# Perl, Perl References, and SNMP

## *A Quick Guide to Using Perl with SNMP*
## *Includes a description of References in Perl*

Nick Urbanik <nicku(at)nicku.org> 

A computing department



# Advantages of Each Module — 1

- Net::SNMP
  - Slowest of the three
  - Easiest to install (from CPAN or by ppm)
  - Supports SNMPv3, plus new encryption schemes: Triple DES, AES
  - Very good documentation
  - Good, simple to use extensions Net::SNMP::HostInfo, Net::SNMP::Interfaces



# The Choices

- There are *three main choices* (but also many more!) if you want to write SNMP code in Perl:
  - The Perl interface to the Net-SNMP library, called SNMP
    - What you have when you install the software packages net-snmp-perl and net-snmp-devel on Fedora or Red Hat Linux
  - The Net::SNMP Perl library, together with Net::SNMP::HostInfo and Net::SNMP::Interfaces
    - Install from CPAN—see slide§5
  - Finally we have SNMP_Session, from http://www.switch.ch/misc/leinen/snmp/perl/ used in the cricket application (at http://cricket.sourceforge.net/)



# Advantages of Each Module — 2

- SNMP
  - Faster than the others, since it is linked against C libraries
  - Supports SNMPv3
  - Harder to install on some platforms, e.g., Windows
- SNMP_Session
  - Written in pure Perl, so very portable
  - Fast, portable, used for Cricket, MRTG
  - Supports only SNMPv1 and SNMPv2c

# CPAN

- See slide §27 in the Perl lecture notes
- To install `Net::SNMP`:

```
$ sudo perl -MCPAN -e shell
cpan shell -- CPAN exploration and modules installation (v1.7601)
ReadLine support enabled

cpan> install Net::SNMP Net::SNMP::HostInfo Net::SNMP::Interfaces
Running install for module Net::SNMP
Running make for D/DT/DTOWN/Net-SNMP-4.1.2.tar.gz
...
```

- List all CPAN modules relating to SNMP:

```
cpan> m /SNMP/
```



# PPM: installing on Windows

- Just as easy to install into ActiveState Perl:

```
D:\>ppm
PPM - Programmer's Package Manager version 3.1.
Copyright (c) 2001 ActiveState SRL. All Rights Reserved.
...

ppm> install Net-SNMP
====================
Install 'Crypt-DES' version 2.03 in ActivePerl 5.8.1.807.
====================
...
```

- List all SNMP Perl packages:

```
ppm> search snmp
Searching in Active Repositories
   1. Apache-WebSNMP       [0.11] Embed SNMP get statements into HTML
...
```



# References

Used with Nested Data Structures

. . . and with OO programming

A Detour needed to understand Net::SNMP and Net::LDAP



# References

- I tried to avoid telling you about references, but you really need to know about them here
- Needed to implement nested data structures, i.e., arrays of arrays, or arrays of hashes of arrays of arrays of hashes. . .
- An array element or a hash element must be a scalar
- A *reference* is a scalar variable that refers to existing data
  - . . . Such as an entire array or an entire hash (or to just about anything else)
  - Rather like a pointer in C
  - Java implements references behind the scenes; Perl and C++ make them explicit

# The backslash Operator

- Create a reference to an existing variable using '\'
- Like "address-of" operator '&' in C
- Examples:

```
$scalar_ref = \$foo;
$array_ref = \@array;
$hashref = \%hash;
$subroutine_ref = \&subroutine;
```



# Anonymous Array References

- Often needed when constructing nested structures (and with Net::SNMP)
- Use square brackets:

```
my $array_ref = [ 1, 2, [ 'a', 'b', 'c' ] ];
```

- Here we made a reference to an anonymous array of three elements, the last element of which is another anonymous array of three elements.



# Anonymous Hash References

- Use braces:

```
my $hash_ref = {
    Adam => 'Eve',
    Clyde => 'Bonnie',
};
```

- See `perldoc perldsc` for cookbook examples of how to build arrays of hashes, arrays of arrays, hashes of hashes, hashes of arrays, . . .



