Exscript Documentation

Release 0.3.4

Samuel Abels

Aug 24, 2023

Contents

Exscript is a Python module and a template processor for automating network connections over protocols such as Telnet or SSH. We attempt to create the best possible set of tools for working with Telnet and SSH.

Exscript also provides a set of tools and functions for sysadmins, that simplify **regular expression matching**, **reporting** by email, **logging**, or **syslog** handling, **CSV parsing**, **ip address handling**, **template processing**, and many more.

Exscript may be used to automate sessions with routers from Cisco, Juniper, OneAccess, Huawei, or any others. If you want to configures machines running Linux/Unix, IOS, IOS-XR, JunOS, VRP, or any other operating system that can be used with a terminal, Exscript is what you are looking for.

CHAPTER 1

Using Exscript with Python

Make sure to check out the *Python Tutorial*. You may also want to look at the Python examples.

CHAPTER 2

Using the Exscript command line tool

Have a look at our *CLI Tutorial*. You will also want to learn about *Exscript Templates*.

CHAPTER 3

Main design goals

- Exscript provides **high reliability** and **scalability**. Exscript is used by some of the world's largest ISPs to maintain hundreds of thousands of sessions every day.
- Exscript is **extremely well tested**. The Exscript public API has almost 100% test coverage.
- Exscript is **protocol agnostic**, so if you are migrating from Telnet to SSH later, you can switch easily by simply changing an import statement.

CHAPTER 4

Development

Exscript is on GitHub.

CHAPTER 5

License

Exscript is published under the MIT licence.

CHAPTER 6

Contents

6.1 Installation

6.1.1 Prerequisites

Exscript requires Python 2.7, and the following modules:

```
future
configparser
pycryptodomex
paramiko>=1.17
```

6.1.2 Installing with PIP

```
sudo pip install exscript
```

6.1.3 Installing from GitHub

```
git clone git://github.com/knipknap/exscript
cd exscript
sudo make install
```

6.1.4 Running the automated test suite

If you installed from GitHub, you can run the integrated testsuite:

```
make tests
```

There shouldn't be any errors, so if something comes up, please file a bug.

6.2 Python Tutorial

6.2.1 Introduction

This is a step by step introduction to using Exscript in Python.

We'll assume that Exscript is already installed. You can confirm that your installation works by typing `exscript --version` into a terminal; if this prints the version number, your installation is complete.

We will also assume that you have at least a little bit of programming experience, though most of the examples should be pretty simple.

Exscript also has extensive *API documentation*, which may be used as a reference throughout this tutorial.

6.2.2 Getting started

As a first simple test, let's try to connect to a network device via SSH2, and execute the `uname -a` command on it.

Create a file named `start.py` with the following content:

```
from Exscript.util.interact import read_login
from Exscript.protocols import SSH2

account = read_login()     # Prompt the user for his name and password
conn = SSH2()              # We choose to use SSH2
conn.connect('localhost') # Open the SSH connection
conn.login(account)        # Authenticate on the remote host
conn.execute('uname -a')   # Execute the "uname -a" command
conn.send('exit\r')        # Send the "exit" command
conn.close()               # Wait for the connection to close
```

Awesome fact: Just replace `SSH2` by `Telnet` and it should still work with Telnet devices.

As you can see, we prompt the user for a username and a password, and connect to `localhost` using the entered login details. Once logged in, we execute `uname -a`, log out, and make sure to wait until the remote host has closed the connection.

You can see one important difference: We used `conn.execute` to run `uname -a`, but we used `conn.send` to execute the `exit` command. The reason is that *''conn.execute'' waits until the server has acknowledged that the command has completed*, while `conn.send` does not. Since the server won't acknowledge the `exit` command (instead, it just closes the connection), using `conn.execute` would lead to an error.

6.2.3 Making it easier

While the above serves as a good introduction on how to use `Exscript.protocols`, it has a few drawbacks:

1. It only works for SSH2 or for Telnet, but not for both.
2. It contains a lot of unneeded code.
3. You can't use the script to connect to multiple hosts.

Let's solve drawbacks 1. and 2. first. Here is a shortened version of the above script:

```
from Exscript.util.start import quickstart

def do_something(job, host, conn):
```

(continues on next page)

(continued from previous page)

```
    conn.execute('uname -a')

quickstart('ssh://localhost', do_something)
```

As you can see, we made two major changes:

1. We moved the code that executes `uname -a` into a function named `do_something`. (Note: We could have picked any other name for the function.)
2. We imported and used the `Exscript.util.start.quickstart()` function.

`Exscript.util.start.quickstart()` does the following:

1. It prompts the user for a username and a password.
2. It connects to the specified host, using the specified protocol.
3. It logs in using the given login details.
4. It calls our `do_something()` function.
5. When `do_something()` completes, it closes the connection.

