



JAMF NATION

user conference

OCTOBER 21-23 2014 MINNEAPOLIS, MN

Objects to Objects

The Casper Suite API through Ruby





Chris Lasell
Apple Peeler
Pixar Animation Studios



John Sutcliffe
VP, Software Engineering
JAMF Software



Objects to Objects



JNUC 2012



d3



d3admin



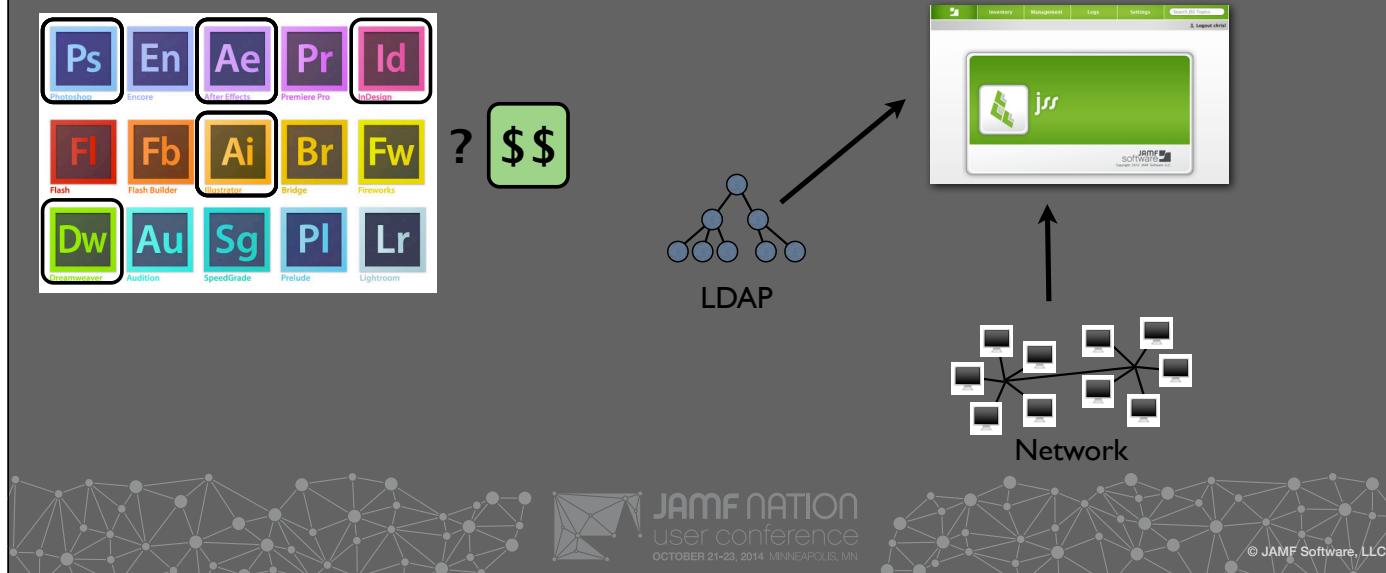
puppytime



At JNUC 2012

I did a 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



I also mentioned a few other things we do with the Casper API like

- purging unused applications from user's machines
- LDAP synchronization
- automatic network segment updating

JNUC 2012



MODULE PixJSS
[/Library/Ruby/Site/1.8/pixar/jss/computer.rb](#)
[show all...](#)
Parent: [jss.rb](#)

MODULE PixD3
[/Library/Ruby/Site/1.8/pixar/d3/installation_package.rb](#)
[show all...](#)
Parent: [d3.rb](#)

Classes and Modules

[PixJSS::AlreadyExistsError](#)
[PixJSS::DBConnection](#)
[PixJSS::FileServiceError](#)
[PixJSS::InvalidConnectionError](#)
[PixJSS::InvalidDataError](#)
[PixJSS::JSSComputer](#)
[PixJSS::JSSComputerGroup](#)
[PixJSS::JSExtAttrib](#)
[PixJSS::JSNetSegment](#)
[PixJSS::JSPkq](#)
[PixJSS::JSScript](#)
[PixJSS::MissingDataError](#)
[PixJSS::NoSuchItemError](#)
[PixJSS::RESTConnection](#)