# Using a Reference

- Three methods: use a reference as the variable name, use block, use arrow method.
- First method: Access the value of a reference by putting the "right" funny character in front of the reference
- Examples:

```
print "value is $$scalar_ref\n";
foreach ( @$array_ref ) {
    print;
}
foreach my $key ( sort keys %$hash_ref ) {
    print "$key => $$hash_ref{$key}\n";
}
my $return_value = &$subroutine_ref( 1, 2 );
```

# Using a Block as a Variable Name

- Method 2: use a block returning a reference where a variable name would be.
  - Block returns a value either with the `return` statement, or returns the last value appearing in the block
- Examples:

```
print "value is ${$scalar_ref}\n";
foreach ( @{$array_ref} ) {
    print;
}
my $return_value = &{$subroutine_ref}( 1, 2 );
${$array_ref}[0] = "January";
${$hash_ref}{key} = "value";
```



# The Arrow Operator

- Method 3: Used when reference expression complicated; often used to call *methods* in OO
  - ... because a Perl object *is* a reference
- The following are equivalent:

```
$ $array_ref  [0] = "January";
${$array_ref} [0] = "January";
  $array_ref->[0] = "January";
```

  - So are these:

```
$ $hash_ref  {key} = "value";
${$hash_ref} {key} = "value";
  $hash_ref->{key} = "value";
```



# Documentation About References

- The following documentation is provided with Perl:
  - `perldoc perlreftut` is a short tutorial introduction to references
  - `perldoc perldsc` is a tutorial cookbook with lots of examples showing how to build complex nested data structures
  - `perldoc perlref` is the complete manual for Perl references
- I suggest read them in that order
- We have finished looking at References — Now we return to `Net::SNMP`



# The `Net::SNMP` Package

- Provides an object-oriented interface to SNMP
- One `Net::SNMP` object corresponds to one remote SNMP agent or manager
- Each `Net::SNMP` object has either *blocking* or *non-blocking* properties
  - Blocking: methods do not return until response received or timeout
  - Non-blocking: queue requests. Response comes back via a *callback* routine.
  - We only cover blocking operations here because they are simpler, but do not be afraid to read the good documentation and use other methods; see slide §31

# Results of Method Calls

- Methods that require a response return a *hash reference* containing query results
  - Returns undefined value on failure
  - `error()` method shows cause of failure
- *key* in hash is dotted OID
- *value* is, well, the value returned for that OID
- The hash reference can also be obtained using the `var_bind_list()` method



# Named Method Parameters

- Method parameters used a dashed-option naming style:

```
$object->method( -argument => $value );
```

- Use a hash as the parameter list, a common ideom in Perl
- The "=>" is just the quoting comma commonly used in initialising hashes, as explained in slide §44 in version 1.6 of my Perl slides



# `session()` — Create New Session

```
($session, $error) = Net::SNMP->session(
     [-hostname     => $hostname,]
     [-port         => $port,]
     [-localaddr    => $localaddr,]
     [-localport    => $localport,]
     [-nonblocking  => $boolean,]
     [-version      => $version,]
     [-timeout      => $seconds,]
     [-retries      => $count,]
     [-maxmsgsize   => $octets,]
     [-translate    => $translate,]
     [-debug        => $bitmask,]
     [-community    => $community,]    # v1/v2c
     [-username     => $username,]     # v3
     [-authkey      => $authkey,]      # v3
     [-authpassword => $authpasswd,]   # v3
     [-authprotocol => $authproto,]    # v3
     [-privkey      => $privkey,]      # v3
     [-privpassword => $privpasswd,]   # v3
     [-privprotocol => $privproto,]    # v3
 );
```



# `session()` — Parameters

- `-version` parameter indicates security model
  - Can be `1`, `2` or `3`, or the strings `'snmpv1'`, `'snmpv2c'` or `'snmpv3'`
- Read the section *User-based Security Model Arguments* in `perldoc Net::SNMP` for details about `-username`, `-authkey`, `-authpassword`, `-privkey`, `-privpassword`, `-authprotocol`, `-privprotocol` parameters

# get_request()

- send a SNMP `get-request` to the remote agent

```
$result = $session->get_request(
   [-contextengineid => $engine_id,] # v3
   [-contextname     => $name,]      # v3
   -varbindlist      => \@oids,
);
```

- The square brackets mean the parameter is optional
- The parameter `-varbindlist` is a reference to a list of OIDs
- This is a reference to an array of strings that give the full numerical OID of an SNMP variable, such as `'1.3.6.1.2.1.1.3.0'`.
  - See example in slide 29