6.2.4 Running a script on multiple hosts

In practice, you may want to run this script on multiple hosts, and optimally at the same time, in parallel. Using the `Exscript.util.start.quickstart()` function, this is now really easy:

```
from Exscript.util.start import quickstart

def do_something(job, host, conn):
    conn.execute('uname -a')

hosts = ['ssh://localhost', 'telnet://anotherhost']
quickstart(hosts, do_something, max_threads=2)
```

We only changed the last lines of the script:

1. We pass in two hosts, `localhost` and `anotherhost`. Note that `localhost` uses SSH, and `anotherhost` speaks Telnet.
2. We added the `max_threads=2` argument. This tells Exscript to open two network connections in parallel.

If you run this script, it will again ask for the login details, and run `do_something()` for both hosts in parallel.

Note that the login details are only asked once and used on both hosts - this may or may not be what you want. For instructions one using different login mechanisms please refer to the section on accounts later.

6.2.5 Loading hosts from a text file

If you do not wish to hard code the host names into the script, you may also list them in a text file and load it using `Exscript.util.file.get_hosts_from_file()` as follows:

```
from Exscript.util.start import start
from Exscript.util.file import get_hosts_from_file

def do_something(job, host, conn):
    conn.execute('uname -a')
```

(continues on next page)

(continued from previous page)

```
hosts = get_hosts_from_file('myhosts.txt')
start(hosts, do_something, max_threads=2)
```

6.2.6 Reading login information

Depending on how you would like to provide the login information, there are a few options. The first is by hard coding it into the hostname:

```
hosts = ['ssh://localhost', 'telnet://myuser:mypassword@anotherhost']
quickstart(hosts, do_something, max_threads=2)
```

In this case, `quickstart` still prompts the user for his login details, but the entered information is only used on hosts that do not have a user/password combination included in the hostname.

If you do not wish to hard code the login details into the hostname, you can also use the Exscript.Host object as shown in the following script:

```
from Exscript import Host, Account
...
account1 = Account('myuser', 'mypassword')
host1 = Host('ssh://localhost')
host1.set_account(account1)

account2 = Account('myuser2', 'mypassword2')
host2 = Host('ssh://otherhost')
host2.set_account(account2)

quickstart([host1 , host2], do_something, max_threads=2)
```

This script still has the problem that it prompts the user for login details, even though the details are already known. By using `Exscript.util.start.start()` instead of `Exscript.util.start.quickstart()`, you can avoid the prompt, and optionally pass in a pre-loaded list of accounts as seen in the following code:

```
from Exscript.util.start import start
from Exscript.util.file import get_hosts_from_file

def do_something(job, host, conn):
    conn.execute('uname -a')

accounts = [] # No account needed.
hosts = get_hosts_from_file('myhosts.txt')
start(accounts, hosts, do_something, max_threads=2)
```

Instead of passing in no account at all, you may also create one in the script:

```
from Exscript import Account
...
accounts = [Account('myuser', 'mypassword')]
...
```

Or you may load it from an external file:

```
from Exscript.util.file import get_accounts_from_file
...
```

(continues on next page)

(continued from previous page)

```
accounts = get_accounts_from_file('accounts.cfg')
...
```

Note that `accounts.cfg` is a config file with a defined syntax as seen in the API documentation for `Exscript.util.file.get_accounts_from_file()`.

6.2.7 Logging

Exscript has built-in support for logging. In a simple case, just pass the `stdout` and `stderr` parameters for log and errors to `start()` or `quickstart()` and you are done:

```
with open('log.txt','w+') as fp:
    start(accounts, hosts, do_something, stdout=fp)
```

Exscript creates one logfile per device. In the case that an error happened on the remote device, it creates an additional file that contains the error (including Python's traceback).

6.2.8 Interacting with a device

So far we only fired and forgot a command on a device, there was no true interaction. But Exscript does a lot to make interaction with a device easier. The first notable tool is `Exscript.util.match` - a module that builds on top of Python's regular expression support. Let's look at an example:

```
from Exscript.util.start import quickstart
from Exscript.util.match import first_match

def do_something(job, host, conn):
    conn.execute('uname -a')
    print "The response was", repr(conn.response)
    os, hostname = first_match(conn, r'^(\S+)\s+(\S+)')
    print "The hostname is:", hostname
    print "Operating system:", os

quickstart('ssh://localhost', do_something)
```

The experienced programmer will probably wonder what happens when `Exscript.util.match.first_match()` does not find a match. The answer is: It will return a tuple `None, None`. In other words, no matter what happens, the one liner can not fail, because `Exscript.util.match.first_match()` always returns a tuple containing the same number of elements as there are groups (bracket expressions) in the regular expression. This is more terse than the following typical regular idiom:

```
match = re.match(r'^(\S+)\s+(\S+)', conn.response)
if match:
    print match.group(1)
```

Similarly, the following use of `Exscript.util.match.any_match()` can never fail:

```
from Exscript.util.start import quickstart
from Exscript.util.match import any_match

def do_something(job, host, conn):
    conn.execute('ls -l')
    for permissions, filename in any_match(conn, r'^(\S+).*\s+(\S+)$'):
```

(continues on next page)

(continued from previous page)

```
        print "The filename is:", filename
        print "The permissions are:", permissions

quickstart('ssh://localhost', do_something)
```

`Exscript.util.match.any_match()` is designed such that it always returns a list, where each item contains a tuple of the same size. So there is no need to worry about checking the return value first.

