpAddress() function. This is cheaper, computationally and easier to understand.

JavaScript Functions

JavaScript, like many other languages is built from a series of building blocks called "functions". A function is a JavaScript building block that executes a specific action or runs a specific test based on variables you send it.

JavaScript has a large number of built-in functions frequently used in PAC files. These functions help you perform tests like "Is this host resolvable in DNS?" "Is this host in this DNS domain?" "Is this IP address in this subnet?" "What is my IP address?" "What time is it?"

When you call a function you're going to need to give it some information. For example, "Is www.cnn.com in the cnn.com domain?" This is done by passing specific values or variables to the function and letting it process it.

Functions return one of two things - Either a specific value ("What is the local workstation IP address?") or a true/false answer ("Is www.cnn.com in the cnn.com domain?").

There are a number of basic documents from Netscape and Microsoft that list out the basic functions and what they do in the Reference section of this document, in the Base References section. There are also some more details on a few of those functions in the "Details on Selected Functions" section, as well.

Using Comments

Comments in JavaScript are very simple - Anytime JavaScript sees two forward slashes it ignores anything else in the line. This can be used to add notes or comments to the code to improve readability. Comments can be used in their own line or at the end of an existing line. For example.

```
// PAC file Date 7/1/2009, Version 2.33  
if (host == "intranet.company.com") { //Check for intranet.company.com
```


PAC Files - The First Line

Every PAC file must follow the same general structure which is pre-defined by the system. As mentioned elsewhere, a PAC file is written in JavaScript and can use many common Javascript functions and operands.

PAC files always begins with the same line

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

To decode this, a JavaScript function called "FindProxyForURL" is being created. Two variables are being passed to the function from the browser, "url" and "host".

The URL and HOST Variables

The "url" variable contains the full URL of the request - "http://www.yahoo.com/index.html". The "host" variable contains just the host name - "www.yahoo.com"

In general, you want to do almost all of your PAC file checks on the host and NOT on the URL. Internet Explorer has a "feature" called Automatic Proxy Results Cache, discussed in the "Lessons Learned" section of this site. This "feature" means IE will only process the PAC file ONCE per hostname per browser session. Because of this it is generally inadvisable to make proxy routing decisions based on a URL. There are some cases where this is appropriate, but be sure to think through the ramifications.

Return Statements

The basic role of a PAC file is to tell the browser if it should retrieve the site directly or if it should use a proxy. If the return is via proxy, the PAC file specifies which proxy should be used.

The way you do this is by using a "return". This tells the browser to stop processing the PAC file and gives the browser instructions on what to do.

Instructing your browser to retrieve the site directly, bypassing the proxy is simple. The return value is "DIRECT"

```
return "DIRECT";
```

Instructing the browser to use a proxy is also very simple. The return value is the word "PROXY" plus the proxy you want the browser to use. Either give the name or the IP address, a colon, then the port the proxy listens on.

```
return "PROXY proxy.company.com:8080";
```

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

If you use a name, the browser will accept a simple name "proxy" or a fully-qualified domain name (FQDN) "proxy.company.com". In general, it's better form to use the FQDN since it's a bit less work on the client PC and doesn't have any dependencies on the search path configuration of the PC.

You can also return more than one proxy server in the list, separated by a semi-colon. The browser will always start with the first one in the list and only move to the second one if it thinks the first has failed. This is, however, not foolproof - See the "Lessons Learned" section for more reading on this topic.

```
return "PROXY proxy1.company.com:8080; PROXY proxy2.company.com:8080";
```

I've seen a lot of questions around what you can return from a PAC file. The answer is a simple one - You can return DIRECT or one or more proxies. Nothing else. No alternate PAC files, etc.

PAC File Special Characters

This document isn't the place to get deep into the details of developing JavaScript, but you should understand how a number of special characters are used and why. For examples of each of these things, check the sample PAC files on this site.

Semicolons

A semicolon is used to end a statement or "action", for lack of a better term. Whenever you have line that does something it needs to end in a semicolon. Examples are below.

```
return "DIRECT";  
proxy = "PROXY nyproxy.company.com:8080";  
alert("I got here, something must be wrong below");
```

Semicolons should NOT be used after conditionals (IF statements).

Equals Signs

There are two very different ways that equals signs can be used. CHECKING a value or SETTING a value.

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

The most common thing to do with an equals sign is to check if two things are the same. To check a value, you use two equals signs together. For example "if (host == "www.company.com")" checks to see if the variable called "host" has the value of "www.company.com". Note that you can do a "Not equals" check by using the "!=" test - if (host != "www.company.com")

Using the equals to SET a value uses a single equals sign. For example, "localip = myIpAddress();" will set a variable called "localip" to have the value returned by the function myIpAddress.

Be careful not to confuse these two uses. A statement of "if (host = "www.company.com")" will always true - It is read as "IF I can successfully set the variable called "host" to the value of "www.company.com".

Carriage Returns, Extra Blank Spaces and Tabs

In general, JavaScript will ignore carriage returns/newline characters, extra blank spaces and tabs. The only time where these really make any difference is when they split apart something inside of quotes. This can be used to add formatting and increase readability. To a PAC file, the two examples below are processed identically.

Example 1:

```
if (host == "www.cnn.com" ||
    host == "images.cnn.com") {
    alert("Found CNN!");
    return "PROXY proxy.company.com:8080";
}
```

Example 2:

```
if (host=="www.cnn.com" || host == "images.cnn.com")
{ alert("Found CNN!"); return "PROXY proxy.company.com:8080"; }
```

Obviously, example #1 is far easier to read, but both will function identically.

Braces

Braces are used to group statements together and to signify a beginning and an end of a section of code. Braces ALWAYS come in pairs - Whenever you see an open brace "{" you should always be able to match it to a close brace - "}". Braces are most commonly used in

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

conditionals (IF statements).

