



<http://systems.pixar.com>
@pixarsystems

Objects to Objects to Tools

Enhancing Casper with the API through Ruby



Chris Lasell
Apple Peeler
Pixar Animation Studios





A Long time ago

I did a JNUC session about some command-line tools we wrote that extend the Casper Suite

Primarily about

- d3, our package deployment & patch management system, which includes
- puppytime to entertain users during a logout-install,

and

- d3admin which provides manual or scriptable deployment of packages, even directly from XCode

JNUC 2012

“Is it open sourced?”

-Rich Trouton

JNUC 2014

“Yes!”

-Pixar

P I X A R
ANIMATION STUDIOS

Class List

Classes Methods Files

Search

Top Level Namespace

Array < Object

FileText

Hash < Object

IPAddr < Object

* JSS

APIConnection < Object

APIObject < Object

AdvancedComputerSearch < Adv

AdvancedDesktopDeviceSearch < Adv

AdvancedSearch < APIObject

AdvancedUserSearch < Advanced

AlreadyExistsError < RuntimeError

Building < APIObject

Category < APIObject

Client < Object

Composer

Computer < APIObject

ComputerExternalAttribute < R

ComputerGroup < Group

Configuration < Object

Createable

> Createable

DISConnection < Object

Department < APIObject

DistributionPoint < APIObject

Extendable

ExternalAttribute < APIObject

FileServiceError < RuntimeError

FileUpload

Group < APIObject

InvalidConnectionError < Runtim

InvalidValueError < RuntimeError

LDAPServer < APIObject

HOME > FILE README

The JSS Ruby Gem - access to the CasperSuite API

DESCRIPTION

JSS is a Ruby gem providing access to the REST API of JAMF Software's Casper Suite. It abstracts API resources as Ruby objects, and provides methods for interacting with these resources. It also provides some features that aren't a part of the API itself, but come with other Casper-related tools, such as uploading files and doing JSS Package data to the master distribution point, and the installation of JSS Package objects on client machines. (See BEYOND THE API)

The gem is not a complete implementation of the Casper API. Only some API objects are modeled, some only nominally. Of those, some are read only, some partially writable, some fully read-write. See OBJECTS IMPLEMENTED for a list. Basically the implemented the things we need in our environment, and as our needs grow, I'll add more. Hopefully others will find it useful, and add more to it as well.

SYNOPSIS

```
require 'jss'

JSS::API.connect :user => :jss_user, :pw => :jss_user_pw, :server => :jss_server_hostname

JSS::Package.all # returns an array of data about all JSS::Package objects in the JSS
JSS::Package.all_names # returns an array of names of all JSS::Package objects in the JSS

# Get a specific computer group
mg = JSS::ComputerGroup.new :name => "Mac of Interest"

# Add a computer to the group
mg.add_member "prickteppits"

# Save my changes
mg.update

# Create a new network segment to store in the JSS
ns = JSS::NetworkSegment.new :id => :new, :name => "Private Class C", :starting_address => "192.168.8.8", :ending_address => "192.168.8.255"

# Associate this network segment with a specific building, which must exist in the JSS, and be listed in JSS::Building.all_names
ns.building = "Main Office"

# Associate this network segment with a specific software update server, which must exist in the JSS, and be listed in JSS::SoftwareUpdateServer.all_names
ns.swu_server = "Main SWU Server"

# Store the new network segment in the JSS
ns.create
```

USAGE

Connecting to the API

Before you can work with JSS Objects via the API, you have to connect to it.

Table of Contents (only)

1. DESCRIPTION

2. SYNOPSIS

3. USAGE

1. Connecting to the API

2. Working with JSS Objects (is it a REST Resource?)

1. Listing Objects

2. Removing Objects

3. Creating Objects

4. Updating Objects

5. Deleting Objects

4. OBJECTS IMPLEMENTED

1. Createable and Updateable

2. Updateable but not Createable

3. Read-Only

4. Deletable

5. CONFIGURATION

6. BEYOND THE API

7. REQUIREMENTS

8. INSTALL

9. RUNNING TESTS

10. LICENSE

The biggest question after that talk •

- Is it open sourced?