6.2.9 Advanced queueing and reporting

`Exscript.Queue` is a powerful, multi-threaded environment for automating more complex tasks. It comes with features such as logging, user account management, and error handling that make things a lot easier. The above functions `Exscript.util.start.start()` and `Exscript.util.start.quickstart()` are just convenience wrappers around this queue.

In some cases, you may want to use the `Exscript.Queue` directly. Here is a complete example that also implements reporting:

```
#!/usr/bin/env python
from Exscript import Queue, Logger
from Exscript.util.log import log_to
from Exscript.util.decorator import autologin
from Exscript.util.file import get_hosts_from_file, get_accounts_from_file
from Exscript.util.report import status, summarize

logger = Logger()  # Logs everything to memory.


@log_to(logger)
@autologin()
def do_something(job, host, conn):
    conn.execute('show ip int brie')

# Read input data.
accounts = get_accounts_from_file('accounts.cfg')
hosts = get_hosts_from_file('hostlist.txt')

# Run do_something on each of the hosts. The given accounts are used
# round-robin. "verbose=0" instructs the queue to not generate any
# output on stdout.
queue = Queue(verbose=5, max_threads=5)
queue.add_account(accounts)      # Adds one or more accounts.
queue.run(hosts, do_something)   # Asynchronously enqueues all hosts.
queue.shutdown()                 # Waits until all hosts are completed.

# Print a short report.
print status(logger)
print summarize(logger)
```

6.2.10 Emulating a remote device

Exscript also provides a dummy protocol adapter for testing purposes. It emulates a remote host and may be used in place of the Telnet and SSH adapters:

```
from Exscript.protocols import Dummy
conn = Dummy()
...
```

In order to define the behavior of the dummy, you may define it by providing a Python file that maps commands to responses. E.g.:

```
def echo(command):
    return command.split(' ', 1)[1]

commands = (
('ls -l', """
-rw-r--r-- 1 sab nmc 1906 Oct  5 11:18 Makefile
-rw-r--r-- 1 sab nmc 1906 Oct  5 11:18 myfile
"""),

(r'echo [\r\n]+', echo)
)
```

Note that the command name is a regular expression, and the response may be either a string or a function.

6.3 CLI Tutorial

6.3.1 Introduction

With the *exscript* command line tool, you can quickly automate a conversation with a device over Telnet or SSH.

This is a step by step introduction to using the Exscript command line tool.

We'll assume that Exscript is already installed. You can confirm that your installation works by typing `exscript --version` into a terminal; if this prints the version number, your installation is complete.

6.3.2 Getting started

As a first simple test, let's try to connect to a Linux/Unix machine via SSH2, and execute the `uname -a` command on it.

Create a file named `test.exscript` with the following content:

```
uname -a
```

To run this Exscript template, just start Exscript using the following command:

```
exscript test.exscript ssh://localhost
```

Awesome fact: Just replace `ssh://` by `telnet://` and it should still work with Telnet devices.

Hint: The example assumes that `localhost` is a Unix server where SSH is running. You may of course change this to either an ip address (such as `ssh://192.168.0.1`), or any other hostname.

Exscript will prompt you for a username and a password, and connect to `localhost` using the entered login details. Once logged in, it executes `uname -a`, waits for a prompt, and closes the connection.

6.3.3 Running a script on multiple hosts

In practice, you may want to run this script on multiple hosts, and optimally at the same time, in parallel. Using the `-c` option, you tell Exscript to open multiple connections at the same time:

```
exscript -c 2 test.exscript ssh://localhost ssh://otherhost
```

`-c 2` tells Exscript to open two connections in parallel. So if you run this script, Exscript will again ask for the login details, and run `uname -a` for both hosts in parallel.

Note that the login details are only asked once and used on both hosts - this may or may not be what you want. The following section explains some of the details of using different login accounts.

6.3.4 Reading login information

Depending on how you would like to provide the login information, there are a few options. The first is by including it in the hostname:

```
exscript -c 2 test.exscript ssh://localhost
ssh://user:password@otherhost
```

In this case, Exscript still prompts the user for his login details, but the entered information is only used on hosts that do not have a user/password combination included in the hostname.