When writing code, it's often recommended to indent the lines grouped together inside a set of braces. This makes it easier to follow.

It is, unfortunately, very easy to get mismatched braces. You might put an open brace in and forget to close it. This can lead to some odd results that are hard to troubleshoot, so check your braces carefully.

You'll notice the first line of the PAC file ends in an open brace. When building a PAC file, you're defining a function called FindPROxyForURL. The opening brace here signifies the beginning of the function. As with any brace, there **MUST** be a closing brace somewhere. In this case, you will **ALWAYS** close this brace as the very last line of your PAC file. See the PAC File examples for what it looks like.

Parenthesis

Parenthesis are used in two different ways - Sending values to other functions or grouping values together in conditional (IF) statements. As with braces, parenthesis always come in pairs - An open "(" and close ")" must be matched together.

If you want to call a JavaScript function (either built-in or one you created) you need to use set of parenthesis after the function name. If you need to pass a variable to the function you put it inside of the parenthesis. Examples:

```
isResolveable(host)
shExpMatch(host, "*cnn.com")
alert("Hello World!")
```

Whenever you use a conditional (IF statement) you need to put the "test" inside a set of parenthesis. In some cases, you might be using a function which has parenthesis of its own. This means multiple layers of parenthesis, each of which has to have an open and close. Here are some examples.

```
if (host == "www.cnn.com")
if (dnsDomainIs(host, ".cnn.com"))
if (isInNet(myIpAddress(), "192.168.1.0", "255.255.255.0"))
```

With parenthesis, it's **VERY** easy to get them mismatched, especially in a long line. The trick is to count the open parenthesis and make sure you've got the same number.

Lastly, parenthesis can "group" different things together in a conditional, combined with a Boolean operator, discussed below in more detail.

Conditionals

This is one of the key aspects of a PAC file - Testing and checking values to determine how the PAC file should respond.

Using the IF statement

The "if" statement is the fundamental building block for PAC file conditionals.

The general format for using an "if" statement is the word "if" followed by a value or values to be tested in parenthesis followed by a series of actions to be taken inside of braces. If the value is true, the actions inside the braces are run, otherwise the PAC file skips down to the line below the close brace and continues processing. For example, to send traffic to a proxy for the host webmail.company.com you would do the following.

```
if (host == "webmail.company.com") {  
    return "PROXY proxy.company.com:8080";  
}
```

Using the IF / ELSE

In some cases, you will want to check a value and perform an action if that value is true and a DIFFERENT action if that value is not true. To do this, you use an ELSE at the end of your IF statement. If the IF statement is true, it executes normally. If the IF statement is false, the code in the ELSE brackets is executed, as shown below.

```
if (host == "testsite.company.com") {  
    alert("You are going to the test site!");  
}  
else {  
    alert("You are NOT going to the test site!");  
}
```

ELSE statements are rarely used in most PAC files. A PAC file is typically very linear - You check one thing, if that doesn't match, you go onto the next. There are few absolute "If this do that, otherwise do that" kind of scenarios. There are, however, cases where using an ELSE can be very useful. For an example of an ELSE "in action" check out the Simple PAC File with Load Balancing sample PAC file.

Using Boolean Operators

Boolean operators are a way to increase the flexibility of a conditional. The primary Boolean operators are AND, OR and NOT. These very simple building-blocks can be combined together to form some very powerful tests. JavaScript uses special characters for these operators. &&

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

is used to represent AND, || is used to represent OR and ! is used to represent NOT.

A Boolean AND means that all the things being compared must be true for the overall statement to be true. The OR means that only one of the things in the list need to be true for the overall statement to be true.

Examples:

```
if (host == "www.company.com" && isInNet(myIpAddress(), "192.168.0.0", "255.255.0.0"))
if (host == "webmail.company.com" || host == "intranet.company.com")
if (dnsDomainIs(host, ".company.com") || isResolvable(host))
```

The Boolean NOT means that nothing in the conditional should be true to match. Examples:

```
if (!isPlainHostName(host))
if (!dnsDomainIs(host, "www.company.com"))
```

Boolean operators can also combine together more complex conditionals put in parenthesis. When this occurs, the sub-conditional inside the parenthesis is evaluated first, then the larger conditional. For example...

```
if ( (host == "www.company.com" || host == "webmail.company.com") &&
isInNet(myIpAddress(), "192.168.0.0", "255.255.0.0"))
```

The browser will first check to see if the host is www.company or webmail.company.com. If one of those is true then it will check to see if the host's local IP address is in the specified subnet. If that all is true, then the overall condition will be true, otherwise it will be false.

Conditionals and Boolean logic are immensely powerful often require a lot of thinking and processing, especially when trying to work out how they interact with a number of AND, OR, and NOT. (especially the NOT operator - It adds a lot of power but can be very confusing). If you're having trouble trying to visualize a complex conditional statement, write it out on a piece of paper. You might try to substitute simple letters or variable names instead of each condition. "IF (A and B) OR NOT C " or "IF (localdomain and localsubnet) AND NOT (dmzsubnet or InternetHostedServer)..." Get your head wrapped around it there in simple terms and walk through some possibilities to make sure it works right.

Details on Selected PAC File Functions

Most common functions are covered in the Netscape guide. The following are those that are not

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

effectively covered or are can be used in a different manner.

