Spectra D500 Lessons Learned

Hardware interfacing, Ruby on Windows, and other nasty stuff.

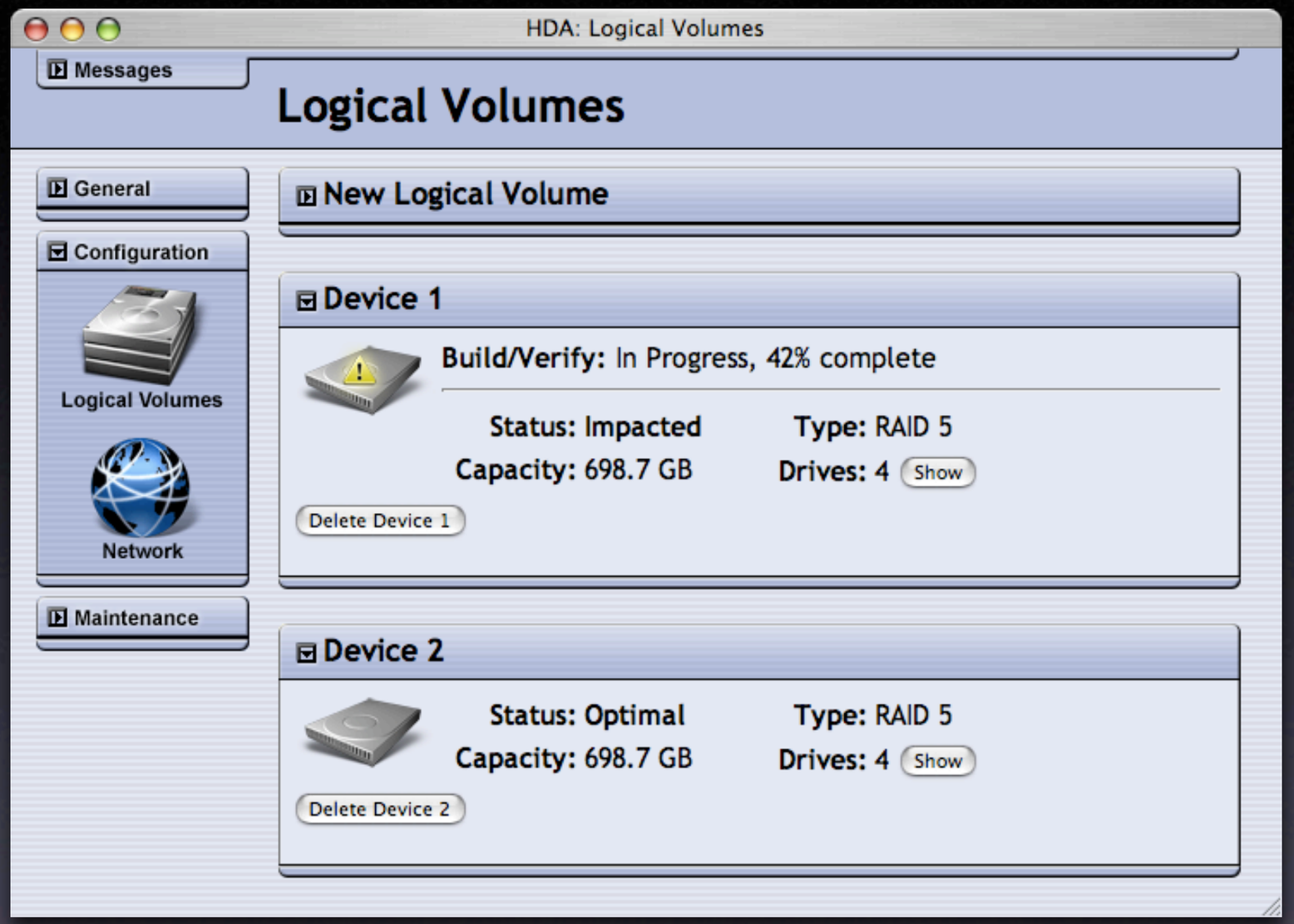
Josh Carter <2006@joshcarter.com>

Spectra Logic D500

Storage appliance with:

- Bunch of disk
- Choice of pre-installed applications

What *we're* interested in...



BlueScale: Rails-based management console

(yes, marketing made up the name)

BlueScale features

- Configures/monitors RAID volumes
- Configures/monitors storage applications
- Monitors drives
- Monitors chassis (fans, sensors, power, etc.)