Two years later,

- I was happy to say that Yes it has been, partially, open sourced

The JSS Ruby module

Overview

Working with objects

d3

Overview

Packages

Client

Admin

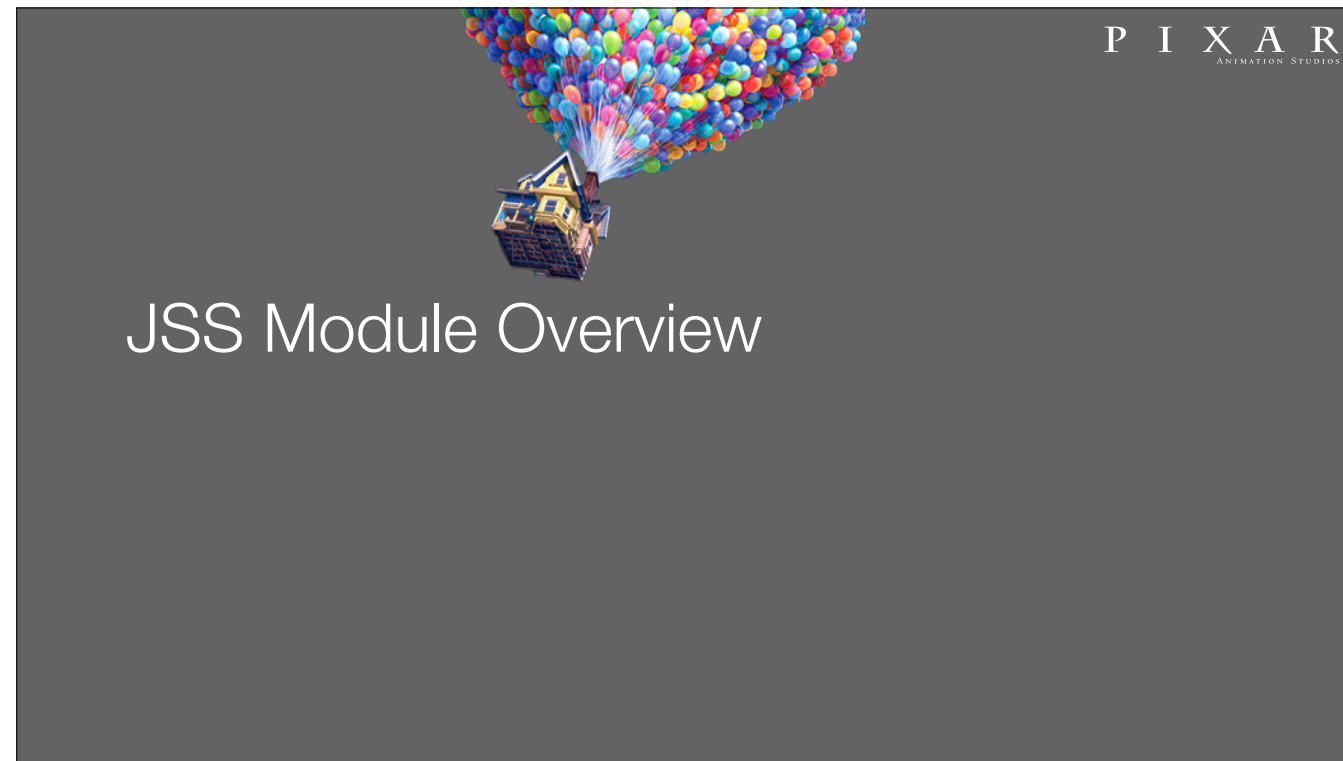
A little over a year ago we released the jss-api gem

- provides a Ruby module that makes interacting with the API much simpler

- That module is the foundation of d3

Within the next few months we'll be releasing d3 itself as a part of that project.

- Today I'm going to chat a bit about the ruby module,
- and give you an overview of d3.