IsInNet()

isInNet() is a very, very useful function. It is typically used in two different ways: Identifying the client IP address or the host server IP address.

isInNet(myIpAddress(), "a.b.c.d", "w.x.y.z") can be used to check the local IP address of the client machine against a subnet a.b.c.d with the mask of w.x.y.z.. Example:

```
if (isInNet(myIpAddress(), "10.1.33.0", "255.255.255.0")) {  
  
return "PROXY proxy1.company.net:8000";  
  
}
```

This is a very handy function when you need to make different routing decisions based on the IP address of the client. i.e. different proxies which are geographically distributed, machines on one particular subnet needing special handling based on a special network configuration, etc. isInNet(host, "a.b.c.d", "w.x.y.z") can be used to check the IP address of the host that the request is going to. This can be somewhat dangerous, however – See the "lessons learned" section before using this. Example:

```
if (isInNet(host, "192.168.0.0", "255.255.0.0")) {  
  
return "PROXY proxy1.company.com:8000";  
  
}
```

shExpMatch and test() Regex Matches

shExpMatch() is simple function to perform a regex match:

```
if (shExpMatch(host, "localhost.*")) { ..... }
```

You can also use a "test" to match a regex. You do this by creating a variable with the regular expression and then testing it against another variable. Example:

```
reip = /^d+.d+.d+.d+$/g;if (reip.test(host)) { ..... }
```

This can be useful and lets you do a few things a bit more gracefully than shExpMatch. It also lets you re-use the regular expression multiple times without having to enter it each time you need to do a check which can be useful. I have, however, heard of times when the Regex test matches don't work as effectively as shExpMatch, so test carefully.

alert()

`alert()` is a native JavaScript function to create a pop-up on the screen. This can be VERY handy for troubleshooting and diagnostics of a PAC file. Just be aware that EVERY request that is processed through the PAC file will result in a pop-up. Be very cautious how you use it, especially in a PAC file that's going to be put into production. One note – I've noticed that Firefox doesn't seem to display alerts as well as IE does.

Example:

```
alert("I got to this point, something in the PAC file must be broken later");
```

PAC File Tricks and Tips

Dealing with Case Sensitivity

As mentioned above, PAC files are case sensitive. If you are seeing issues with upper/lower case URL's it is relatively simple to convert everything to lower case at the top of the PAC file and not have to worry about case later on. To do so, simply put this section somewhere near the top of your PAC file:

```
var lhost = host.toLowerCase();  
host = lhost;
```

Effective use of Indentations

Another fairly simple programming trick is to make sure you effectively use indentations. It makes your PAC file much easier to read and easier to troubleshoot. The very simple rule says "If you are putting something inside braces ({ }), indent it one more tab stop. Your close brace should be at the same indent level as the item that opened the brace. The only exception to this rule is that you don't need to indent your entire PAC file that's between the "function FindProxyForURL(url, host) {" at the top and the very last closing brace at the bottom. It's safe

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

to cheat here. Just be sure to indent your IF statements and make things line up nicely for readability. You (and your co-workers) will be happy about this later as they can more easily read through the PAC file.

For a good examples of indenting, see the sample PAC files in the menus to the left.

Dealing with localhost and loopback addresses

Localhost and loopback should always bypass the proxy – Put this near the top of your PAC file.

```
if ((host == "localhost") ||
    (shExpMatch(host, "localhost.*")) ||
    (host == "127.0.0.1")) {
    return "DIRECT";
}
```

Safely Blocking Sites at the Browser Instead of the Proxy

Blocking sites is also handy. This can be done for a number of reasons – Spyware/malware sites are very good examples Blocking these sites can be done very easily – Simply return a proxy value somewhere on a loopback address so that the requests never actually leave the local machine to take up network bandwidth. The only caveat with this is to ensure that your selection of port number isn't actually listening on the PC which could odd behavior.

```
if (dnsDomainIs(host, ".badspyware.com") ||
    dnsDomainIs(host, ".worsespyware2.com")) {
    return "PROXY 127.0.0.1:48890";
}
```

Using alert() for Notifications and Troubleshooting

Alert() can be used very effectively to help troubleshooting browser issues, especially the one mentioned above where the browser occasionally picks the wrong IP for myIpAddress(). For this, pick a bogus hostname – proxyinfo.company.com would work fine. It doesn't have to be in DNS or registered anywhere – Just placing it in the PAC file is all you need. Be aware, however that the alert message is displayed with EVERY request that matches the condition. If you accidentally get an alert in the wrong place it can be very annoying and render the browser

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

nearly useless. Used properly, however, it can be VERY handy.

```
if ((host == "proxyinfo.company.com")) {  
    alert("Local IP address is: " + myIpAddress());  
}
```

Using Variables

Using variables within a PAC file can be an extremely powerful way to simplify your PAC file. You can create variables to hold almost any value - A client's IP address, a proxy server address, a true/false value, etc. This variable can be read and reset at any point in the PAC file.

To use a variable, all you need to do is set it to a value. For example, to set a variable called "myip", enter the following near the top of your PAC file.

```
myip = myIpAddress();
```

When setting a variable, you can optionally insert the word "var" in front of it. I typically do not do this, however.

Once the variable has been set, you can use the variable name at any further point in the PAC file. For example:

```
if(isInNet(myip, "192.160.1.0","255.255.255.0")) { .... }
```

Why would you do this? In this particular case, it's much "cheaper" computationally to call the function myIpAddress() once and put it into a variable rather than to call it multiple times.

One of most effective uses of variables is to set a variable to one piece of data and then change it as your PAC file progresses. I use this for setting a proxy variable and then adjusting it based on client IP address or other conditions. When I get to a RETURN statement I simply return the proxy variable. Here's an example of how that can be used:

```
function FindProxyForURL(url, host) {  
    // Set the default proxy variable that users get if they don't match  
    // any more specific rule.  
    proxy = "PROXY coreproxy.company.com:8000";  
  
    // Los Angeles WAN subnets go to LA proxy  
    if (isInNet(myIpAddress(), "10.100.0.0", "255.252.0.0")) {  
        proxy = "PROXY la-proxy.company.com:8000";  
    }  
}
```

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

```
// New York WAN subnets go to New York proxy
if (isInNet(myIpAddress(), "10.200.0.0", "255.252.0.0")) {
    proxy = "PROXY ny-proxy.company.com:8000";
}
```

When writing this kind of PAC file, remember to start with the most generic and get to the most specific. If you need to route a single /24 to a specific proxy, do it **AFTER** you're done routing the /16's. Pay attention to where the variable gets set and make sure it doesn't get overwritten later.