If you do not want for Exscript to prompt for login details, the `-i` switch tells Exscript to not ask for a user and password. You need to make sure that all hosts have a user and password in the hostname if you use it.

6.3.5 Reading host names from a text file

If you do not wish to hard code the host names or login details into the command, you may also list the hosts in an external file and load it using the `--hosts` option as follows:

```
exscript -c 2 --hosts myhosts.txt test.exscript
```

Note that *hosts.txt* is a file containing a list of hostnames, e.g.:

```
host1
host2
...
host20
```

6.3.6 Reading host names from a CSV file

Similar to the `--hosts`, you may also use `--csv-hosts` option to pass a list of hosts to Exscript while at the same time providing a number of variables to the script. The CSV file has the following format:

```
address my_variable another_variable
telnet://myhost value another_value
ssh://yourhost hello world
```

Note that fields are separated using the tab character, and the first line **must** start with the string "address" and is followed by a list of column names.

In the Exscript template, you may then access the variables using those column names:

```
ls -l $my_variable
touch $another_variable
```

6.3.7 Using Account Pooling

You can also pass in a pre-loaded list of accounts from a separate file. The accounts from the file are used for hosts that do not have a user/password combination specified in their URL.

```
exscript -c 2 --hosts myhosts.txt --account-pool accounts.cfg test.exscript
```

Note that `accounts.cfg` is a config file with a defined syntax as seen in the API documentation for `Exscript.util.file.get_accounts_from_file()`.

It is assumed that you are aware of the security implications of saving login passwords in a text file.

6.3.8 Logging

Exscript has built-in support for logging - just pass the `--logdir` or `-l` option with a path to the directory in which logs are stored:

```
exscript -l /tmp/logs -c 2 --hosts myhosts.txt --account-pool accounts.cfg test.
↪exscript
```

Exscript creates one logfile per device. In the case that an error happened on the remote device, it creates an additional file that contains the error (including Python's traceback).

6.3.9 Interacting with a device

So far we only fired and forgot a command on a device, there was no true interaction. But Exscript does a lot to make interaction with a device easier. The first notable tool is the `extract` keyword. Let's look at an example:

```
uname -a{extract /^(\S+)\s+(\S+)/ as os, hostname}
```

6.3.10 The Exscript Template Language

The Exscript template language is in some ways comparable to Expect, but has unique features that make it a lot easier to use and understand for non-developers.

A first example:

```
{fail "not a Cisco router" if connection.guess_os() is not "ios"}

show ip interface brief {extract /^(\S+)\s/ as interfaces}
configure terminal
{loop interfaces as interface}
    interface $interface
    description This is an automatically configured interface description!
    cdp enable
    no shut
    exit
{end}
copy running-config startup-config
```

Exscript templates support many more commands. Here is another example, to automate a session with a Cisco router:

```
show version {extract /^(cisco)/ as vendor}
{if vendor is "cisco"}
  show ip interface brief {extract /^(\S+)\s/ as interfaces}
  {loop interfaces as interface}
    show running interface $interface
    configure terminal
    interface $interface
    no shut
    end
  {end}
  copy running-config startup-config
{end}
```

6.3.11 Advanced Templates

Exscript templates support many more commands. For a full overview over the template language, please check *Exscript Templates*.

6.4 Command Line Options

6.4.1 Overview

You can pass parameters (or lists of parameters) into the templates and use them to drive what happens on the remote host. Exscript easily supports logging, authentication mechanisms such as TACACS and takes care of synchronizing the login procedure between multiple running connections.

These features are enabled using simple command line options. The following options are currently provided:

```
Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  --account-pool=FILE   Reads the user/password combination from the given
                        file instead of prompting on the command line. The
                        file may also contain more than one user/password
                        combination, in which case the accounts are used round
                        robin.
  -c NUM, --connections=NUM
                        Maximum number of concurrent connections. NUM is a
                        number between 1 and 20, default is 1.
  --csv-hosts=FILE      Loads a list of hostnames and definitions from the
                        given file. The first line of the file must have the
                        column headers in the following syntax: "hostname
                        [variable] [variable] ...", where the fields are
                        separated by tabs, "hostname" is the keyword
                        "hostname" and "variable" is a unique name under which
                        the column is accessed in the script. The following
                        lines contain the hostname in the first column, and
                        the values of the variables in the following columns.
  -d PAIR, --define=PAIR
                        Defines a variable that is passed to the script. PAIR
                        has the following syntax: STRING=STRING.
  --default-domain=STRING
```

(continues on next page)