JSS Module Overview

Lets take a look at using the JSS API with ruby

P I X A R
ANIMATION STUDIOS

It's all about the objects

JSS REST API Resource Documentation

/accounts	Show/Hide	List Operations	Expand Operations
/activationcode	Show/Hide	List Operations	Expand Operations
/advancedcomputersearches	Show/Hide	List Operations	Expand Operations
/advancedmobiledevicesearches	Show/Hide	List Operations	Expand Operations
/advancedusersearches	Show/Hide	List Operations	Expand Operations
/buildings	Show/Hide	List Operations	Expand Operations
/byopprofiles	Show/Hide	List Operations	Expand Operations
/categories	Show/Hide	List Operations	Expand Operations
/classes	Show/Hide	List Operations	Expand Operations
/computercheckin	Show/Hide	List Operations	Expand Operations
/computercommands	Show/Hide	List Operations	Expand Operations
/computerextensionattributes	Show/Hide	List Operations	Expand Operations
/computergroups	Show/Hide	List Operations	Expand Operations
/promoterinvestorcollection	Show/Hide	List Operations	Expand Operations

Class List

Classes | Methods | Files

Search

- Top Level Namespace
- Array < Object
- FileTest
- Hash < Object
- IPAddr < Object
- ▼ JSS
 - APIConnection < Object
 - APIObject < Object
 - AdvancedComputerSearch < AdvancedComputerSearch
 - AdvancedMobileDeviceSearch < AdvancedMobileDeviceSearch
 - AdvancedSearch < APIObject
 - AdvancedUserSearch < AdvancedUserSearch
 - AlreadyExistsError < RuntimeError
 - Building < APIObject
 - Category < APIObject
 - Client < Object
 - Composer
 - Computer < APIObject
 - ComputerExtensionAttribute < ExtensionAttribute
 - ComputerGroup < Group
 - Configuration < Object
 - Creatable
 - Criteriable
 - DBConnection < Object
 - Department < APIObject
 - DistributionPoint < APIObject
 - Extendable
 - ExtensionAttribute < APIObject
 - FileServiceError < RuntimeError
 - FileUpload
 - Group < APIObject

The Casper REST API provides a way to interact with many kinds of objects stored in the JSS

- The JSS Ruby module abstracts some of those objects as Ruby objects.

The Ruby objects have many easy-to-use methods for manipulating the API objects

P I X A R
ANIMATION STUDIOS

Which Objects?

Create, Read, Update, Delete:

AdvancedComputerSearch
AdvancedMobileDeviceSearch
AdvancedUserSearch
Building
Category
ComputerExtensionAttribute
ComputerGroup
Department
MobileDeviceExtensionAttribute
MobileDeviceGroup

Read, Delete:

DistributionPoint
LDAPServer
NetBootServer
SoftwareUpdateServer

Read, Update, Delete:

Computer
name
barcodes
asset tag
ip address
location data
purchasing data
editable extension attributes

MobileDevice
asset tag
location data
purchasing data
editable extension attributes

Policy

scope
name
enabled
category
triggers
file actions
process actions

So far the Ruby module implements about half of the API objects, to varying degrees

- Create, Read, Update & Delete
- Read, Update & Delete
- Read & Delete

Why not all the objects, fully implemented? Mostly because of time
- I'm the only developer, the API is large, and Pixar doesn't need access to everything in there.

Non-API classes

APIConnection

An object representing the REST connection

Configuration

Stored settings for the connection

Server

Info about the JSS at the other end of the connection

Client

JAMF-related items on the local machine.

Composer (Module)

For creating simple .pkg and .dmg installers from a pre-made 'root' folder.

There are a few things in the Module that don't reflect API objects directly, but have other uses.

- APIConnection object
 - contains server addresses, authentication credentials, timeouts
 - methods for connecting, disconnecting, examining.
- Configuration object
 - for working with system-wide and user-specific config files
- Server object
 - represents the JSS you're connected to through the APIConnection
- Client
 - represents the Casper-managed machine on which the code is running
- Composer
 - for building simple .pkg and .dmg packages.

Synchronicity

Extending the API by making objects work together...