D3 module – general d3-related values and methods

Classes and Modules

[PixD3::D3Pkq](#)
[PixD3::D3InstalledPkq](#)
[PixD3::D3Postflight](#)
[PixD3::D3Preflight](#)
[PixD3::D3ServerPkq](#)
[PixD3::InstallError](#)
[PixD3::InvalidDataError](#)
[PixD3::MissingDataError](#)
[PixD3::NoSuchItemError](#)
[PixD3::PendingPuppy](#)
[PixD3::PostflightError](#)
[PixD3::PuppyQ](#)
[PixD3::UninstallError](#)



We do all that with a homegrown library of Ruby code.♦

That library has been growing & by Casper 9, it needed a full re-write.

-For the past year plus, I've been working on that rewrite...

JNUC 2012

“Is it open sourced?”

-Rich Trouton

JNUC 2014

“Yes!”

-Pixar



The biggest question after my 2012 talk •

Here we are in 2014 and, in response to Rich Trouton,

- I'm happy to say that the JSS Ruby Gem has been open sourced.

Where to get it

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

The screenshot shows the GitHub page for the JSS Ruby Module. At the top, there's a navigation bar with links for 'About' and 'Contact'. Below the navigation is the title 'JSS Ruby Module'. A brief description follows: 'The JSS module is a Ruby framework for interacting with the REST API of the JAMF Software Server (JSS), the core of the Casper Suite, an enterprise-level management tool for Apple devices from JAMF Software, LLC. The module abstracts API objects as Ruby classes, which interact to allow simpler automation of Casper-related tasks.' Below this, there are three cards: 'Installation' (with a large red 'R' icon), 'Source' (with a GitHub logo icon), and 'Documentation' (with a red document icon). To the right of the main content area is the 'README' file, which contains code snippets and documentation for the JSS Ruby Gem. The README includes sections like 'DESCRIPTION', 'SYNOPSIS', and 'USAGE'. At the bottom of the page, there's a banner for 'JAMF NATION user conference OCTOBER 21-23, 2014 MINNEAPOLIS, MN'.

Here're the relevant details, they'll also be available at the end of the slideshow.

The JSS Ruby Gem

Overview

Working with objects

Mining the Gem

Example programs



The rest of my talk is going to go over the JSS Gem...

- Overview - what it does, what it doesn't (yet) do
- Working with objects -The basics of using the gem to interact with the API
- Mining the Gem - Coding the Gem itself, how it's built
- Example Programs - A couple of shell commands built with the Gem.

There's no way I can cover it all today, but
hopefully I'll give you enough to whet your appetites, and plant the seeds of possibility.



P I X A R
ANIMATION STUDIOS JAMF software

Overview



Lets take a look at what it does.

It's all about the objects



The API provides a way to interact with all kinds of objects stored in the JSS
The JSS Gem is a Ruby module which abstracts those objects as Ruby classes•

When you create an instance of a class, you get a Ruby object with many easy-to-use methods for manipulating it.

Which Objects?

Create, Read, Update, Delete:

- JSS::AdvancedComputerSearch
- JSS::AdvancedMobileDeviceSearch
- JSS::AdvancedUserSearch
- JSS::Building
- JSS::Category
- JSS::ComputerExtensionAttribute
- JSS::ComputerGroup
- JSS::Department
- JSS::MobileDeviceExtensionAttribute
- JSS:: MobileDeviceGroup

Read, Delete:

- JSS::DistributionPoint
- JSS::LDAPServer
- JSS::NetBootServer
- JSS::SoftwareUpdateServer

Read, Update, Delete:

