NAME

dhcp-eval - ISC DHCP conditional evaluation

DESCRIPTION

The Internet Systems Consortium DHCP client and server both provide the ability to perform conditional behavior depending on the contents of packets they receive. The syntax for specifying this conditional behaviour is documented here.

REFERENCE: CONDITIONAL BEHAVIOUR

Conditional behaviour may be specified using the if statement and the else or elsif statements or the switch and case statements. A conditional statement can appear anywhere that a regular statement (e.g., an option statement) can appear, and can enclose one or more such statements.

CONDITIONAL BEHAVIOUR: IF

A typical conditional if statement in a server might be:

```
if option dhcp-user-class = accounting {
max-lease-time 17600;
option domain-name accounting.example.org;
option domain-name-servers ns1.accounting.example.org,
ns2.accounting.example.org;
} elsif option dhcp-user-class = sales {
max-lease-time 17600;
option domain-name sales.example.org;
option domain-name-servers ns1.sales.example.org,
ns2.sales.example.org;
} elsif option dhcp-user-class = engineering {
max-lease-time 17600;
option domain-name engineering.example.org;
option domain-name-servers ns1.engineering.example.org,
ns2.engineering.example.org;
} else {
max-lease-time 600;
option domain-name misc.example.org;
option domain-name-servers ns1.misc.example.org,
ns2.misc.example.org;
}
```

On the client side, an example of conditional evaluation might be:

```
# example.org filters DNS at its firewall, so we have to use their DNS
# servers when we connect to their network. If we are not at
# example.org, prefer our own DNS server.
if not option domain-name = example.org {
prepend domain-name-servers 127.0.0.1;
}
```

The **if** statement and the **elsif** continuation statement both take boolean expressions as their arguments. That is, they take expressions that, when evaluated, produce a boolean result. If the expression evaluates to true, then the statements enclosed in braces following the **if** statement are executed, and all subsequent **elsif** and **else** clauses are skipped. Otherwise, each subsequent **elsif** clause's expression is checked, until an elsif clause is encountered whose test evaluates to true. If such a clause is found, the statements in braces following it are executed, and then any subsequent **elsif** and **else** clauses are skipped. If all the **if** and **elsif** clauses are checked but none of their expressions evaluate true, then if there is an **else** clause, the statements enclosed in braces following the **else** are evaluated. Boolean expressions that evaluate to null are treated as false in conditionals.

CONDITIONAL BEHAVIOUR: SWITCH

The above example can be rewritten using a switch construct as well.

```
switch (option dhcp-user-class) {
case accounting:
max-lease-time 17600;
option domain-name accounting.example.org;
option domain-name-servers ns1.accounting.example.org,
ns2.accounting.example.org;
case sales:
max-lease-time 17600;
option domain-name sales.example.org;
option domain-name-servers ns1.sales.example.org,
ns2.sales.example.org;
break;
case engineering:
max-lease-time 17600;
option domain-name engineering.example.org;
option domain-name-servers ns1.engineering.example.org,
ns2.engineering.example.org;
break;
default:
max-lease-time 600;
option domain-name misc.example.org;
option domain-name-servers ns1.misc.example.org,
ns2.misc.example.org;
break;
}
```

The **switch** statement and the **case** statements can both be data expressions or numeric expressions. Within a switch statement they all must be the same type. The server evaluates the expression from the switch statement and then it evaluates the expressions from the case statements until it finds a match.

If it finds a match it starts executing statements from that case until the next break statement. If it doesn't find a match it starts from the default statement and again proceeds to the next break statement. If there is no match and no default it does nothing.

BOOLEAN EXPRESSIONS

The following is the current list of boolean expressions that are supported by the DHCP distribution.

data-expression-1 = *data-expression-2*

The = operator compares the values of two data expressions, returning true if they are the same, false if they are not. If either the left-hand side or the right-hand side are null, the result is also null.

data-expression-1 ~= *data-expression-2* *data-expression-1* ~~ *data-expression-2*

The ~= and ~~ operators (not available on all systems) perform extended regex(7) matching of the values of two data expressions, returning true if *data-expression-1* matches against the regular expression evaluated by *data-expression-2*, or false if it does not match or encounters some error. If either the left-hand side or the right-hand side are null or empty strings, the result is also false. The ~~ operator differs from the ~= operator in that it is case-insensitive.

boolean-expression-1 **and** *boolean-expression-2*

The **and** operator evaluates to true if the boolean expression on the left-hand side and the boolean expression on the right-hand side both evaluate to true. Otherwise, it evaluates to false. If either the expression on the left-hand side or the expression on the right-hand side are

null, the result is null.

boolean-expression-1 **or** *boolean-expression-2*

The **or** operator evaluates to true if either the boolean expression on the left-hand side or the boolean expression on the right-hand side evaluate to true. Otherwise, it evaluates to false. If either the expression on the left-hand side or the expression on the right-hand side are null, the result is null.

not *boolean-expression*

The **not** operator evaluates to true if *boolean-expression* evaluates to false, and returns false if *boolean-expression* evaluates to true. If *boolean-expression* evaluates to null, the result is also null.

exists *option-name*

The **exists** expression returns true if the specified option exists in the incoming DHCP packet being processed.

known

The **known** expression returns true if the client whose request is currently being processed is known - that is, if there's a host declaration for it.

static

The **static** expression returns true if the lease assigned to the client whose request is currently being processed is derived from a static address assignment.