(continued from previous page)

```
                      The IP domain name that is used if a given hostname
                      has no domain appended.
--delete-logs         Delete logs of successful operations when done.
-e EXSCRIPT, --execute=EXSCRIPT
                      Interprets the given string as the script.
--hosts=FILE          Loads a list of hostnames from the given file (one
                      host per line).
-i, --non-interactive
                      Do not ask for a username or password.
-l DIR, --logdir=DIR  Logs any communication into the directory with the
                      given name. Each filename consists of the hostname
                      with "_log" appended. Errors are written to a separate
                      file, where the filename consists of the hostname with
                      ".log.error" appended.
--no-echo             Turns off the echo, such that the network activity is
                      no longer written to stdout. This is already the
                      default behavior if the -c option was given with a
                      number greater than 1.
-n, --no-authentication
                      When given, the authentication procedure is skipped.
                      Implies -i.
--no-auto-logout      Do not attempt to execute the exit or quit command at
                      the end of a script.
--no-prompt           Do not wait for a prompt anywhere. Note that this will
                      also cause Exscript to disable commands that require a
                      prompt, such as "extract".
--no-initial-prompt   Do not wait for a prompt after sending the password.
--no-strip            Do not strip the first line of each response.
--overwrite-logs      Instructs Exscript to overwrite existing logfiles. The
                      default is to append the output if a log already
                      exists.
-p STRING, --protocol=STRING
                      Specify which protocol to use to connect to the remote
                      host. Allowed values for STRING include: dummy,
                      pseudo, ssh, ssh1, ssh2, telnet. The default protocol
                      is telnet.
--retry=NUM           Defines the number of retries per host on failure.
                      Default is 0.
--retry-login=NUM     Defines the number of retries per host on login
                      failure. Default is 0.
--sleep=TIME          Waits for the specified time before running the
                      script. TIME is a timespec as specified by the 'sleep'
                      Unix command.
--ssh-auto-verify     Automatically confirms the 'Host key changed' SSH
                      error  message with 'yes'. Highly insecure and not
                      recommended.
--ssh-key=FILE        Specify a key file that is passed to the SSH client.
                      This is equivalent to using the "-i" parameter of the
                      openssh command line client.
-v NUM, --verbose=NUM
                      Print out debug information about the network
                      activity. NUM is a number between 0 (min) and 5 (max).
                      Default is 1.
-V NUM, --parser-verbose=NUM
                      Print out debug information about the Exscript
                      template parser. NUM is a number between 0 (min) and 5
                      (max). Default is 0.
```

6.4.2 Using Account Pooling

It is possible to provide an account pool from which Exscript takes a user account whenever it needs to log into a remote host. Depending on the authentification mechanism used in your network, you may significantly increase the speed of parallel connections by using more than one account in parallel. The following steps need to be taken to use the feature:

1. Create a file with the following format:

```
[account-pool]
user=password
other_user=another_password
somebody=yet_another_password
```

   Note that the password needs to be base64 encrypted, just putting plain passwords there will NOT work.

2. Save the file. It is assumed that you are aware of the security implications of saving your login passwords in a text file.
3. Start Exscript with the `--account-pool FILE` option. For example:

```
exscript --account-pool /home/user/my_accounts my.exscript host4
```

6.4.3 Using a CSV file as input

By providing the –csv-hosts option you may pass a list of hosts to Exscript while at the same time providing a number of variables to the script. The CSV file should have the following format:

```
hostname my_variable another_variable
myhost value another_value
yourhost hello world
```

Note that fields are separated using the tab character, and the first line must start with the string “hostname” and is followed by a list of column names.

In the Exscript, you may then access the variables using those column names:

```
ls -l $my_variable
touch $another_variable
```

6.5 Exscript Templates

6.5.1 Simple example

The simplest possible template is one that contains only the commands that are sent to the remote host. For example, the following Exscript template can be used to retrieve the response of the ls -l and df commands from a unix host:

```
ls -l
df
```

6.5.2 Comments

Lines starting with a hash (“#”) are interpreted as comments and ignored. For example:

```
1. # This line is ignored...
2. {if __hostname__ is "test"}
3.   # ...and so is this one.
4. {end}
```

6.5.3 Using Variables

The following template uses a variable to execute the ls command with a filename as an argument:

```
ls -l $filename
```

When executing it from the command line, use:

```
exscript -d filename=.profile my.exscript localhost
```

Note that the -d switch allows passing variables into the template. The example executes the command ls -l .profile. You can also assign a value to a variable within a template:

```
{filename = ".profile"}
ls -l $filename
```

You may also use variables in strings by prefixing them with the "$" character:

```
1. {test = "my test"}
2. {if "my test one" is "$test one"}
3.   # This matches!
4. {end}
```

In the above template line 3 is reached. If you don't want the "$" character to be interpreted as a variable, you may prefix it with a backslash:

```
1. {test = "my test"}
2. {if "my test one" is "\$test one"}
3.   # This does not match
4. {end}
```

6.5.4 Adding Variables To A List

In Exscript, every variable is a list. You can also merge two lists by using the "append" keyword:

```
1. {
2.   test1 = "one"
3.   test2 = "two"
4.   append test2 to test1
5. }
```

This results in the "test1" variable containing two items, "one" and "two".