When you have reached a point in your PAC file that you need to return a proxy value, you just return the variable, as shown below.

```
return proxy;
```

Safely using `IsInNet(host, ...)`

It is very useful to use `isInNet(host, ...)`, especially when the host is an IP address and you're trying to match it. For example, you might need to send all traffic in the 10.0.0.0/8 and 192.168.0.0/16 address spaces browser direct but all other direct IP's via the proxy. Unfortunately, just using `IsInNet(host, ...)` alone causes problems, discussed the Lessons Learned article.

Fortunately, there is a way around it: Write an IF statement that uses a regex check to see if the host is an IP address then use `IsInNet(host, ...)` to check to see if it's in a specific subnet. Example:

```
reip = /^d+.d+.d+.d+$/g;
if (reip.test(host)) {
    if (isInNet(host, "10.0.0.0", "255.0.0.0") ||
        isInNet(host, "192.168.0.0", "255.255.0.0")) {
        return "DIRECT";
    }
}
```

Some have reported issues with later versions of IE not working properly with this type of regex test, but that `shExpMatch` works properly.

```
if (shExpMatch(host, "/^d+.d+.d+.d+$/g")) {
    if (isInNet(host, "10.0.0.0", "255.0.0.0") ||
        isInNet(host, "192.168.0.0", "255.255.0.0")) {
        return "DIRECT";
    }
}
```

```
}
```

You can also use simple regex matches for your IP space, but this is far more elegant and effective, especially if you're testing multiple subnets.

Using Substrings

It is possible to check only a specified range of characters within the URL or within the host variable. This is referred to as a substring. It can be extremely handy for many things. One of the most common ways to use a substring is to check what protocol it is. This might be handy, for example, if you have a configuration where you have a separate proxy for different protocols – i.e. HTTP requests go to the main company proxy, ftp requests go to the FTP relay and MMS links go to the streaming infrastructure. One important note on substrings - Because of IE's Automatic Proxy Results Cache (See the Lessons Learned section for details) the browser only queries the PAC file once per host and caches that result. Be sure to consider this when using substrings.

Using substrings is fairly simple. The syntax is "varname.substring(begin,end)" where varname is the text-based variable (i.e. host or url) that you want to check, begin is where the substring starts and end is where it finishes. Keep in mind that the first character in a string is position #0, not #1 as you might expect.

Examples:

```
if (url.substring(0,4) == "http") { return "PROXY http-proxy.company.com:8000"; } //matches  
HTTP and HTTPS URLs  
if (url.substring(0,3) == "ftp") { return "PROXY ftp-proxy.company.com:8000"; } //matches  
FTP:// links  
if (url.substring(0,3) == "mms") { return "PROXY http-proxy.company.com:8000"; } //matches  
MMS links
```

This is only one way to use substrings – If you can find a consistent character position within the URL you can split it out and look at it.

Advanced JavaScript Ideas

This guide is focused on the things that I've found are useful when writing PAC files. That isn't to say that it covers everything you might need to do. JavaScript is a complex language with a lot of very powerful features. You can use nearly all of them within a PAC file. Don't let this (or any other such guide) limit you to what you can try. Dig into a JavaScript reference and experiment a bit with some of the more advanced features. Think about some of the string manipulation features (length, used in combination with substring, for example), using arrays, writing your own reusable subroutines within the PAC file, etc.

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

Here's a non-real-world example to get you thinking about what you could do...

Let's say your company has fifty office buildings. Each one is assigned a /16 in 10.x – 10.1.0.0/16, 10.2.0.0/16, etc. Each floor gets a range of ten /24 subnets. Floor 1 is .10 -.19, floor 2 is .20-29, etc. In each building, the executives are placed on the 20th floor. So, their subnet range is .200 - .209. An executive in building #13 might be assigned an IP address of 10.13.202.84.

Your team has built a special proxy to be used only by executives, to ensure they get the fastest possible response time. Using basic PAC file functions described so far, you'd have to put in 100+ `isInNet(myIpAddress())` lines to match all of these. But there's a better way – Using the JavaScript `split` function and arrays. Here's how..

```
var myip = myIpAddress(); // Set a variable for the local IP address.
var my-addr-array = myip.split("."); // Split that IP address var into an array.
var mysubnet=parseInt(my-addr-array[2]); // Convert array element #2 into a number and store
it in a new variable
if ((mysubnet >= 200) && (mysubnet <= 209)) { // If that number is between 200 and 209//
  proxy = "PROXY execproxy.company.com:8000"; // Assign it to the executive proxy
}
else {
  proxy = "PROXY proxy.company.com:8000"; // Otherwise they get the standard proxy.
}
```

Another very useful JavaScript idea is using `split`. This is a way to break out a string into chunks by splitting on a specific character. The result of this will be an array that you can use containing the split-apart text. A good example of this is to split the client IP address into chunks. For example..

```
var myip=myIpAddress()
var octects=myip.split(".")
```

You now have an array variable called "octects" that is the different octects of the IP address string split into an array. You can refer to "octects[0]" for the first octect, "octects[1]" for the second, etc.