DATA EXPRESSIONS

Several of the boolean expressions above depend on the results of evaluating data expressions. A list of these expressions is provided here.

substring (*data-expr*, *offset*, *length*)

The **substring** operator evaluates the data expression and returns the substring of the result of that evaluation that starts *offset* bytes from the beginning, continuing for *length* bytes. *Offset* and *length* are both numeric expressions. If *data-expr*, *offset* or *length* evaluate to null, then the result is also null. If *offset* is greater than or equal to the length of the evaluated data, then a zero-length data string is returned. If *length is greater then the remaining length of the evaluated data after offset*, then a data string containing all data from *offset* to the end of the evaluated data is returned.

suffix (*data-expr*, *length*)

The **suffix** operator evaluates *data-expr* and returns the last *length* bytes of the result of that evaluation. *Length* is a numeric expression. If *data-expr* or *length* evaluate to null, then the result is also null. If *suffix* evaluates to a number greater than the length of the evaluated data, then the evaluated data is returned.

lcase (*data-expr*)

The **lcase** function returns the result of evaluating *data-expr* converted to lower case. If *data-expr* evaluates to null, then the result is also null.

ucase (*data-expr*)

The **ucase** function returns the result of evaluating *data-expr* converted to upper case. If *data-expr* evaluates to null, then the result is also null.

option *option-name*

The **option** operator returns the contents of the specified option in the packet to which the server is responding.

config-option *option-name*

The **config-option** operator returns the value for the specified option that the DHCP client

or server has been configured to send.

gethostname()

The **gethostname()** function returns a data string whose contents are a character string, the results of calling gethostname() on the local system with a size limit of 255 bytes (not including NULL terminator). This can be used for example to configure dhclient to send the local hostname without knowing the local hostname at the time dhclient.conf is written.

hardware

The **hardware** operator returns a data string whose first element is the type of network interface indicated in packet being considered, and whose subsequent elements are client's link-layer address. If there is no packet, or if the RFC2131 *hlen* field is invalid, then the result is null. Hardware types include ethernet (1), token-ring (6), and fddi (8). Hardware types are specified by the IETF, and details on how the type numbers are defined can be found in RFC2131 (in the ISC DHCP distribution, this is included in the doc/ subdirectory).

packet (*offset*, *length*)

The **packet** operator returns the specified portion of the packet being considered, or null in contexts where no packet is being considered. *Offset* and *length* are applied to the contents packet as in the **substring** operator.

string

A string, enclosed in quotes, may be specified as a data expression, and returns the text between the quotes, encoded in ASCII. The backslash ('') character is treated specially, as in C programming: 't' means TAB, 'r' means carriage return, 'n' means newline, and 'b' means bell. Any octal value can be specified with 'nnn', where nnn is any positive octal number less than 0400. Any hexadecimal value can be specified with 'xnn', where nn is any positive hexadecimal number less than or equal to 0xff.

colon-separated hexadecimal list

A list of hexadecimal octet values, separated by colons, may be specified as a data expression.

concat (*data-expr1*, **...**, *data-exprN*)

The expressions are evaluated, and the results of each evaluation are concatenated in the sequence that the subexpressions are listed. If any subexpression evaluates to null, the result of the concatenation is null.

reverse (*numeric-expr1*, *data-expr2*)

The two expressions are evaluated, and then the result of evaluating the data expression is reversed in place, using hunks of the size specified in the numeric expression. For example, if the numeric expression evaluates to four, and the data expression evaluates to twelve bytes of data, then the reverse expression will evaluate to twelve bytes of data, consisting of the last four bytes of the input data, followed by the middle four bytes, followed by the first four bytes.

leased-address

In any context where the client whose request is being processed has been assigned an IP address, this data expression returns that IP address. In any context where the client whose request is being processed has not been assigned an ip address, if this data expression is found in executable statements executed on that client's behalf, a log message indicating there is no lease associated with this client is syslogged to the debug level (this is considered dhcpd.conf debugging information).

binary-to-ascii (*numeric-expr1*, *numeric-expr2*, *data-expr1*, *data-expr2*)

Converts the result of evaluating data-expr2 into a text string containing one number for each element of the result of evaluating data-expr2. Each number is separated from the other by the result of evaluating data-expr1. The result of evaluating numeric-expr1 specifies the base (2 through 16) into which the numbers should be converted. The result of evaluating numeric-

expr2 specifies the width in bits of each number, which may be either 8, 16 or 32.