6.5.5 Using Built-in Variables

The following variables are available in any Exscript template, even if they were not explicitly passed in:

1. `__hostname__` contains the hostname that was used to open the current connection.

2. __response__ contains the response of the remote host that was received after the execution of the last command.

Built-in variables are used just like any other variable. You can also assign a new value to a built-in variable in the same way.

6.5.6 Using Expressions

An expression is a combination of values, variables, operators, and functions that are interpreted (evaluated) according to particular rules and that produce a return value. For example, the following code is an expression:

```
name is "samuel" and 4 * 3 is not 11
```

In this expression, *name* is a variable, *is*, *is not*, and * are operators, and *"samuel"*, *4*, *3*, and *11* are values. The return value of this particular expression is *true*.

In Exscript, expressions are used in many places, such as if-conditions or variable assignments. The following operators may be used in an expression.

Priority 1 Operators

1. * multiplies the operators (numerically).
2. / divides the operators (numerically).

Priority 2 Operators

1. + adds the operators (numerically).
2. − subtracts the operators (numerically).

Priority 3 Operators

1. . concatenates two strings.

Priority 4 Operators

1. `is` tests for equality. If both operators are lists, only the first item in the list is compared.
2. `is not` produces the opposite result from is.
3. `in` tests whether the left string equals any of the items in the list given as the right operator.
4. `not in` produces the opposite result from in.
5. `matches` tests whether the left operator matches the regular expression that is given as the right operator.
6. `ge` tests whether the left operator is (numerically) greater than or equal to the right operator.
7. `gt` tests whether the left operator is (numerically) greater than the right operator.
8. `le` tests whether the left operator is (numerically) less than or equal to the right operator.
9. `lt` tests whether the left operator is (numerically) less than the right operator.

Priority 5 Operators

1. `not` inverts the result of a comparison.

Priority 6 Operators

1. `and` combines two tests such that a logical AND comparison is made. If the left operator returns FALSE, the right operator is not evaluated.
2. `or` combines two tests such that a logical OR comparison is made. If the left operator returns TRUE, the right operator is not evaluated.

6.5.7 Using Hexadecimal Or Octal Numbers

Exscript also supports hexadecimal and octal numbers using the following syntax:

```
{
  if 0x0a is 012
    sys.message("Yes")
  else
    sys.message("No")
  end
}
```

6.5.8 Using Regular Expressions

At some places Exscript uses Regular Expressions. These are NOT the same as the expressions documented above, and if you do not know what regular expressions are it is recommended that you read a tutorial on regular expressions first.

Exscript regular expressions are similar to Perl and you may also append regular expression modifiers to them. For example, the following is a valid regular expression in Exscript:

```
/^cisco \d+\s+\w/i
```

Where the appended "i" is a modifier (meaning case-insensitive). A full explanation of regular expressions is not given here, because plenty of introductions have been written already and may be found with the internet search engine of your choice.

6.5.9 Built-in Commands

By default, any content of an Exscript template is sent to the remote host. However, you can also add instructions with special meanings. Such instructions are enclosed by curly brackets (`{` and `}`). The following commands all use this syntax.

Extracting Data From A Response

Exscript lets you parse the response of a remote host using regular expressions. If you do not know what regular expressions are, please read a tutorial on regular expressions first.

extract ... into ...

If you already know what regular expressions are, consider the following template:

```
ls -l {extract /^(d.*)/ into directories}
```

The extract command matches each line of the response of "ls -l" against the regular expression `/(d.*)/` and then appends the result of the first match group (a match group is a part of a regular expression that is enclosed by brackets) to the list variable named directories.

You can also extract the value of multiple match groups using the following syntax:

```
ls -l {extract /^(d\S+)\s.*\s(\S+)$/ into modes, directories}
```

This extracts the mode and the directory name from each line and appends them to the modes and directories lists respectively. You can also apply multiple matches to the same response using the following syntax:

```
ls -l {
  extract /^[^d].*\s(\S+)$/ into files
  extract /^d.*\s(\S+)$/    into directories
}
```

There is no limit to the number of extract statements.

extract ... into ... from ...

When used without the "from" keyword, "extract" gets the values from the last command that was executed. You may however also instruct Exscript to extract the values from a variable. The following example shows how this may be done.

```
ls -l {
  extract /^(.*)/  into lines
  extract /^(d.*)/ into directories from lines
}
```

extract ... as ...

The "as" keyword is similar to "into", the difference being that with as, the destination variable is cleared before new values are appended.

```
ls -l {extract /^(d.*)/ as directories}
```

"as" may be used anywhere where "into" is used.

