Service Manager Scripting Language

In This Chapter

This section describes the service manager scripting language and provides example scripts.

The following topics are included:

Service Manager Scripting Language

The service manager scripting language is based on Python version 2.4.2. Python has a set of language features (such as functions, lists, and dictionaries) and a very large set of packages which provide most of the Python functionality. By keeping the language features intact and drastically reducing the number of packages available, the operator is provided with a flexible, although small, scripting language.

The only feature removed from the Python language is unicode support. The only packages provided to the operator are:

- alc — Access to DHCP packet and export variables back to service manager.
- binascii — Common ASCII decoding like base64.
- re — Regular expression support.
- struct — Parse and manipulate binary strings.

The alc package contains only one object, DHCP, and has the following members:

Table 21: DHCP Object Members

Name	Read	Write	Class
htype	X		integer
hlen	X		integer
hops	X		integer
flags	X		integer
ciaddr	X		integer
yiaddr	X		integer
siaddr	X		integer
giaddr	X		integer
chaddr	X		string
sname	X		string
file	X		string
options	X		TLV
sub_ident		X	string
sub_profile_string		X	string
sla_profile_string		X	string
ancp_string		X	string

The TLV type provides easy access to the value part of a stream of type-length-value variables, as is the case for the DHCP option field. In example us5.py on page 1722, the circuit-ID is accessed as alc.dhcp.options[82][1].

Some DHCP servers do not echo the relay agent option (option 82) when the DHCP message was snooped instead of relayed. For the convenience of the operator, the relay agent option from the request message is returned when alc.dhcp.options[82] is called.

Python Changes

Some changes have been made to Python in order to run on an embedded system:

- No files or sockets can be opened from inside Python scripts.
- No system calls can be made from inside Python scripts nor is the posix package available.
- The maximum recursion depth is fixed to twenty.
- The total amount of dynamic memory available for Python itself and Python scripts is capped at 2MB.
- The size of the script source file must be less than 16KB.

Configuration

As an example consider script us5.py on page 1722 which sets the sub_ident variable based on the circuit ID of three different DSLAMs:

```
import re
import alc
import struct

# ASAM DSLAM circuit ID comes in three flavours:
#       FENT    string          "TLV1: ATM:3/0:100.33"
#       GELT    octet-stream    0x01010000A0A0A0A0000000640022
#       GENT    string          "ASAM11 atm 1/1/01:100.35"
#
# Script sets output ('subscriber') to 'sub-vpi.vci', e.g.: 'sub-100.33'.

circuitid = str(alc.dhcp.options[82][1])

m = re.search(r'(\d+\.\d+)$', circuitid)
if m:
        # FENT and GENT
        alc.dhcp.sub_ident = "sub-" + m.group()
elif len(circuitid) >= 3:
        # GELT
        # Note: what byte order does GELT use for the VCI?
        # Assume network byte (big endian) order for now.
        vpi = struct.unpack('B', circuitid[-3:-2])[0]
        vci = struct.unpack('>H', circuitid[-2:])[0]
        alc.dhcp.sub_ident = "sub-%d.%d" % (vpi, vci)
```

Configure the url to this script in a sub-ident-policy as follows:

```
-----------------------------------------------
        sub-ident-policy "DSLAM" create
            description "Parse circuit IDs from different DSLAMs"
            primary
                script-url "ftp://xxx.xxx.xxx.xx/py/us5.py"
                no shutdown
            exit
        exit
-----------------------------------------------
```

And attach this sub-ident-policy to the sub-sla-mgmt from a SAP:

```
A:dut-A>config>service>vpls>sap# info
----------------------------------------------
                dhcp
                    description "client side"
                    lease-populate 50
                    no shutdown
                exit
                anti-spoof ip-mac
                sub-sla-mgmt
                    sub-ident-policy "DSLAM"
                    no shutdown
                exit
----------------------------------------------
```

Note that DHCP snooping/relaying should be configured properly in order for this to work.

