



Lecture 8

RHN Application Programming Interface

Upon completion of this unit, you should be able to:

- Describe the practical uses for the Red Hat Network API
- Give an overview of basic RHN API program structure
- Modify and write simple programs using the RHN API



- Application programming interface provided by RHN
- Red Hat Network API uses:
 - Custom queries
 - Reporting
 - Automated RHN administration
- Red Hat Network API functionality continues to expand
- Every Satellite server has relevant API documentation
 - `http://satellite.fqdn/rhn/apidoc/`

The Red Hat Network Application Programming Interface (API) provides a mechanism that allows programmers to write programs which interact with a RHN Satellite Server. Many tasks that can be performed through the web user interface can also be accomplished by the methods provided by the RHN API.

The API extends the functionality of RHN because it allows scripts to replace repetitive tasks that would be extremely difficult to do using the web interface. This added functionality facilitates automation and scalability within the enterprise.



- Works with languages with XML-RPC client support
- Most common languages used are Perl and Python
- Perl
 - Requires `perl-Frontier-RPC` package
- No additional requirements for Python

XML-RPC is a client/server remote procedure call communications protocol that uses XML tags to encode its messages. It uses the HTTP protocol as its transport mechanism.

Perl scripts that interact with the Red Hat Network API typically use the `Frontier::Client` modules provided by the `perl-Frontier-RPC` package. The `perl-Frontier-RPC` RPM is not provided as part of the standard Red Hat Enterprise Linux channels. It is provided as part of the Red Hat Network Satellite software channel.

The `python` RPM provides, `xmlrpclib`, a library that implements XML-RPC client support. Since it is part of the standard Python installation, no additional packages are required to write Python scripts that use the Red Hat Network API.



- Programs which use the RHN API have a consistent pattern
- Connect to the Satellite server via XML-RPC library
- In most cases, authenticate as a valid user
 - Normal Satellite server permissions/roles apply
- Perform queries and operations of interest
- Logout (only when authenticated)

Before calling any Red Hat Network API methods, an XML-RPC connection must be established with the RHN server. This is the scripted parallel to using a URL with a browser to connect to the server.

Next the `login` method in the `auth` namespace is called with a valid RHN login and password to authenticate into the RHN server. The session key returned by the method is passed as an argument to other method calls. This session key represents an authenticated user's access to the RHN server and determines which privileges and access is given to the other methods. For example, many of the `user` namespace methods will only work with a session key generated by the successful authentication by the Satellite Administrator or Organization Administrator to access information about other RHN users on the system. Also queries and changes will be made within the organization of the authenticated user.



API Method Namespaces

RHN Software/Server Admin

8-4

- `api`
 - Provides `getVersion` and `systemVersion` methods
- `preferences`
 - Methods for locale and timezone configuration
- `proxy`
 - Provides methods to manage RHN Proxies
- `satellite`
 - Provides RHN Satellite management methods



API Method Namespaces

User/Organization Management

8-5

- `auth`
 - Provides `login` and `logout` methods
- `org`
 - Provides methods for Organization management
- `user`
 - Provides methods for RHN user administration



API Method Namespaces

Software and Configuration Management

8-6

- `channel`
 - Provides methods for managing Software Channels
- `configchannel`
 - Provides methods for Configuration Channel management
- `errata`
 - Provides methods to manage RHN errata
- `packages`
 - Methods that search for and deletes packages within RHN

For use only by a student enrolled in a Red Hat training course taught by Red Hat, Inc. or a Red Hat Certified Training Partner. No part of this publication may be photocopied, duplicated, stored in a retrieval system, or otherwise reproduced without prior written consent of Red Hat, Inc. If you believe Red Hat training materials are being improperly used, copied, or distributed please email training@redhat.com or phone toll-free (USA) +1 (866) 626 2994 or +1 (919) 754 3700.