One important item to keep in mind is that a string value and a number value are very different. The character "3" stored in text is just that - A string character. The number 3 is a real number that can be manipulated. In the above case, for a IP address of 192.168.103.93 you will have an array with four text strings. "192", "168", "103" and "93". These can be used for an exact match, but not to compare or run any math. You can, however, convert them to numbers using the `ParseInt()` function. Here's an example:

```
firstoctet = ParseInt(octects[0]);
```

Now, the variable "firstoctet" will contain the number 192, not the string "192". This new

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

variable can use any type of math function, greater/less than compares, etc.

These are just a few example – If you can think of something you need to do, it's probably possible to do it. Need to block all “.pif” files? Use the url.length-3 as the beginning of your url.substring and url.length as the end of it – Is the substring “pif”? If so, send it to a loopback address, like a spyware block. Don't be afraid to dig in and experiment.

It is, of course, important to add the standard caveat – Assume that you're going to get hit by a bus tomorrow. Make sure that someone else can support your PAC file. Document it, add appropriate comments, etc. and follow the golden rule - Keep It Simple and Supportable! A PAC file is not the place to be writing obfuscated JavaScript!

Proxy Load Balancing within a PAC File

Many organizations have multiple proxy servers in the same location without a hardware load balancer to distribute traffic. This section provides a method to load balance traffic between two proxies by looking at the last octet of the users' IP address. Addresses with even numbers go to one proxy, addresses with odd numbers go to another. This includes failover, should one proxy become non-responsive. PAC-file based proxy failover can have it's challenges (as noted in the Lessons Learned section) but is perfectly acceptable to use here.

This section is based on a document by a Novell BorderManager guru named Shawn Pond and is used with his permission. It is simple, logical and easy to implement. His original document can be found at www.novell.com/coolsolutions/feature/7949.html.

First, the PAC file code:

```
// Find the 4th octet
var myip=myIpAddress()
var ipbits=myip.split(".")
var myseg=parseInt(ipbits[3])

// Check to see if the 4th octet is even or odd
if (myseg==Math.floor(myseg/2)*2) {
  // Even
  proxy = "PROXY p1.company.com:8080; PROXY p2.company.com:8080";
}
else {
  // Odd
  proxy = "PROXY p2.company.com:8080; PROXY p1.company.com:8080";
}
```

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

In this code, the first thing we do is to find the users IP address into a variable called "myip". We then split the address into an array called "ipbits", separating by the period character. We find convert the 4th octet (ipbits[3]) and convert it to a numeric value which is stored in the variable "myseg":

```
var myip=myIpAddress()
var ipbits=myip.split(".")
var myseg=parseInt(ipbits[3])
```

To check if even/odd we divide it by the last octet by two, discarding any remainder, then multiply that result by two. If the result is the same as the original 4th octet then it is even. If the number is different then the last octet is odd. If it's even, we set the variable called "proxy" to be the proper proxies to use - P1 first, then P2 as a failover. If it's odd, we set the "proxy" variable to the proper value for odd-number users - P2 as first and P1 as failover.

```
// Check to see if the 4th octet is even or odd
if (myseg==Math.floor(myseg/2)*2) {
  // Even
  proxy = "PROXY p1.company.com:8080; PROXY p2.company.com:8080";
}
else {
  // Odd
  proxy = "PROXY p2.company.com:8080; PROXY p1.company.com:8080";}
```

Add the rest of your PAC file code. When you have reached a point in your PAC file that you need to return a proxy value, you just use "return proxy;" to send back the proper proxy value for the user.

For an example of this, see the "Simple PAC File with Load Balancing" in the "Sample PAC Files" section. This could also be integrated into the more complex PAC files - If you're interested in this, put a post in the forums and I'll write it up.

Lessons Learned

The following are some issues that I've found in my time writing and deploying PAC files. Sometimes even the most trivial functions can have serious repercussions on the network and browser performance, so be careful on what you put into your PAC file.

Some Browsers are Case Sensitive

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

It is always important to keep that some browsers are case sensitive with respect to hosts. "www.ibm.com" is not the same as "WWW.IBM.COM" and won't match the same rules in a PAC file. Granted, most of the world uses lower case, but you never quite know when you're going to run into someone that might capitalize an acronym or a company name somewhere in their pages or in their code.

You can either simply hope that your users and sites they access stick to lower case or you can use some of the tips I mention later on in this section to convert everything to lower case.

Note: Most modern browsers simply convert all the host variables to lower case anyhow, so this probably isn't a big deal anymore but it used to cause some challenges. If you haven't seen this as a problem I wouldn't worry about it too much.

The danger of `IsInNet`, `isResolveable` and `dnsResolve` forcing client DNS Lookups for every Query

Using `isInNet(host, "a.b.c.d")` can cause major chaos with your DNS infrastructure. This relatively innocuous looking function will force EVERY browser to perform a reverse DNS lookup on every unique hostname that it tries to contact that's not already a simple IP address. This can rapidly have significant consequences, depending on how many workstations you have and how robust your DNS infrastructure is. This doesn't necessary mean that you can't use this function, but that you must do it carefully and understand the implications. See the "PAC File Tricks" section for one way that it can be used safely.

The function `isResolveable` and `dnsResolve` have very similar issues as the described above. They force a DNS query every time the PAC is parsed for every request. Be careful using this for the same reasons. If you have a problem that only one of these functions can solve, use it carefully. Ensure that you don't do unrestricted lookups for all hosts. Use logic like "if the domain name is company.com and `IsResolveable`"

If you use these functions, unexpected behavior might result. For example, if you try to look up an invalid domain name it's possible that there might be a 30+ second delay for the request as the client tries to resolve the name. It can also place a tremendous load on your company's DNS infrastructure, so be careful of its capacity.