Operator Debugging

Verbose debug output is sent to debug-trace on compile errors, execution errors, execution output and the exported result variables.

```
A:dut-A>config>subscr-mgmt>sub-ident-pol>primary# script-url "ftp://xxx.xxx.xx.xx/py/
parsefail1.py"
A:dut-A>config>subscr-mgmt>sub-ident-pol>primary# no shutdown

1 2006/07/30 01:17:33.14 UTC MINOR: DEBUG #2001 - Python Compile Error
"Python Compile Error: parsefail1.py
  File "ftp://xxx.xxx.xx.xx/py/parsefail1.py", line 2
  def invalid_function():
      ^
IndentationError: expected an indented block
"

A:dut-A>config>subscr-mgmt>sub-ident-pol>primary# script-url "ftp://xxx.xxx.xx.xx/py/
dump.py"

2 2006/07/30 01:24:55.50 UTC MINOR: DEBUG #2001 - Python Output
"Python Output: dump.py
htype    = 0
hlen     = 0
hops     = 0
flags    = 0
ciaddr   = '0.0.0.0'
yiaddr   = '0.0.0.0'
siaddr   = '0.0.0.0'
giaddr   = '0.0.0.0'
chaddr   = ''
sname    = ''
file     = ''
options  = '5\x01\x056\x04\n\x01\x07\n3\x04\x00\x00\x00\xb4\x01\x04\xff\xff\xff
\x00\x1c\x04\n\x02\x02\xffR\x0f\x01\rdut-A|1|1/1/1\xff'
"

3 2006/07/30 01:24:55.50 UTC MINOR: DEBUG #2001 - Python Result
"Python Result: dump.py
"

A:dut-A>config>subscr-mgmt>sub-ident-pol>primary# script-url "ftp://xxx.xxx.xx.xx/py/end-
less.py"

4 2006/07/30 01:30:17.27 UTC MINOR: DEBUG #2001 - Python Output
"Python Output: endless.py
"

5 2006/07/30 01:30:17.27 UTC MINOR: DEBUG #2001 - Python Error
"Python Error: endless.py
Traceback (most recent call last):
  File "ftp://xxx.xxx.xx.xx/py/endless.py", line 2, in ?
FatalError: script interrupted (timeout)
"
```

Note that all the Python Result events are empty because none of the scripts set any of the output variables.

Python Scripts

Note that the scripts in this section are test scripts and not scripts which the operator would normally use.

dump.py

```
from alc import dhcp

def print_field(key, value):
        print '%-8s = %r' % (key, value)

def ipaddr2a(ipaddr):
        return '%d.%d.%d.%d' % (
                (ipaddr & 0xFF000000) >> 24,
                (ipaddr & 0x00FF0000) >> 16,
                (ipaddr & 0x0000FF00) >>  8,
                (ipaddr & 0x000000FF))

print_field('htype',    dhcp.htype)
print_field('hlen',     dhcp.hlen)
print_field('hops',     dhcp.hops)
print_field('flags',    dhcp.flags)
print_field('ciaddr',   ipaddr2a(dhcp.ciaddr))
print_field('yiaddr',   ipaddr2a(dhcp.yiaddr))
print_field('siaddr',   ipaddr2a(dhcp.siaddr))
print_field('giaddr',   ipaddr2a(dhcp.giaddr))
print_field('chaddr',   dhcp.chaddr)
print_field('sname',    dhcp.sname)
print_field('file',     dhcp.file)
print_field('options',  str(dhcp.options))
```

us5.py

```
import re
import alc
import struct

# ASAM DSLAM circuit ID comes in three flavours:
#       FENT    string          "TLV1: ATM:3/0:100.33"
#       GELT    octet-stream    0x01010000A0A0A0A0000000640022
#       GENT    string          "ASAM11 atm 1/1/01:100.35"
#
# Script sets output ('subscriber') to 'sub-vpi.vci', e.g.: 'sub-100.33'.

circuitid = str(alc.dhcp.options[82][1])

m = re.search(r'(\d+\.\d+)$', circuitid)
if m:
        # FENT and GENT
        alc.dhcp.sub_ident = "sub-" + m.group()
elif len(circuitid) >= 3:
        # GELT
        # Note: what byte order does GELT use for the VCI?
        # Assume network byte (big endian) order for now.
        vpi = struct.unpack('B', circuitid[-3:-2])[0]
        vci = struct.unpack('>H', circuitid[-2:])[0]
        alc.dhcp.sub_ident = "sub-%d.%d" % (vpi, vci)
```

Limitations

'%' operator — While %f is supported, %g and %e are not supported.

Floating Point Arithmetic — The floating point arithmetic precision on the box is less than the precision required by the regression suites of Python. For example, pow(2., 30) equals to 1024.*1024.*1024. until five numbers after the point instead of seven and sqrt(9) equals to 3. for the first seven numbers after the point.