API Method Namespaces

Systems Management

8-7

- `activationkey`
 - Provides methods for managing Activation Keys
- `kickstart`
 - Provides methods to manage kickstart profiles
- `schedule`
 - Methods to search and manage scheduled events
- `system`
 - Provides methods for queries and management of registered systems
- `systemgroup`
 - Provides methods for System Group administration



```
#!/usr/bin/perl
use Frontier::Client;

my $URL = 'https://satellite.example.com/rpc/api';
my $user = 'rhn-username';
my $pass = 'rhn-password';

my $client = new Frontier::Client(url => $URL);
my $session = $client->call('auth.login', $user, $pass);

my $systems = $client->call('system.listUserSystems',
                           $session);
foreach my $system (@$systems) {
    print $system->{'name'}."\n";
}
$client->call('auth.logout', $session);
```

The Perl script above lists the systems the user named `rhn-username` can manage. It connects to the Satellite Server and authenticates as `rhn-username` with the password of `rhn-password`. The `listUserSystems` method in the `system` namespace is called to generate a list of systems that `rhn-username` can administrate. The `foreach` loop prints the `name` field of each of the values in the list of systems.

This script should work as long as `rhn-username` is a valid Red Hat Network user. Permissions aren't an issue since the user authenticating is accessing his own system group information. The `listUserSystems` method can take a second argument which would be a user's login name, but since it is omitted in this example it lists the systems managed by the authenticated RHN user. If this script were used to list the systems owned by other users, then it would have to authenticate as an Organization Administrator.



```
#!/usr/bin/python
import xmlrpclib

URL = "https://satellite.example.com/rpc/api"
user = "rhn-username"
pswd = "rhn-password"

client = xmlrpclib.Server(URL, verbose=0)
session = client.auth.login(user, pswd)

list = client.user.list_assigned_system_groups(session, user)
for group in list:
    print group.get('name')

client.auth.logout(session)
```

The Python script above lists the system groups that the user `rhn-username` can manage. It connects to the Satellite Server and authenticates as `rhn-username` with the password of `rhn-password`. The `listAssignedSystemGroups` method in the `user` namespace is called to generate a list of system groups that `rhn-username` (passed as the variable `user`) can administrate. The `for` loop prints the `name` field of each of the values in the list of system groups.

This script should work as long as `rhn-username` is a valid Red Hat Network user. Permissions aren't an issue since the user authenticating is accessing his own system group information. If this script were used to inspect the system groups of other users, it would have to authenticate as an Organization Administrator.



End of Lecture 8

- Questions and Answers
- Summary
 - Uses for Red Hat Network API
 - Basic API program structure
 - Sample programs

Lab 8.1: Getting started with the Red Hat Network API

Scenario: This exercise will introduce you to the Red Hat Network API. Modify two versions of a script written in Perl and Python so they successfully query a RHN Satellite Server.

Instructions:

1. There is a Perl script, `list-users.pl`, and a Python script, `list-users.py`, which list all the users within an Red Hat Network organization. The scripts can be found at the following URL: <http://station30.example.com/pub/rhn-api>.

Log in as `student` on your workstation, copy the scripts, and modify them so they will successfully query the Satellite Server and list the users in the “SummitXX” organization. The `XX` corresponds to the 2-digit equivalent of your station number. The Organization Admin user for that organization authenticates as `summitXX` with a password of `redhat`.

Lab 8.2: Using the Red Hat API to produce reports

Scenario: Modify one of the provided scripts to produce a useful report by using the Red Hat Network API to get more detailed information about the users from the Satellite Server.

Instructions:

1. Write a script, `list-user-roles.pl` or `list-user-roles.py`, that lists all of the users within your Summit organization. Print the following information about each user:
 - Login name
 - List of RHN administrative roles

Copy one of your working scripts as a starting point for your new script.

Lab 8.3: Using the RHN API to perform Satellite administration

Scenario: Write a couple Red Hat Network API scripts that perform RHN Satellite administration functions.

Instructions:

1. Write two scripts that use the Red Hat Network API to administrate users. The `user-disable.pl`, or `user-disable.py`, script should deactivate a RHN user account. Its positive counterpart, `user-enable.pl` or `user-enable.py`, should reactivate an existing user account. Use a program variable for the RHN login to be enabled/disabled.

These programs don't have to be fancy, they just have to be functional. There is no need to process command-line arguments or do extensive error checking.

2. Use one of your scripts to disable one of your organization's RHN accounts. Try to log into the RHN Satellite web interface as that user with the password `redhat` and verify the account is disabled. Execute the other script to reactivate that account and verify they can log into the Satellite Server.

Lab 8.1 Solutions

1. There is a Perl script, `list-users.pl`, and a Python script, `list-users.py`, which list all the users within an Red Hat Network organization. The scripts can be found at the following URL: <http://station30.example.com/pub/rhn-api>.