Package + DistributionPoint + NetworkSegment + Client = Package.install

Script + Client = Script.run

ExtensionAttribute + AdvancedSearch + Computer = Better Ext.Attr. Reporting

Objects that refer to other objects can check for existence = data consistency



The API provides access to data about JSS objects, but once we get them into a unified programming environment, those objects can work together to make something even more powerful.

- Install packages from the correct dist.point. for the current net seg. or building
- Run scripts
- Generate reports on ExtAttr values by creating on-the-fly AdvancedSearchs as needed
- ensure data consistency before getting API errors

Working with objects



So lets take a look at how to use API objects this way.

The examples I'll show you are captured from an "IRB" session,

which is like a shell where you can type ruby code

just like you can type bash code in a bash shell.

You'll see how easy it is to manipulate API objects without ever seeing any XML.

Installing

In a terminal: `gem install jss-api`

Downloads from rubygems.org

```
root@kimchi ~: gem install jss-api
```

Automatically installs other gem dependencies

`rest-client`

`plist`

`net-ldap`

Like all gems, JSS module is easy to install..

- `gem install jss`
- installs dependencies as needed

Connecting to the API

Require the Gem

```
1:> require 'jss-api'
=> true
```

JSS::API.connect

Specify all options, or...

```
2:> JSS::API.connect(
:server => 'myjss.myorg.org',
:user => 'jssadmin',
:pw => 'jssadmpass',
:verify_cert => false
)
=> true
```

Options stored in JSS::CONFIG
will be used automatically

```
3:> JSS::API.connect :pw => 'jssadmpass'
=> true
```

Passwords can be provided, piped
through stdin, or prompted.

```
4:> JSS::API.connect :pw => :prompt
Enter the password for JSS user 'jssadmin':
=> true
```

Once installed, fire up IRB or open your favorite editor

- require the gem, which loads the module
- API.connect, specify all options
- or use Config and just the passwd
- or :prompt or :stdin

Listing Objects

JSS::APIObject.all

JSS::APIObject.all_identifier

JSS::APIObject.map_all_ids_to

```
4:> JSS::MobileDevice.all
=> [{:name=>"greyhearts",
      :id=>330,
      :serial_number=>"Cgggggggg9V"},
     {:name=>"tipple",
      :id=>343, ...}]
```

```
5:> JSS::Computer.all_names
=> ["rowdy",
    "takeoff",
    "lights",
    "windward",
    "trifecta" ...]
```

```
6:> JSS::Policy.map_all_ids_to :name
=> {465=>"conf.metro-pixarification",
    693=>"Firefox ESR",
    627=>"Pilot-pixelmator",
    523=>"login.hive", ...}
```

- **APIObject.all**

- An Array of Hashes for each object
- The JSON response of the API list resource

- **APIObject.all_identifiers** - all_names, all_ids, all_macaddresses, etc

- **APIObject.map_all_ids_to** :name, :serialnumber, etc...

Retrieving Objects

`JSS::APIObject.new`

`:id => Integer`

-or-

`:name => String`

Attributes as methods

Simple

Complex

```
7:> a_comp = JSS::Computer.new :name => "kimchi"
=> #<JSS::Computer:0x10763bc50>
```

```
8:> a_comp.serial_number
=> "W80336UK2A5GV"
```

```
9:> a_comp.last_contact_time
=> Thu Aug 28 12:23:03 -0700 2014
```

```
10:> a_comp.hardware[:processor_type]
=> "Intel Core i5"
```

- Objects are retrieved by creating a Ruby instance with the `:id` or `:name` of a JSS object.

Sometimes by other identifiers, eg Computers by `:id`, `:name`, `:serialnumber`, `:macaddress`, `:udid`

- Attributes from the API are available via method calls, usually with the same name.

All objects have `#id` and `#name` methods. Other vary depending on the object.