# get_next_request()

- send a SNMP `get-next-request` to the remote agent

```
$result = $session->get_next_request(
   [-contextengineid => $engine_id,] # v3
   [-contextname     => $name,]      # v3
   -varbindlist      => \@oids,
);
```



# set_request()

- send a SNMP `set-request` to the remote agent

```
$result = $session->set_request(
   [-contextengineid => $engine_id,] # v3
   [-contextname     => $name,]      # v3
   -varbindlist      => \@oid_value,
);
```

- The `-varbindlist` parameter is a reference to an array of three values: a string representing the full numerical OID, the type, and the value.
- Here is an example:

```
my $sysContact = '1.3.6.1.2.1.1.4.0';
my $result = $session->set_request(
   -varbindlist => [ $sysContact, OCTET_STRING, 'Help Desk x911' ]
);
```



# trap()

- send an SNMP `trap` to the remote manager

```
$result = $session->trap(
   [-enterprise      => $oid,]
   [-agentaddr       => $ipaddress,]
   [-generictrap     => $generic,]
   [-specifictrap    => $specific,]
   [-timestamp       => $timeticks,]
   -varbindlist      => \@oid_value,
);
```

# get_bulk_request()

- send a `get-bulk-request` to the remote agent

```
$result = $session->get_bulk_request(
   [-contextengineid => $engine_id,] # v3
   [-contextname     => $name,]      # v3
   [-nonrepeaters    => $non_reps,]
   [-maxrepetitions  => $max_reps,]
   -varbindlist      => \@oids,
);
```



# snmpv2_trap()

- send a `snmpV2-trap` to the remote manager

```
$result = $session->snmpv2_trap(
   -varbindlist      => \@oid_value,
);
```

- Only works with `-version` corresponding to `'snmpv2c'`
- Net::SNMP does not support using this with `snmpv3`



# inform_request()

- send an `inform-request` to the remote manager

```
$result = $session->inform_request(
   [-contextengineid => $engine_id,] # v3
   [-contextname     => $name,]      # v3
   -varbindlist      => \@oid_value,
);
```



# error(), close()

- `error()` — get the current error message from the object
  - This method returns a text string explaining the reason for the last error.
  - An empty string is returned if no error has occurred.

```
$error_message = $session->error;
```

- `close()` — clear the Transport Layer associated with the object

```
$session->close;
```

# Example: `get-request` — 1

```
use Net::SNMP;

my ($session, $error) = Net::SNMP->session(
   -hostname  => shift || 'localhost',
   -community => shift || 'public',
   -port      => shift || 161
);

if (!defined($session)) {
   printf("ERROR: %s.\n", $error);
   exit 1;
}

my $sysUpTime = '1.3.6.1.2.1.1.3.0';

my $result = $session->get_request(
   -varbindlist => [ $sysUpTime ]
);
```



# Example: `get-request` — 2

```
if (!defined($result)) {
   printf("ERROR: %s.\n", $session->error);
   $session->close;
   exit 1;
}

printf("sysUpTime for host '%s' is %s\n",
   $session->hostname, $result->{$sysUpTime}
);

$session->close;

exit 0;
```



# Documentation for `Net::SNMP`

- The `perldoc` documentation for `Net::SNMP` is good, and with examples. Read the following documents using either `perldoc` or, on Linux, `man` if you like:
  - `Net::SNMP`
  - `Net::SNMP::HostInfo`
  - `Net::SNMP::HostInfo::IpAddrEntry`
  - `Net::SNMP::HostInfo::IpNetToMediaEntry`
  - `Net::SNMP::HostInfo::IpRouteEntry`
  - `Net::SNMP::HostInfo::TcpConnEntry`
  - `Net::SNMP::HostInfo::UdpEntry`
  - `Net::SNMP::Interfaces`
  - `Net::SNMP::Interfaces::Details`



# Docs: `NetSNMP`, `SNMP_Session`

- For Net-SNMP (Yes, names are a little confusing; I mean the software from `http://net-snmp.sourceforge.net/`), read:
  - `SNMP`
  - `NetSNMP::ASN`
  - `NetSNMP::OID`
- For `SNMP_Session` and `BER`, see `http://www.switch.ch/misc/leinen/snmp/perl/`