If-Conditions

You can execute commands depending on the runtime value of a variable or expression.

if ... end

The following Exscript template executes the `ls` command only if `ls -l .profile` did not produce a result:

```
ls -l .profile {extract /(\.profile)$/ as found}
{if found is not ".profile"}
  ls
{end}
```

if ... else ... end

You can also add an else condition:

```
ls -l .profile {extract /(\.profile)$/ as found}
{if found is not ".profile"}
  ls
{else}
  touch .profile
{end}
```

if ... else if ...

You can perform multiple matches using else if:

```
ls -l .profile {extract /(.*profile)$/ as found}
{if found is ".profile"}
  ls
{else if found matches /my_profile/}
  ls -l p*
{else}
  touch .profile
{end}
```

Loops

Loops with counters

You can execute commands multiple times using the "loop" statement. The following Exscript template executes the "ls" command three times:

```
{number = 0}
{loop until number is 3}
  {number = number + 1}
  ls $directory
{end}
```

Similarly, the while statement may be used. The following script is equivalent:

```
{number = 0}
{loop while number is not 3}
  {number = number + 1}
  ls $directory
{end}
```

Another alternative is using the "loop from ... to ..." syntax, which allows you to specify a range of integers:

```
# Implicit "counter" variable.
{loop from 1 to 3}
  ls $directory$counter
{end}

# Explicit variable name.
{loop from 1 to 3 as number}
  ls $directory$number
{end}
```

Loops over lists

The following Exscript template uses the ls command to show the content of a list of subdirectories:

```
ls -l {extract /^d.*\s(\S+)$/ as directories}
{loop directories as directory}
  ls $directory
{end}
```

You can also walk through multiple lists at once, as long as they have the same number of items in it:

```
ls -l {extract /^(d\S+)\s.*\s(\S+)$/ as modes, directories}
{loop modes, directories as mode, directory}
  echo Directory has the mode $mode
  ls $directory
{end}
```

List loops can also be combined with the until or while statement seen in the previous section:

```
ls -l {extract /^d.*\s(\S+)$/ as directories}
{loop directories as directory until directory is "my_subdir"}
  ls $directory
{end}
```

Functions

Exscript provides builtin functions with the following syntax:

```
type.function(EXPRESSION, [EXPRESSION, ...])
```

For example, the following function instructs Exscript to wait for 10 seconds:

```
{sys.wait(10)}
```

For a list of supported functions please check here:

Exscript.stdlib package

Submodules

Exscript.stdlib.connection module

Exscript.stdlib.crypt module

Exscript.stdlib.file module

Exscript.stdlib.ipv4 module

Exscript.stdlib.list module

Exscript.stdlib.mysys module

Exscript.stdlib.string module

Exscript.stdlib.util module

Exiting A Script

fail "message"

The "fail" keyword may be used where a script should terminate immediately.

```
show something
{fail "Error: Failed!"}
show something else
```

In this script, the "show something else" line is never reached.

fail "message" if ...

It is also possible to fail only if a specific condition is met. The following snippet terminates only if a Cisco router does not have a POS interface:

```
show ip int brie {
  extract /^(POS)\S+/ as pos_interfaces
  fail "No POS interface found!" if "POS" not in pos_interfaces
}
```

Error Handling

Exscript attempts to detect errors, such as commands that are not understood by the remote host. By default, Exscript considers any response that includes one of the following strings to be an error:

```
invalid
incomplete
unrecognized
unknown command
[^\r\n]+ not found
```

If this default configuration does not suit your needs, you can override the default, setting it to any regular expression of your choice using the following function:

```
{connection.set_error(/[Ff]ailed/)}
```

Whenever such an error is detected, the currently running Exscript template is cancelled on the current host. For example, when the following script is executed on a Cisco router, it will fail because there is no ls command:

```
ls -l
show ip int brief
```

The "show ip int brief" command is not executed, because an error is detected at "ls -l" at runtime.

If you want to execute the command regardless, you can wrap the "ls" command in a "try" block:

```
{try}ls -l{end}
show ip int brief
```

You can add as many commands as you like in between a try block. For example, the following will also work:

```
{try}
  ls -l
  df
  show running-config
{end}
show ip int brief
```

6.6 Trouble Shooting

6.6.1 Common Pitfalls

Generally, the following kinds of errors that may happen at runtime:

1. **A script deadlocks.** In other words, Exscript sends no further commands even though the remote host is already waiting for a command. This generally happens when a prompt is not recognized.
2. **A script executes a command before the remote host is ready.** This happens when a prompt was detected where none was really included in the response.
3. **A script terminates before executing all commands.** This happens when two (or more) prompts were detected where only one was expected.

The following sections explain when these problems may happen and how to fix them.