Using the round operator fixes these problems. For example, round(pow(2., 30)) equals round(1024.*1024.*1024.) and round(sqrt(9)) equals 3.

RADIUS Script Policy Overview

Python scripts for RADIUS AAA packets support manipulation in subscriber management application. This feature is supported on 7750 SRs and 7450 ESSes in mixed mode. A Python script can be executed in following cases:

- Before the system sends an access-request packet.
- After the system receives an access-accept packet.
- After the system receives an CoA-request packet.
- Before the system sends an accounting-request packet.

The input of the script is the corresponding original packet; and the output of packet will be used as the new corresponding packet for further ESM AAA process.

The **radius-script-policy** contains URLs of a primary and optionally a secondary Python script, which could be a local CF file path or a FTP URL. the configured radius-script-policy could be used in different ESM polices like authentication-policy or radius-accounting-policy.

The following operations are supported within the script:

- Get the value of an existing attribute or VSA.
- Modify the value of an existing attribute or VSA.
- Add a new attribute or VSA.
- Remove an existing attribute or VSA.

Note: The following RADIUS attributes or VSA are read-only to Python script:

- Message-Authenticator
- Alc-LI-Action
- Alc-LI-Direction
- Alc-LI-Destination
- Alc-LI-FC
- Alc-LI-Intercept-Id
- Alc-LI-Session-Id

Python Changes

New Python objects: alc.radius.attributes, have the following methods:

- get(type) : return the first attribute with specified type as string
- getTuple(type) : same as above but return a tuple of strings
- getVSA(vendor, type) : return the first VSA as string
- getVSATuple(vendor, type) : same as above but return a tuple of strings
- set(type, value) : set the specified attribute to the the value, value must be either string or tuple of strings
- setVSA(vendor, type, value) : set the specified VSA to the the value, value must be either string or tuple of strings
- clear(type) : remove the specified attribute
- clearVSA(vendor, type) : remove the specified VSA

Sample Script

```
From alc import radius
#1. Get the value of an existing Attribute
Username=radius.attributes.get(1)
#2. Modify an existing attribute
radius.attributes.set(1, 'Tom')

#3. Remove an existing attribute
radius.attributes.clear(1)
#4. Add a new attribute
radius.attributes.set(126,”WIFI-operator”)
```

Tips and Tricks

- Use xrange() instead of range().
- Avoid doing a lot of string operations. Following scripts provide the same output:

```
# This script takes 2.5 seconds.
s = ""
for c in 'A'*50000:
     s += str(ord(c)) + ', '
print '[' + s[:-2] + ']'

# This script takes 0.1 seconds.
print map(ord, 'A'*50000)
```

Sample Python Scripts

This section provides examples to show how the script can be used in the context of Enhanced Subscriber Management.

Note that these scripts are included for informational purposes only. Operator must customize the script to match their own network and processes..

Example

This script uses the IP address assigned by the DHCP server to derive both *sub_ident* and *sla_profile_string*.

Script:

```
1.   import alc
2.   yiaddr = alc.dhcp.yiaddr
3.   # Subscriber ID equals full client IP address.
4.   # Note: IP address 10.10.10.10 yields 'sub-168430090'
5.   # and not 'sub-10.10.10.10'
6.   alc.dhcp.sub_ident = 'sub-' + str(yiaddr)
7.   # DHCP server is configured such that the third byte (field) of the IP
8.   # address indicates the session Profile ID.
9.   alc.dhcp.sla_profile_string = 'sp-' + str((yiaddr & 0x0000FF00) >> 8)
```

Explanation:

Line 1: Imports the library “alc” – Library imports can reside anywhere in the script as long as the items are imported before they are used.

Line 2: Assigns the decimal value of the host’s IP address to a temporary variable “yiaddr”.

Line 6: The text “sub_“ followed by yiaddr is assigned to “sub_ident” string.

Line 9: The text “sp-“ followed with the third byte of the IP address is assigned to the “sla-profile” string.