As an example of the preceding three types of expressions, to produce the name of a PTR record for the IP address being assigned to a client, one could write the following expression:

```
concat (binary-to-ascii (10, 8, .,
reverse (1, leased-address)),
.in-addr.arpa.);
```

encode-int (*numeric-expr***, *width*)**

Numeric-expr is evaluated and encoded as a data string of the specified width, in network byte order (most significant byte first). If the numeric expression evaluates to the null value, the result is also null.

pick-first-value (*data-expr1* [... *expr*n] **)**

The pick-first-value function takes any number of data expressions as its arguments. Each expression is evaluated, starting with the first in the list, until an expression is found that does not evaluate to a null value. That expression is returned, and none of the subsequent expressions are evaluated. If all expressions evaluate to a null value, the null value is returned.

host-decl-name

The host-decl-name function returns the name of the host declaration that matched the client whose request is currently being processed, if any. If no host declaration matched, the result is the null value.

NUMERIC EXPRESSIONS

Numeric expressions are expressions that evaluate to an integer. In general, the maximum size of such an integer should not be assumed to be representable in fewer than 32 bits, but the precision of such integers may be more than 32 bits.

In addition to the following operators several standard math functions are available. They are:

operation	symbol
add	+
subtract	-
divide	/
multiply	*
modulus	%
bitwise and	&
bitwise or	|
bitwise xor	^

extract-int (*data-expr***, *width*)**

The **extract-int** operator extracts an integer value in network byte order from the result of evaluating the specified data expression. Width is the width in bits of the integer to extract. Currently, the only supported widths are 8, 16 and 32. If the evaluation of the data expression doesn't provide sufficient bits to extract an integer of the specified size, the null value is returned.

lease-time

The duration of the current lease - that is, the difference between the current time and the time that the lease expires.

number

Any number between zero and the maximum representable size may be specified as a numeric expression.

client-state

The current state of the client instance being processed. This is only useful in DHCP client configuration files. Possible values are:

- Booting - DHCP client is in the INIT state, and does not yet have an IP address. The next message transmitted will be a DHCPDISCOVER, which will be broadcast.
- Reboot - DHCP client is in the INIT-REBOOT state. It has an IP address, but is not yet using it. The next message to be transmitted will be a DHCPREQUEST, which will be broadcast. If no response is heard, the client will bind to its address and move to the BOUND state.
- Select - DHCP client is in the SELECTING state - it has received at least one DHCPOFFER message, but is waiting to see if it may receive other DHCPOFFER messages from other servers. No messages are sent in the SELECTING state.
- Request - DHCP client is in the REQUESTING state - it has received at least one DHCPOFFER message, and has chosen which one it will request. The next message to be sent will be a DHCPREQUEST message, which will be broadcast.
- Bound - DHCP client is in the BOUND state - it has an IP address. No messages are transmitted in this state.
- Renew - DHCP client is in the RENEWING state - it has an IP address, and is trying to contact the server to renew it. The next message to be sent will be a DHCPREQUEST message, which will be unicast directly to the server.
- Rebind - DHCP client is in the REBINDING state - it has an IP address, and is trying to contact any server to renew it. The next message to be sent will be a DHCPREQUEST, which will be broadcast.

REFERENCE: ACTION EXPRESSIONS

log (*priority***,** *data-expr***)**

Logging statements may be used to send information to the standard logging channels. A logging statement includes an optional priority (**fatal**, **error**, **info**, or **debug**), and a data expression.

Logging statements take only a single data expression argument, so if you want to output multiple data values, you will need to use the **concat** operator to concatenate them.

execute (*command-path* **[,** *data-expr1***, ...** *data-exprN***]);**

The **execute** statement runs an external command. The first argument is a string literal containing the name or path of the command to run. The other arguments, if present, are either string literals or data- expressions which evaluate to text strings, to be passed as command-line arguments to the command.

execute is synchronous; the program will block until the external command being run has finished. Please note that lengthy program execution (for example, in an on commit in dhcpd.conf) may result in bad performance and timeouts. Only external applications with very short execution times are suitable for use.

Passing user-supplied data to an external application might be dangerous. Make sure the external application checks input buffers for validity. Non-printable ASCII characters will be converted into dhcpd.conf language octal escapes (nnn), make sure your external command handles them as such.

It is possible to use the execute statement in any context, not only on events. If you put it in a regular scope in the configuration file you will execute that command every time a scope is evaluated.

REFERENCE: DYNAMIC DNS UPDATES

See the dhcpd.conf and dhclient.conf man pages for more information about DDNS.

SEE ALSO

dhcpd.conf(5), dhcpd.leases(5), dhclient.conf(5), dhcp-options(5), dhcpd(8), dhclient(8), RFC2132, RFC2131.

AUTHOR

Information about Internet Systems Consortium can be found at https://www.isc.org.