Updating Objects

Simple Attributes

```
11:> a_comp.barcode_1 = "123abc890"
=> "123abc890"

12:> a_comp.po_number = "po-98765"
=> "po-98765"

13:> a_comp.po_date = 'Sep 13, 2014'
=> "Sep 13, 2014"
```

- Change the value of a simple attribute with =
- Some attributes are more complex, like scope, which is a Scope object. - errors can be raised
- Save is an alias of update

Updating Objects

```
14:> a_policy = JSS::Policy.new :name => "my-Pol"
=> #<JSS::Policy:0x101d3d978>

15:> a_policy.scope.add_limitation :network_segments, 'backups'

>>>>> JSS::NoSuchItemError: No existing network_segments with name 'backups'

16.^ JSS::NetworkSegment.all_names
=> ["backup-lan", "av-theater", "main-3-l1"...]

17:> a_policy.scope.add_limitation :network_segments, 'backup-lan'
=> true
```

Complex Attributes

- Change the value of a simple attribute with =
- Some attributes are more complex, like scope, which is a Scope object. - errors can be raised
- Save is an alias of update

Updating Objects

Send Changes to the JSS

```
18:> a_comp.update  
=> 3313
```

```
19:> a_policy.save  
=> 735
```

- Change the value of a simple attribute with =
- Some attributes are more complex, like scope, which is a Scope object. - errors can be raised
- Save is an alias of update

Creating Objects

`JSS::APIObject.new :id => :new, :name => 'a_name'`

```
20:> new_pkg = JSS::Package.new :id => :new, :name => "filemaker13.pkg"
=> #<JSS::Package:0x101d3d978>
```

```
21:> new_pkg.in_jss?
=> false
```

Add attribute data

```
22:> new_pkg.category = 'Useful Software'
=> true
```

```
23:> new_pkg.os_limitations = '>=10.6.x'
=> true
```

Create it in the JSS

```
24:> new_pkg.create # Also new_pkg.save
=> 573
```

- `:id => :new`, need at least a `:name`.

Some classes require more than just `:name` when creating
e.g. Groups require `:type` as either `:smart` or `:static`

- This doesn't create anything in the JSS, just gives you a Ruby object to work with..
- Save is also an alias for create

Deleting Objects

All API objects can be deleted

Call #delete on an object

Happens immediately!

Be Careful!

One-liner

```
25:> my_dead_cat = JSS::Category.new :name => 'my category'  
=> #< Category:0x001d3d578>
```

```
26:> my_dead_cat.delete  
=> true
```

```
27:> JSS::Category.new(:name => 'my category').delete  
=> true
```

- All API objects can be deleted
- retrieve any object, and you can call its delete method
- Ruby lets you retrieve and delete in one step.

Documentation

Heavily documented source

YARD-generated html

<http://www.rubydoc.info/gems/jss-api>

```
## Note: This code must be run as root to uninstall packages
##
## Causes the pkg to be uninstalled via the jamf command.
##
## @param args[Hash] the arguments for installation
##
## Option args :target[String,Pathname] The drive from which to uninstall the package, defaults to '/'
##
## Option args :verbose[Boolean] be verbose to stdout, defaults to false
##
## Option args :feu[Boolean] fill existing users, defaults to false
##
## Option args :fut[Boolean] fill user template, defaults to false
##
## @return [Process::Status] the result of the 'jamf uninstall' command
##
def uninstall(args = {})
  raise JSS::UnsupportedError, \
    "This package cannot be uninstalled. Please use CasperAdmin to index it and allow uninstalls" unless removable?
  raise JSS::UnsupportedError, "You must have root privileges to uninstall packages" unless JSS.superuser?
  args
end

- (Process::Status) uninstall(args = {})

Note: This code must be run as root to uninstall packages

Causes the pkg to be uninstalled via the jamf command.

Parameters:
  * args (Hash) (defaults to: {}) — the arguments for installation

Options Hash (args):
  * :target (String,Pathname) — The drive from which to uninstall the package, defaults to '/'
  * :verbose (Boolean) — be verbose to stdout, defaults to false
  * :feu (Boolean) — fill existing users, defaults to false
  * :fut (Boolean) — fill user template, defaults to false

Returns:
  * (Process::Status) — the result of the 'jamf uninstall' command

Raises:
  * (JSS::UnsupportedError)

[View source]
```

The things I've shown so far are just the tip of the iceberg.