Returning Multiple Proxy Servers – A Warning

In a PAC file, it is perfectly valid to provide multiple proxy servers in the return. The browser will first try to use the first proxy and, should it timeout or appear to fail, retry the query on the second and then third and so on. Again, this sounds relatively harmless but Internet Explorer interprets "timeout" very loosely. Experience has shown that if a user tries to access a WEB PAGE that doesn't respond back in a timely manner IE will decide that it is the result of a proxy timeout and start sending requests to the secondary proxy. Needless to say, if your proxies are geographically dispersed this can start to spray traffic all across the network. This could be potentially mitigated by doing very careful tuning with proxy and Internet Explorer timeouts. Summary: If you use these functions be very, very careful that your traffic is going where you

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

expect it to and test it extensively under a number of failure scenarios.

In general, for the large enterprise, a Global Server Load Balancing solution (i.e. F5's GTM, Cisco GSS, etc.) provides for far more efficient failover and redundancy across the board.

A PAC is Processed Only Once Per Hostname

A "Feature" of Internet Explorer is that it only reads the PAC file once per hostname and then remembers the result for all future queries. This means that you cannot send traffic to <http://www.company.com/foo.html> via the proxy and <http://www.company.com/bar.html> browser direct by editing the PAC file. This is referred to by Microsoft as "Automatic Proxy Result Cache". More information is available on the Microsoft website – A direct link can be found in the References menu, under the Vendor links.

Microsoft Word and SmartQuotes

Microsoft Word has a feature called "SmartQuotes" which will replace your standard vertical quotes with very pretty indented quotes. This looks great in text, but these SmartQuotes aren't readable by a PAC file. Editing a PAC file in Word can result in all quotes getting SmartQuote-ified. Needless to say, this is messy. If this happens, copy-and-paste into Notepad and do a search/replace to move everything back to normal quotes. You can, of course, turn off SmartQuotes, but the million monkeys waiting to use your computer would be very dissatisfied.

Lesson learned - Don't use Word to edit any configuration.

Notepad and Word Wrap

Notepad is a significantly.. safer.. editor than Word. It does, however, have its drawbacks. I've seen it eat a configuration file when Word Wrap was turned on. It doesn't happen frequently, but it's best to turn OFF Word Wrap in Notepad before doing any major work.

Commenting your Code

All good programmers follow a simple rule. Thou Must Document Thy Code. In a PAC file, however, this should be somewhat moderated. Remember that this file could be downloaded 50+ times per day per workstation on your network. The more comments you put in, the bigger the file gets the more network bandwidth it takes and the slower it is to load. Resist the temptation to over-comment. At the same point, comments are important for historical tracking. You need to find the right mix of comments for your environment.

Invalid Workstation Responses to myIpAddress()

When using `isInNet(myIpAddress(), ...)` to find the local client IP address you will occasionally get an invalid response with Internet Explorer. This occurs when there are multiple active NIC's in the machine. IE occasionally gets confused which was used to send the request and

responds back to `myIpAddress()` with the wrong one. For example, if you have one NIC on a company network and one bound to a wireless access point `myIpAddress()` will occasionally return back the IP of the wireless NIC, even when communicating with the proxy across the corporate network. Past experience has shown this to happen vary rarely, but it *does* happen. Unfortunately, there is no rhyme or reason in why or when this occurs and it's impossible to predict. In short, don't absolutely depend on this returning 100% accurate data and be sure to put in a fallback condition if none of your standard IP subnet match.

Understand how JavaScript processes Equals Signs

This is an important concept to understand if you've chosen to use variables in your PAC file. JavaScript uses the equals sign in two very different ways, depending if you use one (=) or two (==) together.

A single equals sign is always used to set a variable to a value. "Set the variable X to the value 3" is written as "`X = 3`". If you write a function that says "`if (X=3) { .... }`" it will always be true since what you're really writing is "If you can successfully set the variable X to the value 3 do { }"

If you are trying to compare two values, use two equals signs - `==`. This tells the system to do a compare and return a true/false value. If you want to check to SEE if the value of X is 3, you would use "`if (X== 3) { .... }`". If X has the value of 2, the IF will fail. If X is 3, it will succeed and whatever is between the braces will process.

IsInNet(host..) Failing

A number of users have contacted me about issues with the `isInNet` function failing when the host being parsed is a domain name and not an IP address. Good sources have indicated that this occurs in Internet Explorer 8.0 and also the Microsoft .NET Framework 2.0 and likely has something to do with an IPv6 implementation. I have not yet been able to replicate this behavior, but I do have a solution – Resolve the IP address of the host first with `dnsResolve` and then do a match with `isInNet` to look for a subnet match. As mentioned above, be careful of doing this as it can cause more DNS queries and some odd behavior, but it can be used effectively in certain circumstances. Here's an example of how this would normally be written in a PAC file:

```
if (isInNet(dnsResolve(host), "10.0.0.0", "255.0.0.0") ||  
    isInNet(dnsResolve(host), "172.16.0.0", "255.240.0.0") ||  
    isInNet(dnsResolve(host), "192.168.0.0", "255.255.0.0")) {  
    return "DIRECT";  
}
```

That forces DNS resolution of the host variable and then compares that IP address against the

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

subnet you specify using `isInNet`.

The “Fixed” way to do it would be to place the resolved host IP address in a variable, then reference that variable multiple times with `isInNet`.

```
hostip=dnsResolve(host);

if (isInNet(hostip, "10.0.0.0", "255.0.0.0") ||

isInNet(hostip, "172.16.0.0", "255.240.0.0") ||

isInNet(hostip, "192.168.0.0", "255.255.0.0")) {

return "DIRECT";

}
```

There are several links regarding this topic in the "References" menu, to the left.

Text vs. Numeric Values

It is very important to remember that a string value and a number value are stored and used very differently. The "93" stored in text is just that - A set of string characters, no different than any other text. The number 93 is a real number that can be manipulated. If you end up with a number stored as text that you use as a number you can convert it using the `parseInt()` function.

```
numvariable = parseInt(textvar);
```

Testing and Troubleshooting

When you're building a PAC file or making a major modification to one, chances are you won't get it right the first time – You'll miss a closing brace, a semi-colon, or something just won't be right. Unfortunately, most browsers provide no error handling for PAC files – It just won't load and everything will go browser direct.