If this script is run, for example, with a DHCP assigned IP address of :

```
yiaddr = 10.10.0.2
```

The following variables are returned:

```
sub_ident: sub-168427522(hex = A0A00002 = 10.10.0.2)
sla_ident: sp-0
```

Example

This script returns the *sub_profile_string* and *sla_profile_string*, which are coded directly in the Option 82 string

Script:

```
1.   import re
2.   import alc
3.   # option 82 formatted as follows:
4.   # "<subscriber Profile>-<sla-profile>"
5.   ident = str(alc.dhcp.options[82][1])
6.   alc.dhcp.sub_ident = ident
7.   tmp = re.match("(?P<sub>.+)-(?P<sla>.+)", str(ident))
8.   alc.dhcp.sub_profile_string = tmp.group("sub")
9.   alc.dhcp.sla_profile_string = tmp.group("sla")
```

Explanation:

Line 1-2: Import the libraries “re” and “alc”. Library imports can reside anywhere in the script as long as the items are imported before they are used.

Line 6: Assigns the full contents of the DHCP Option 82 field to the “sub_ident” variable.

Line 7: Splits the options 82 string into two parts, separated by “-”.

Line 8: Assigns the first part of the string to the variable “sub_profile_string”.

Line 9: Assigns the second part of the string to the variable “sla_profile_string”.

.

If this script is run, for example, with DHCP option field:

```
options = \x52\x0D\x01\0x0Bmydsl-video
```

The following variables are returned:

```
sub_ident: mydsl-video
sub_profile_string: mydsl
sla_profile_string: video
```

Example

This script parses the option 82 “circuit-id” info inserted in the DHCP packet by a DSLAM, and returns the *sub_ident* string.

Script:

```
1.   import re
2.   import alc
3.   import struct
4.   # Alcatel 7300 ASAM circuit ID comes in three flavours:
5.   #       FENT    string          "TLV1: ATM:3/0:100.33"
6.   #       GELT    octet-stream    0x01010000A0A0A0A0000000640022
7.   #       GENT    string          "ASAM11 atm 1/1/01:100.35"
8.   #
9.   # Script sets output ('subscriber') to 'sub-vpi.vci',
10.  # e.g.: 'sub- 100.33'.
11.  circuitid = str(alc.dhcp.options[82][1])
12.  m = re.search(r'(\d+\.\d+)$', circuitid)
13.  if m:
14.     # FENT and GENT
15.     alc.dhcp.sub_ident = "sub-" + m.group()
16.  elif len(circuitid) >= 3:
17.     # GELT
18.     # Note: GELT uses network byte (big endian) order for the VCI
19.     vpi = struct.unpack('B', circuitid[-3:-2])[0]
20.     vci = struct.unpack('>H', circuitid[-2:])[0]
21.     alc.dhcp.sub_ident = "sub-%d.%d" % (vpi, vci)
```

Explanation:

Line 1-2 Import the libraries “re” and “alc” – Library imports can reside anywhere in the script as long as the items are imported before they are used. Needed if regular expressions are used.

Line 3: Imports the “struct” library – needed if regular expressions are used.

Line 11: Assigns the contents of the DHCP Option 82 Circuit-ID field to a temporary variable called “circuitid”.

Line 12: Parses the circuitid and checks for the existence of the regular expression “digit.digit” at the end of the string.

Line 15: If found, a string containing the text “sub-“ followed by these two digits is assigned to the variable “sub-ident” .

Line 16: If not found, and the length of circuit-id is at least 3.

Line 19: Parses the circuitid and assigns the third-last byte to the temporary variable “vpi”.

Line 20: Parses the circuitid and assigns the last two bytes to the temporary variable “vci”.

Line 21: Assigns a string containing the text “sub-“ followed by vpi and vci to the variable “sub-ident”.

If this script is run, for example, with DHCP option field (assigned by an ASAM with FENT card) containing:

```
options = \x52\x16\x01\x14TLV1: ATM:3/0:100.33
```

(in decimal: 80, 22, 1, 20TLV…)

The following variables are returned:

```
sub_ident: sub-100.33
```

If the above script is run, for example, with a DHCP option field (assigned by an ASAM with GELT card) containing

```
options = \x52\x10\x01\x0E\x01\x01\x00\x00\xA0\xA0\xA0\xA0\x00\x00\x00\x64 \x00\x22
```

(in decimal: 82, 16, 1, 15, 1, 1, 0, 0, 160, 160, 160, 160, 0, 0, 0, **100**, 0, **34**; corresponding to VPI 100, VCI 34)

Python returns the following variables:

```
sub_ident: sub-100.34
```

If the above script is run, for example, with a DHCP option field (assigned by an ASAM with GENT card) containing

```
options = \x52\x1A\x01\x18ASAM11 atm 1/1/01:100.35
```

The following variables are returned

```
sub_ident: sub-100.35
```