6.6.2 Deadlocks

Exscript tries to automatically detect a prompt, so generally you should not have to worry about prompt recognition. The following prompt types are supported:

```
[sam123@home ~]$
sam@knip:~/Code/exscript$
sam@MyHost-X123$
MyHost-ABC-CDE123$
MyHost-A1$
MyHost-A1(config)$
FA/0/1/2/3$
FA/0/1/2/3(config)$
admin@s-x-a6.a.bc.de.fg:/$
```

Note: The trailing "$" may also be any of the following characters: "$#>%"

However, in some rare cases, a remote host may have a prompt that Exscript can not recognize. Similarly, in some scripts you might want to execute a special command that triggers a response that does not include a prompt Exscript can recognize.

In both cases, the solution includes defining the prompt manually, such that Exscript knows when the remote host is ready. For example, consider the following script:

```
1. show ip int brief
2. write memory
3. {enter}
4. show configuration
```

Say that after executing line 2 of this script, the remote host asks for a confirmation, saying something like this:

```
Are you sure you want to overwrite the configuration? [confirm]
```

Because this answer does not contain a standard prompt, Exscript can not recognize it. We have a deadlock. To fix this, we must tell Exscript that a non-standard prompt should be expected. The following change fixes the script:

```
1. show ip int brief
2. {connection.set_prompt(/\[confirm\]/)}
3. write memory
4. {connection.set_prompt()}
5. {enter}
6. show configuration
```

The second line tells Exscript to wait for "[confirm]" after executing the following commands. Because of that, when the write memory command was executed in line 3, the script does not deadlock (because the remote host's response includes "[confirm]"). In line 4, the prompt is reset to it's original value. This must be done, because otherwise the script would wait for another "[confirm]" after executing line 5 and line 6.

6.6.3 A Command Is Sent Too Soon

This happens when a prompt was incorrectly detected in the response of a remote host. For example, consider using the following script:

```
show interface descriptions{extract /^(\S+\d)/ as interfaces}
show diag summary
```

Using this script, the following conversation may take place:

```
1. router> show interface descriptions
2. Interface              Status         Protocol Description
3. Lo0                    up             up       Description of my router>
4. PO0/0                  admin down     down
5. Serial1/0              up             up       My WAN link
6. router>
```

Note that line 3 happens to contain the string "Router>", which looks like a prompt when it really is just a description. So after receiving the ">" character in line 3, Exscript believes that the router is asking for the next command to be sent. So it immediately sends the next command ("show diag summary") to the router, even that the next prompt was not yet received.

Note that this type of error may not immediately show, because the router may actually accept the command even though it was sent before a prompt was sent. It will lead to an offset however, and may lead to errors when trying to

capture the response. It may also lead to the script terminating too early.

To fix this, make sure that the conversation with the remote host does not include any strings that are incorrectly recognized as prompts. You can do this by using the "connection.set_prompt(...)" function as explained in the sections above.

6.6.4 The Connection Is Closed Too Soon

This is essentially the same problem as explained under "A Command Is Sent Too Soon". Whenever a prompt is (correctly or incorrectly) detected, the next command is send to the remote host. If all commands were already executed and the next prompt is received (i.e. the end of the script was reached), the connection is closed.

To fix this, make sure that the conversation with the remote host does not include any strings that are incorrectly recognized as prompts. You can do this by using the "connection.set_prompt(...)" function as explained in the sections above.

6.7 Exscript

6.7.1 Exscript package

Subpackages

Exscript.emulators package

Submodules

Exscript.emulators.command module

Exscript.emulators.iosemu module

Exscript.emulators.vdevice module

Exscript.protocols package

Subpackages

Exscript.protocols.drivers package

Submodules

Exscript.protocols.drivers.ace module

Exscript.protocols.drivers.adtran module

Exscript.protocols.drivers.aironet module

Exscript.protocols.drivers.aix module

Submodules

Exscript.util.buffer module

Exscript.util.cast module

Exscript.util.collections module

Exscript.util.crypt module

Exscript.util.daemonize module

Exscript.util.decorator module

Exscript.util.event module

Exscript.util.file module

Exscript.util.impl module

Exscript.util.interact module

Exscript.util.ip module

Exscript.util.ipv4 module

Exscript.util.ipv6 module

Exscript.util.log module

Exscript.util.mail module

Exscript.util.match module

Exscript.util.pidutil module

Exscript.util.report module

Exscript.util.sigint module

Exscript.util.sigintcatcher module

Exscript.util.start module

Exscript.util.syslog module

Exscript.util.template module

Exscript.util.tty module

Exscript.util.url module

Exscript.util.weakmethod module

Submodules

Exscript.account module

Exscript.host module

Exscript.key module

Exscript.logger module

Exscript.queue module

Exscript.version module