The important thing to remember about troubleshooting a PAC file is that it is a totally iterative process for each URL. The browser will process the PAC file from the top until it finds an error or until it finds a valid return.

If your PAC is erroring out, use `alert()` (described above) to throw in lots of pop-ups to help you

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

find your error. If you throw in four alerts in your PAC and you only get three of showing up, you know your error is somewhere between alert #3 and alert #4. Keep using this to narrow it down and you'll soon find your problem. It's a pain, but there's no other good way to do it. Needless to say, you should do troubleshooting and testing on a PAC file that's NOT in production.

The day will come when you are trying to troubleshoot a PAC file and NOTHING works – NO alerts pop up, nothing, nada. Hair will be pulled. Co-workers will be yelled at. A bad day will be had by all. This has happened to me, it will happen to you. Before you get to extreme, do yourself a favor – Check the settings in your browser. Did you typo something? Can you copy-and-paste the exact URL shown in the autoconfig URL into the browser address bar and get back a file? This sounds strange, but it happens. Get it in your head to double-check your settings.

Obviously, PAC files require a lot of testing and tuning. Some people might not have a handy HTTP server to load them onto for testing. If that's the case, or if you want to edit them on your local computer it's relatively simple to load the PAC file from your local hard drive.

Simply copy the PAC file as a simple text file somewhere onto your hard drive. In this case, I loaded mine into a file called "C:PACTEST.PAC". To get IE to recognize it as a PAC file you need to enter it with forward slashes in the path, not back slashes, and use the file:// prefix, not http:// as you would loading it from a web server. Here's the URL for IE to load my PAC file from my hard drive - file://c:/pac/test.pac. I would, however, strongly NOT recommend using this notation for a production PAC file. Even if you don't expect to update it frequently it's still better to use a central PAC file hosted on a web server.

Google PAC Tester

Google has produced an incredible tool called the PAC Tester. This application will parse your PAC files and find the errors for you. You can specify the source IP address, to test those islnet calls and the URL to be queried. A link to it can be found in the References and Tools menu of this site. I also took the liberty of writing a quick web front end to the application - This allows you to just copy-and-paste your PAC file into the page and specify the client IP to be simulated and the URL and it will run it through the tester. It's ugly, but functional.

One note on the PAC tester - I've seen some odd behavior when I test PAC files using alert() on the Windows version. Seems to work well on other variants, including the one that runs my web front end, so try your PAC file there and see if it works.

Wrapping a PAC file in HTML

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

I received a great tip from Pasquale Scopelliti on troubleshooting syntax errors that cause a PAC file not to load. You can wrap your PAC file into a HTML document and load it via your browser. If it's HTML and the PAC has a syntax error it will report it to you and tell you what's wrong. Here's how to do so.

Create a basic HTML file on a web server that contains the following:

```
<html>
<head>
<script language="JavaScript">
```

Insert your ENTIRE PAC file here

```
</script>
</head>
<body>
test
</body>
</html>
```

Save the file to a web server or your local hard drive and then load it up in Internet Explorer. You'll just see the text "test" in the main screen of the browser, but the real action happens on the error bar – Look in the very bottom-left corner of the window for a yellow triangle with a black explanation point on top of a piece of paper. Double-click on this icon and then choose "Show details". This displays the error message in question. Note that the line # is from the top of the HTML file, not from your PAC file. It won't always give you the exact details, but it's a good way to help dig into those hard-to-find syntax errors.

Using an Online JavaScript Debugger

There are a number of Online JavaScript debuggers that you can use. They will allow you to copy/paste your PAC file into their site and it will check the syntax and let you know if there are any issues that need to be fixed. One recommended by our users is JSLint - www.jslint.com /

Loading a PAC file from your local hard drive...

Optimizing a PAC file

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

If you're to the point where you need to think about optimizing a PAC file, you probably know what you're doing and don't need much help. Some high-level observations:

Don't focus on making your PAC file shorter – Focus on how to make the browser work less. For example, if you have 75 host exceptions for your internal .company.com domain, write one IF statement that checks `dnsDomainIs(host, ".company.com")` first, then embed your 75 host checks so that they are only run if the host is in .company.com. Most of your traffic probably goes to the Internet, so if you can avoid 75 unique host checks for .company.com hostnames you've saved a lot of processing. There is, of course the network bandwidth tradeoff – Longer and more complex means a larger PAC file to be sent across the network. Try to find a balance that works for you. Example:

```
if (dnsDomainIs(host, ".company.com")) {  
  if ((host == "zeus.company.com") ||  
      (host == "frodo.company.com") ||  
      (host == "zaphod.company.com")) {  
    return proxy;  
  }  
}
```

```
if ((host == "mail.company.com")) ||  
(host == "intranet.company.com") {  
  return "DIRECT";  
}
```

```
} // End of hosts in .company.com
```

If you need to check a lot of IP subnets, try to lump them together as much as possible. Use a single regex to check the host to see if it's an IP address (described above in the tip on using `isInNet(host, ...)` safely) and then list all of the IP's you want to compare against. If you need to do a lot of checks for the local IP address, set a variable to `myIpAddress()` at the top of your PAC file and then use this variable instead of `myIpAddress()` later. From a CPU load perspective, it takes less cycles to check a variable in memory than it does to perform a system call to get the local IP each time you need to compare it.