JSS:: NetworkSegment	JSS:: Computer	JSS::Policy
JSS:: Package	name	scope
JSS:: Peripheral	barcodes	name
JSS:: PeripheralType	asset tag	enabled
JSS:: RemovableMacAddress	ip address	category
JSS:: Script	location data	triggers
JSS:: User	purchasing data	file actions
JSS:: UserExtensionAttribute	editable extension	process actions
JSS:: UserGroup	attributes	

JSS::MobileDevice
asset tag
location data
purchasing data
editable extension
attributes



As of Casper 9.4, the API provides 58 kinds of objects

So far the Gem implements 26, to varying degrees, as Ruby classes

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

Non-API classes

JSS::APIConnection

The single instance stored in constant
JSS::API

JSS::Configuration

Settings for the connection, stored in
JSS::CONFIG

JSS::Server

Info about the connected server, stored
in JSS::API::Server

JSS::Client

JAMF-related items on the local
machine.

JSS::Composer (Module)

Creation of simple .pkg and .dmg
installers from a pre-made ‘root’ folder.



There are a few classes and submodules within the Gem that don't reflect API objects directly, but are useful nonetheless.

- APIConnection
- Only a single instance
- #connect method takes all connection data, used at any time to change credentials,
- timeouts can be changed mid-stream
- Config
- System wide and user-specific config files, settable via methods
- Server
- Client
- Composer

Synchronicity

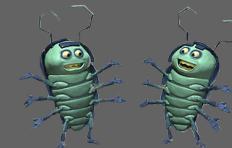
Extending the API by making objects work together...

JSS::Package, JSS::DistributionPoint, JSS::NetworkSegment and JSS::Client

JSS::Script and JSS::Client

JSS::ExtensionAttribute, JSS::AdvancedSearch, Inventory objects

Objects that refer to other objects can check for existence



The API provides access to 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 using the jamf binary from the correct dist.point. for the net seg.
- - Run scripts via jamf binary
- - report on ExtAttr values, enforce data consistency for ExtAttrs
- - Better data consistency before getting API errors

Working with objects



So lets take a look at how to use this thing.

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.

Installing

In a terminal: `gem install jss-api`

Downloads from rubygems.org

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

Automatically installs other gem dependencies

rest-client, which itself depends on mime-types

json

plist

net-ldap



Like all gems, JSS is easy to install..

- `gem install jss`
- installs dependencies as needed

Connecting to the API

Require the Gem

JSS::API.connect

Specify all options, or...

Options stored in JSS::CONFIG will be used automatically

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

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

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

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

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



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

Listing Objects

JSS::<APIObject>.all

JSS::MobileDevice.all

```
4:> pp JSS::MobileDevice.all
[{:name=>"greyhearts",
:id=>330,
:device_name=>"greyhearts",
:serial_number=>"Cgggggggg9V",
:supervised=>false,
:username=>"brian",
:model_display=>"iPhone 4S" ...]
```

JSS::<APIObject>.all_<identifier>

JSS::Computer.all_names

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

JSS::<APIObject>.map_all_ids_to

JSS::Policy.map_all_ids_to :name

```
6:> pp JSS::Policy.map_all_ids_to :name
{465=>"conf.metro-pixarification",
693=>"Firefox ESR",
627=>"Ph Widget",
656=>"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

```
JSS::Computer.new :id => 327
JSS::Computer.new :name => "kimchi"
```

Attributes as methods

Simple

Complex

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

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

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

```
11:> 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

More Complex Data

Send Changes to the JSS

```
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"

14:> a_policy = JSS::Policy.new :name => "sw-update-iphoto"
=> #<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

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>'
```

```
JSS::Package.new :id => :new, :name => "filemaker13.pkg"
```

Add attribute data

Create it in the JSS

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

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

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

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

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



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

Some classes require more than just :name 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')
=> #<JSS::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.

Mining the Gem



The past few slides have shown the basics of using the Gem.

Now we'll look a little deeper, not only at using it, but also programming the Gem itself.

Of course, I've written it with our own environment and JSS in mind, but since it's available in github, I'm hoping others will improve it by adding the functionality they need.