Log in as `student` on your workstation, copy the scripts, and modify them so they will successfully query the Satellite Server and list the users in the “SummitXX” organization. The `XX` corresponds to the 2-digit equivalent of your station number. The Organization Admin user for that organization authenticates as `summitXX` with a password of `redhat`.

Edit `list-users.pl` and `list-users.py` and change the host and user authentication information to work for the Satellite Server. For example the following lines need to be modified in the Perl script:

```
my $SATELLITE_URL = 'https://station30.example.com/rpc/api';
my $SATELLITE_LOGIN = 'summitXX';
my $SATELLITE_PASSWORD = 'redhat';
```

Execute both scripts and troubleshoot any problems they may have. A few possible issues to investigate:

- Are the scripts executable?
- Does `SATELLITE_URL` point to the Satellite Server? Were only the `SATELLITE_*` variable definitions modified?
- Are `SATELLITE_LOGIN` and `SATELLITE_PASSWORD` defined to use Organization Administrator credentials for your organization?

Organization Administrator privileges are required to access user account information about an organization. API scripts run with privileges determined by the account they use to authenticate into the Satellite Server with.

Lab 8.2 Solutions

1. Write a script, `list-user-roles.pl` or `list-user-roles.py`, that lists all of the users within your Summit organization. Print the following information about each user:
 - Login name
 - List of RHN administrative roles

The following commands copy the working Python script:

```
[student@stationX api]$ cp list-users.py list-user-roles.py
```

The basic structure of the new program is the same as the sample scripts: connect to RHN, authenticate, list the users, then log out. Some additional work needs to be done when listing each user. The `User` namespace provides a method called `listRoles` that will get the information we need. This method takes a session key and a login name and returns an array of strings which are the RHN administrative roles assigned to the user.

Additional Perl code needed:

```
my $ulist = $client->call('user.list_users', $key);
foreach my $user (@$ulist) {
    print $user->{'login'} . "\n";
+   # Identify and print each user's roles:
+   my $rlist = $client->call('user.list_roles', $key, $user->{'login'});
+   foreach my $role (@$rlist) {
+       print '    ' . $role . "\n";
+   }
}
```

Additional Python code needed:

```
for user in uelist:
    login=user.get('login')
    print login
+   # Identify and print each user's roles:
+   rlist = client.user.list_roles(key, login)
+   for role in rlist:
+       print '    ' + role
```

Lab 8.3 Solutions

1. Write two scripts that use the Red Hat Network API to administrate users. The `user-disable.pl`, or `user-disable.py`, script should deactivate a RHN user account. Its positive counterpart, `user-enable.pl` or `user-enable.py`, should reactivate an existing user account. Use a program variable for the RHN login to be enabled/disabled.

The basic structure of the new program is the same as the other RHN API scripts: connect to RHN, authenticate, enable/disable the specified user account, then log out. The `User` namespace provides a couple useful methods called `enable` and `disable` that will do what we need. These methods take a session key and the RHN login name of the account to manipulate.

Below is working Perl code that implements the disable script:

```
#!/usr/bin/perl -w
use strict;
use Frontier::Client;

# Define RHN Satellite host and authentication values:
my $SATELLITE_URL = 'https://station30.example.com/rpc/api';
my $SATELLITE_LOGIN = 'summitXX';
my $SATELLITE_PASSWORD = 'redhat';

# Login name of user to disable:
my $login_name = 'login_to_disable';

# Connect to the RHN Satellite Server:
my $client = new Frontier::Client(url => $SATELLITE_URL);

# Authenticate as a valid user to get a session key:
my $key = $client->call('auth.login', $SATELLITE_LOGIN,
    $SATELLITE_PASSWORD);

# Disable the user in our organization:
$client->call('user.disable', $key, $login_name);

# Logout from RHN session:
$client->call('auth.logout', $key);
```

The Python solution is similar to the above code with a few syntactical differences. Also the `enable` function is a trivial change to the above program.

2. Use one of your scripts to disable one of your organization's RHN accounts. Try to log into the RHN Satellite web interface as that user with the password `redhat` and verify the account is disabled. Execute the other script to reactivate that account and verify they can log into the Satellite Server.