Simple PAC file

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

For this example, we will assume that an organization uses company.net as their domain. They have three hosts (www, webmail and portal) in this domain and one subdomain (public.company.net) that are on the Internet. The rest of the domain is internal and shouldn't be proxied.

```
function FindProxyForURL(url, host) {  
  // First start with the exceptions that need to be proxied  
  
  if ((host == "www.company.net") ||  
      (host == "webmail.company.net") ||  
      (host == "portal.company.net") ||  
      (dnsDomainIs(host, ".public.company.net"))) {  
  
    return "PROXY proxy1.company.net:8000";  
  
  }  
  
  // Next, we want to send all traffic to company.net browser direct  
  
  if (dnsDomainIs(host, ".company.net")) {  
  
    return "DIRECT";  
  
  }  
  
  // Default return condition is the proxy, since it's assumed that everything  
  // else is on the Internet.  
  
  return "PROXY proxy1.company.net:8000";  
  
} // End of function
```

Simple PAC file with Load Balancing

This PAC file is identical to the Simple PAC file but adds in load balancing between two proxies - p1.company.com and p2.company.com.

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

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

```
// Load Balancing Code
// Find the 4th octet
var myip=myIpAddress()
var ipbits=myip.split(".")
var myseg=parseInt(ipbits[3])

// Check to see if the 4th octet is even or odd
if (myseg==Math.floor(myseg/2)*2) {
  // Even
  proxy = "PROXY p1.company.com:8080; PROXY p2.company.com:8080";
}
else {
  // Odd
  proxy = "PROXY p2.company.com:8080; PROXY p1.company.com:8080";
}

// First start with the exceptions that need to be proxied
if ((host == "www.company.net") ||
    (host == "webmail.company.net") ||
    (host == "portal.company.net") ||
    (dnsDomainIs(host, ".public.company.net"))) {
  return proxy;
}

// Next, we want to send all traffic to company.com browser direct
if (dnsDomainIs(host, ".company.net")) {
  return "DIRECT";
}

// Default return condition is the proxy, since it's assumed that everything
// else is on the Internet.

return proxy;

} // End of function
```

Complex PAC file

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

The following PAC file is for the same company described above, but with some more complexities:

- * The company has three proxies – coreproxy.company.net is the used for the enterprise except for the LA and NYC areas which use laproxy.company.net and nyproxy.company.net, respectively.
- * Traffic to IBM.COM domain needs to be sent to the core proxy for all LA users.
- * The subnet 10.100.254.0/24 is a lab with its own dedicated proxy – labproxy.company.net.
- * The trading floor has a dedicated connection to Bloomberg. They should go browser direct for this domain. Everyone else should go through the proxy.
- * There is an Internet DMZ at 10.250.0.0/16 that can only be accessed via the proxy.
- * All other hosts in 10.0.0.0/8 must be accessed browser direct.

```
function FindProxyForURL(url, host) {

pacver = "company.net PAC file version 7.2, January 4th, 2008";

// Convert everything to lower case.
var lhost = host.toLowerCase();
host = lhost;

// Set the default proxy variable that users get if they don't match
// any more specific rule.
proxy = "PROXY coreproxy.company.net:8000";

// Los Angeles WAN subnets go to LA proxy
if (isInNet(myIpAddress(), "10.100.0.0", "255.252.0.0")) {
    proxy = "PROXY laproxy.company.net:8000";
}

// New York WAN subnets go to NY proxy
if (isInNet(myIpAddress(), "10.200.0.0", "255.252.0.0")) {
    proxy = "PROXY nyproxy.company.net:8000";
}

// Lab subnet goes to it's own proxy
if (isInNet(myIpAddress(), "10.100.265.0", "255.255.255.0")) {
    proxy = "PROXY labproxy.company.net:8000";
}

// Spyware blocks
if (dnsDomainIs(host, ".badspyware1.com") ||
    dnsDomainIs(host, ".worsespyware2.com")) {
    return "PROXY 127.0.0.1:49234";
}
```

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

```
// Loopback and localhost goes browser direct always.
if ((host == "localhost") ||
    (shExpMatch(host, "localhost.*")) ||
    (host == "127.0.0.1")) {
    return "DIRECT";
}

// Trading Floor Override for Bloomberg
if (isInNet(myIpAddress(), "10.101.32.0", "255.255.252.0")) {
    if (dnsDomainIs(host, "Bloomberg.com")) {
        return "DIRECT";
    }
}

// ibm.com doesn't work through LA, set the variable to core proxy.
// Note you could just return the core proxy here, too but setting the variable is
// more consistent.

if ((dnsDomainIs(host, ".ibm.com")) &&
    (proxy == "PROXY lproxy.company.net:8000")) {
    proxy = "PROXY coreproxy.company.net:8000";
}

// Utility
if ((host == "proxyinfo.company.net")) {

    alert("Local IP address is: " + myIpAddress());
    alert("PAC File Version: " + pacver);

}

// hosts in company.net on the Internet need to be proxied.
if ((host == "www.company.net") ||
    (host == "webmail.company.net") ||
    (host == "portal.company.net") ||
    (dnsDomainIs(host, ".public.company.net"))) {
    return proxy;
}

// Test to see if host is an IP address
reip = /^d+.d+.d+.d+$/g;

if (reip.test(host)) {
    // Check for an Internet DMZ address
```

The Practical Proxy PAC file guide

Written by Administrator

Tuesday, 20 April 2010 05:48 - Last Updated Tuesday, 20 April 2010 09:48

```
if (isInNet(host, "10.250.0.0", "255.255.0.0")) {
return proxy;
}

// Check for an internal 10.x IP address
if (isInNet(host, "10.0.0.0", "255.0.0.0")) {
return "DIRECT";
}
}

// Next, we want to send all traffic to company.net browser direct
if (dnsDomainIs(host, ".company.net")) {

return "DIRECT";

}

// Default return condition is the proxy, since it's assumed that everything
// else is on the Internet.

return proxy;

} // End of function
```