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.
### Spares 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]
def uninstall(args = {})

  raise JSS::UnsupportedError unless args[:target].is_a?(String)
  raise JSS::UnsupportedError unless args[:target].is_a?(Pathname)
  raise JSS::UnsupportedError unless args[:target].exists?

  ## are we doing a --force?
  do_feu = args[:feu]
  do_fut = args[:fut]

  ## use some binary
  jamf_opts = "-tuninst"
  ## run it via a cli
  uninstall_out = JSS.uninstall(jamf_opts, args[:target], do_feu, do_fut)

  return $?
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 methods I've shown so far are just the tip of the iceberg. To really use the Gem you'll have to start looking at the documentation.

And if you'd like to modify (and hopefully contribute to) the gem, the docs are *VERY* important
 The source code is heavily documented. • Here's the code for the uninstall method of the Package class
 The reason for the extensively formatted comments ...auto-generated • YARD HTML docs

Class Inheritance

APIObject...

```
.all, .all_names, .all_ids  
.id, .name, .delete
```

DistributionPoint < APIObject...

```
.master_distribution_point, .my_distribution_point  
.connection_type, .mount
```

MobileDeviceGroup < Group < APIObject...

Also AdvancedSearch and ExtensionAttribute

Class: JSS::APIObject

Inherits: Object	show all
Defined in: lib/jss/api_object/api_object.rb	more...

Class: JSS::DistributionPoint

Inherits: APIObject	show all
Defined in: lib/jss/api_object/distribution_point.rb	more...

Class: JSS::Group

Inherits: APIObject	show all
Includes: Createable, Criterionable, Updatable	
Defined in: lib/jss/api_object/group.rb	more...

Class: JSS::MobileDeviceGroup

Inherits: Group	show all
Defined in: lib/jss/api_object/group/mobile_device_group.rb	more...



Lets talk how the Gem itself is programmed.

- JSS::APIObject is the parent class of all objects from the API, defines things they all share, like ID's and names
- All Gem objects that come from the API are children of APIObject
- #delete is defined in APIObject, which is why all of them can be deleted.
- Some objects have two levels of parents

Mix-in Modules

Shared functionality & attributes in separate modules of code

Computer, MobileDevice, and Peripheral all include the Purchasable module

Group includes the Criteriable module.
So does AdvancedSearch

Computer uses Extendable, Locatable, Purchasable,
Updatable, Uploadable, and Matchable

Module: JSS::Purchasable

Included in: Computer, MobileDevice, Peripheral

Defined in: lib/jss/api_object/purchasable.rb

more...

Module: JSS::Criteriable

Included in: AdvancedSearch, Group

Defined in: lib/jss.rb

more...

Class: JSS::Computer

Inherits: APIObject

show all

Extended by: Matchable

Includes: Extendable, Locatable, Purchasable, Updatable, Uploadable

Defined in: lib/jss/api_object/computer.rb

more...



Ruby uses mixins

- Computers, MobileDevices and Peripherals all have purchasing data
- Smart groups have criteria, so do Advanced Searches
- Computers use six mixin modules

The documentation for module discusses the requirements for mixing in.