To really use the Module you'll have to start looking at the documentation.

And if you'd like to modify (and hopefully contribute to) the code, the docs are ***VERY*** important

- The source code is heavily documented.
- The reason for the extensively formatted comments ...auto-generated RDOC and YARD HTML docs



So there are the basics of using Ruby to access the JSS API

There are a couple sample programs included with the gem to show how to use it for practical purposes.

The main reason it was created, however, is for d3.

What's d3?

A Package & Patch management tool for Casper

Sort-of "munki for Casper", entirely command-line

Extends the capabilities of Casper's package management

A client 'd3', and admin tool 'd3admin'

Utilizes Casper scripts and policies for customization

Stands for 'depot 3'

Whats D3?

- A Patch Management tool for casper packages.

(read the slide)

d3 Packages

Casper Packages that...

- Know they are different versions of the same thing
- Know their status - pilot, live, skipped, deprecated
- Know their .pkg identifier(s)
- Auto-update on clients when new version is released
- Conditionally (un)install based on pre- script status
- Quick and easy manipulation from a shell
- Can Auto uninstall if unused after some days

Packages in d3 are just Casper packages

- but they're smarter.

(read the slide)

d3 client

In a terminal: *d3 options command arguments....*

Install/Uninstall packages

```
root@kimchi ~: d3 install filemaker
```

Pilot packages

```
root@kimchi ~: d3 pilot filemaker-13-3
```

Get lists of many kinds

```
root@kimchi ~: d3 list-installed
```

Sync

```
root@kimchi ~: d3 list-available
```

Puppytime!

```
root@kimchi ~: d3 list-files
```

```
root@kimchi ~: d3 sync
```

- The d3 client is a shell command called 'd3' which takes many subcommands.

It can...

(read slide)

d3 sync

Automates many things

- Updates installed packages
- Installs new packages in scope
- Updates receipts
- Enliven's pilot installs
- Expires packages

Syncing is the heart of d3's automation.

- When the sync command is given,

(read slide)

Syncing should be done multiple times a day - we use a LaunchDaemon to do it every 6 hours

But it can be done manually whenever a machine needs to get caught up

PuppyTime!

Logout-installs of packages requiring reboot.

Packages are added to queue when 'installed'

User is notified to log out ASAP

Logout policy runs puppytime

Packages in the queue are installed

Displays slideshow of cute puppies

Optionally run another policy before reboot.



When d3 installs a package that requires a reboot,

- it doesn't install it, but instead adds it to a queue of packages awaiting logout.

(read slide)

d3admin

In a terminal: `d3admin action target options...`

Add packages to d3 & Casper, building them if needed

Configure package settings

Make packages "live" after piloting them (i.e. release them)

Walk-thru, or provide all options on the command line

Scriptable, integrates with XCode

Client reports - installs, pilots, deprecated installs

Archive packages and metadata



d3admin is the tool for adding & editing packages in d3.

You can think of it as a combination of Casper Admin and Composer, but its all CommandLine.

- Use it to

(read slide)

Add packages to d3 and Casper, building them if needed

Configure package settings

Make packages "live" after piloting them (i.e. release them)

Walk-thru prompting, or provide all options on the command line

Scriptable, integrates with XCode

Client reports - installs, pilots, deprecated installs

Archive packages and metadata

Open Source

Home pages

<http://pixaranimationstudios.github.io/jss-api-gem>

<http://pixaranimationstudios.github.io/depot3> (Coming Soon)

Slides

<https://git.io/vzZ9N>

Email

jss-api-gem@pixar.com

d3@pixar.com



Slack MacAdmins

#jss-api

@glenfarclas17

There's a brief overview of the JSS module and d3.

The Module is already available via github and rubygems.org

A very alpha version of d3 will be up there before summer.

- I still have to write initial documentation and an installer.

- it requires a fair bit of setup in your JSS - some day an installer will walk you through that.

Thnk you!

Q & A

Q & A

Go back to prev. slide.