Modularity

JSS::Creatable

Objects can be created in the JSS

JSS::Updatable

Objects can be updated in the JSS

JSS::Locatable

Objects have location data

JSS::Purchasable

Objects have purchasing data

JSS::Extendable

Objects have extension attribute data

JSS::Scopable

Objects can be scoped

JSS::Criteriable

Objects contain criteria

JSS::Uploadable

Objects support API file-uploads

JSS::Matchable

Class supports simple match searches



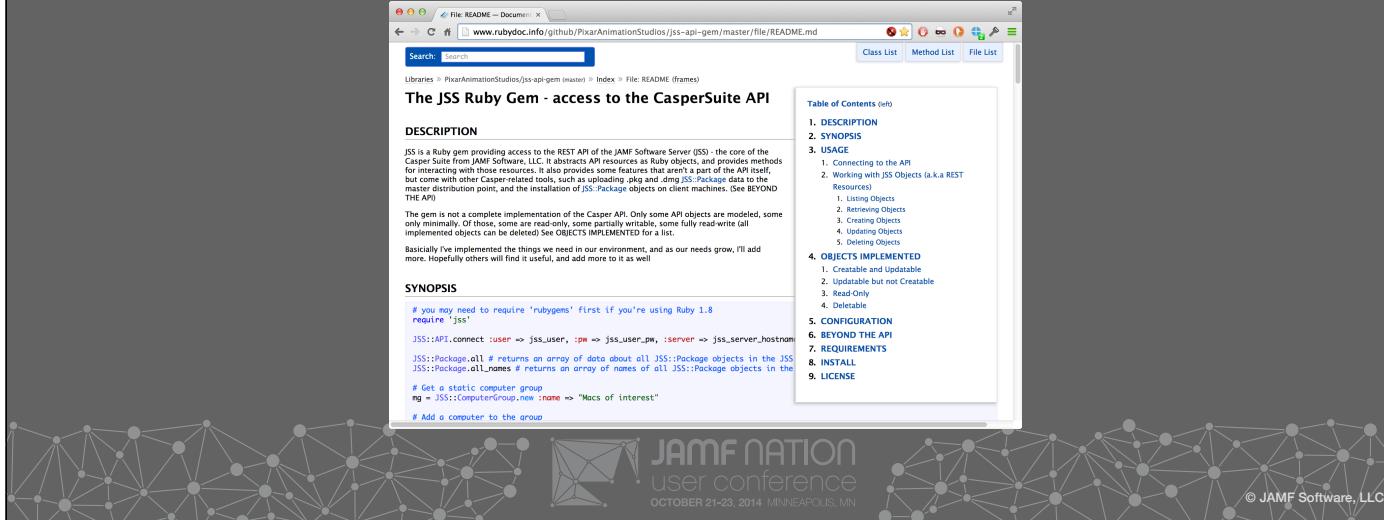
Here are the mix-in modules in the JSS at the moment.

The documentation for each gives details of how to use them.

- 9 clicks till Matchable

Documentation - again

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



Just like other languages and frameworks, when you create subclasses and use mix-in modules

there are certain requirements your code must meet. They are all detailed in the documentation.

Example Programs



Included with the gem are a couple small example programs....

I'm just showing screenshots at the moment, but might be able to show live action during the QnA

cgroupuer

Command line tool for working with
ComputerGroups

List, create, rename, delete,
edit membership

Create and edit only for static groups

```
A tool for working with computer groups in the JSS.

Usage: cgroupuer [-LsmcdlRc] [--help] [-n newname]
                  [-S server] [-U user] [-T timeout] [-V] [-d] [-f /file/path] [computer [computer ...]]

Options:
-L, --list-groups      - list all computer groups in the JSS
-S, --list-static       - list all static computer groups in the JSS
-m, --list-smart        - list all smart computer groups in the JSS
-c, --create-group      - create a new static computer group in the JSS
-n, --rename newname    - rename the specified computer group to newname
-d, --delete             - delete the specified computer group
-l, --list-members       - list all the computers in the group specified
-a, --add-members        - add the specified computer(s) to the specified group
-r, --remove-members     - remove the specified computer(s) from the specified group
-R, --remove-all          - remove all computers from the specified group
-f, --file /path/...       - read computer names/ids from the file at /path/...
-S, --server srvr        - specify the JSS API server name
-P, --port portnum        - specify the JSS API port
-U, --user username        - specify the JSS API user
-V, --no-verify-cert      - Allow self-signed, unverified SSL certificate
-T, --timeout secs        - specify the JSS API timeout
-C, --debug                - don't ask for confirmation before acting
--debug                   - show the Ruby backtrace when errors occur
-H, --help                 - show this help

Notes:
- If no API settings are provided, they will be read from /etc/jss_gem.conf
  and ~/.jss_gem.conf. See the JSS Gem docs for details.
- The password for the connection will be read from STDIN or prompted if needed
- Computers can be specified by name or JSS id number. If a name exists
  more than once in the JSS, the machine is skipped. Use IDs to avoid this.
- Only static groups can be modified. Use the JSS WebUI for editing smart groups
- If a file is used to specify computers, they are combined with any
  specified on the commandline.
- Files of computers must be whitespace-separated
  (spaces, tabs, & returns in any number or combination)
```



cgroupuer - commandline tool for working with computer groups

-Why? We had lots of times where lists of machines came from other sources, but were needed in casper.

subnet-update

Reads a tab-delimited file of subnet data
to create, modify, or delete

NetworkSegments as needed

Update the JSS Network Segments from a delimited file of subnet information.

```
Usage: subnet-update [-d delimiter] [--header] [-c col1,col2,col3] [-m manual-prefix] [--help] /path/to/file

Options:
-d, --delimiter      - The field delimiter in the file, defaults to tab.
-c, --columns [col1,col2,col3]
                    - The column order in file, must include 'name', 'starting',
                    and either 'ending' or 'cidr'.
-h, --header         - The first line of the file is a header line,
                    possibly defining the columns.
-m, --manual-prefix - Network Segment names in the JSS with this prefix are ignored.
                    Defaults to 'Manual.'
--cache /path/...
                    - Where read/save the input data for comparison between runs.
                    Defaults to ~/.lost_subnet_update
-S, --server srvr   - specify the JSS API server name
-P, --port portnum  - specify the JSS API port
-U, --user username  - specify the JSS API user
-V, --no-verify-cert - Allow self-signed, unverified SSL certificate
-T, --timeout secs  - specify the JSS API timeout
-H, --help            - show this help
--debug             - show the ruby backtrace when errors occur
```

This program parses the input file line by line (possibly accounting for a header line).
Each line defines the name and IP-range of a subnet/network segment.

- If a segment doesn't exist in the JSS, it is created.
- If a segment's range has changed, it is updated in the JSS.
- If a JSS segment doesn't exist in the file, it is deleted from the JSS unless its name starts with the --manual-prefix

Input File:

- The file must contain three columns, separated by the --delimiter, with these names, in any order:
 - 'name' (the network segment name)
 - 'starting' (the starting IP address of the network segment)
 - EITHER of:
 - 'ending' (the ending IP address of the network segment)
 - 'cidr' (the network range of the segment as a CIDR bitmask, e.g. '24')

Notes:

- The --columns option is a comma-separated list of the three column names above indicating the column-order in the file.
- If --columns are not provided, and --header is specified, the first line is assumed to contain the column names, separated by the delimiter.
- If --header is provided with --columns, the first line of the file is ignored.
- The raw data from the file is cached and compared to the input file at the next run. If the data is identical, no JSS connection is made.
- If no API settings are provided, they will be read from /etc/jss_gem.conf and ~/.jss_gem.conf. See the JSS Gem docs for details.
- The password for the connection will be read from STDIN or prompted if needed.



subnet-update: maintain NetworkSegments from an external source.

-Why? Our Network Admins maintain canonical subnet data in a git-repo, frequently updated.

d3 ?

Not yet, it's a separate project ... but soon!

Hopefully first half of 2015

A patch management system for Casper built on the JSS Gem

Gives Casper the ability to recognize different versions of the same software

Automatic updates on clients when new versions appear on the server

Puppies!

Provides automation, pilot installs, integration with XCode

All from the command-line, scriptable, automatable.

Will keep in touch with JAMF regarding Patch Management



d3? - •not yet.

separate, very large, project, dependent on this gem
what is it? •

Open Source



Where to get it

Home page

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

Email

jss-api-gem@pixar.com

JAMF Nation

ChrisL



Thank you!



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

jss-api-gem@pixar.com