Web-based vs. Embedded

Web	Embedded
One site, many customers	One site per customer
Many servers per site	One server per site
Software update easy	Software update real pain
Absolute control over servers	Customer maintains (and jacks with) the server

Dealing with Hardware

Is box A *really* the same as box B?

If it runs for 5 hours, will it run for 50?

What's the manufacturing tolerance of widget x?

What happens when it gets hot?
or humid?

What happens if it breaks in shipping?

Testing

Automated tests are HUGE

Unit test: app logic with simulated hardware

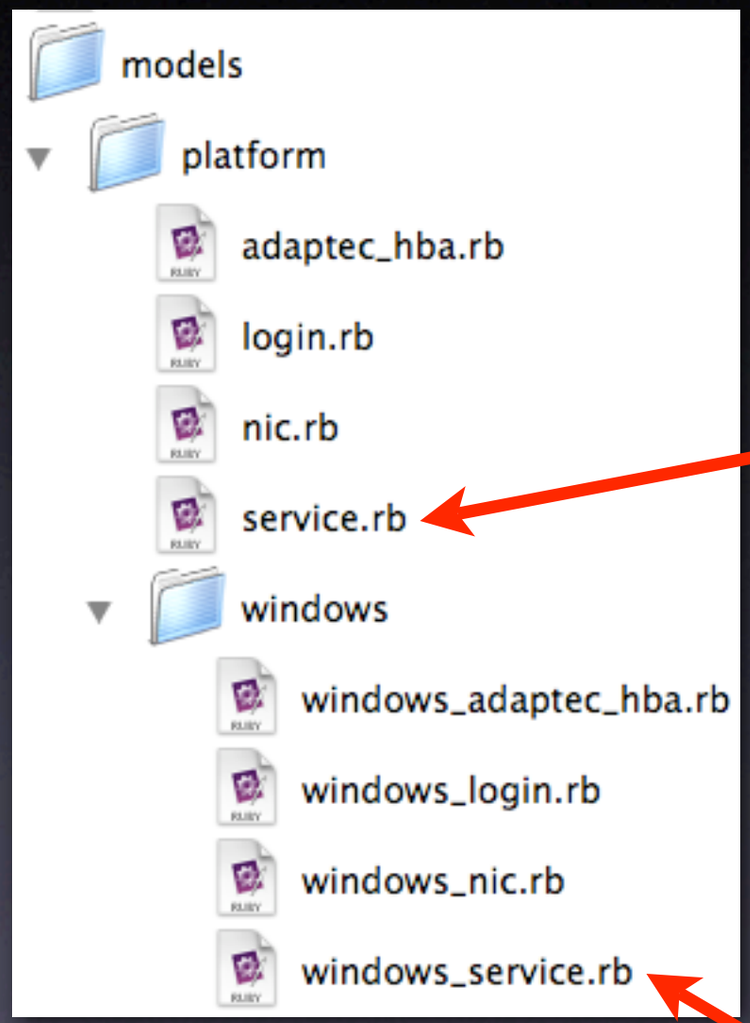
Integration test: whole system, real hardware

Simulation

Testing on real hardware is too slow *(and it won't fit on your laptop, either)*

Therefore: fake it

Platform-specific Models




```
class Service
  attr_accessor :name, :description, :state

  def self.find_all
    #
    # Create some fake services for offline use
    #
    entries
  end
end

if (running_on_windows?)
  require 'windows/windows_service'
end
```

!!

```
class Service
  def self.find_all
    # Get real services from WMI
  end
end
```

Interfacing with 3rd Party Tools

Usually devolves to parsing CLI output:

```
IO.popen("your\\utility\\here.exe") do |pipe|
  pipe.each_line do |line|
    re = /(\w+) nasty (\w+) regexp \w?\s?(\w+)/
    data = re.match(line.chomp)
    # ...
  end
end
```

Text Parsing: blocks for special cases

```
# Map Bar text keys => attributes
map = {
   "Status"           => :status=,
   "Widget Notifier" => :notifier=,
   "Channel/Device"  => lambda do |bar, text|
      channel, device = text.split(/,/)
      bar.channel = channel
      bar.device = device
   end
}

# Parser now checks for mapping type
case (map[key])
   when Proc:   map[key].call(item, val)
   when Symbol: item.send(map[key], val)
end
```

See: http://multipart-mixed.com/software/using_blocks_for_special_cases.html

Interfacing with 3rd Party Tools

Need separate process/ service running and some form of IPC.

Hmm... some form of IPC... where would we get one of those?

Some forms of IPC

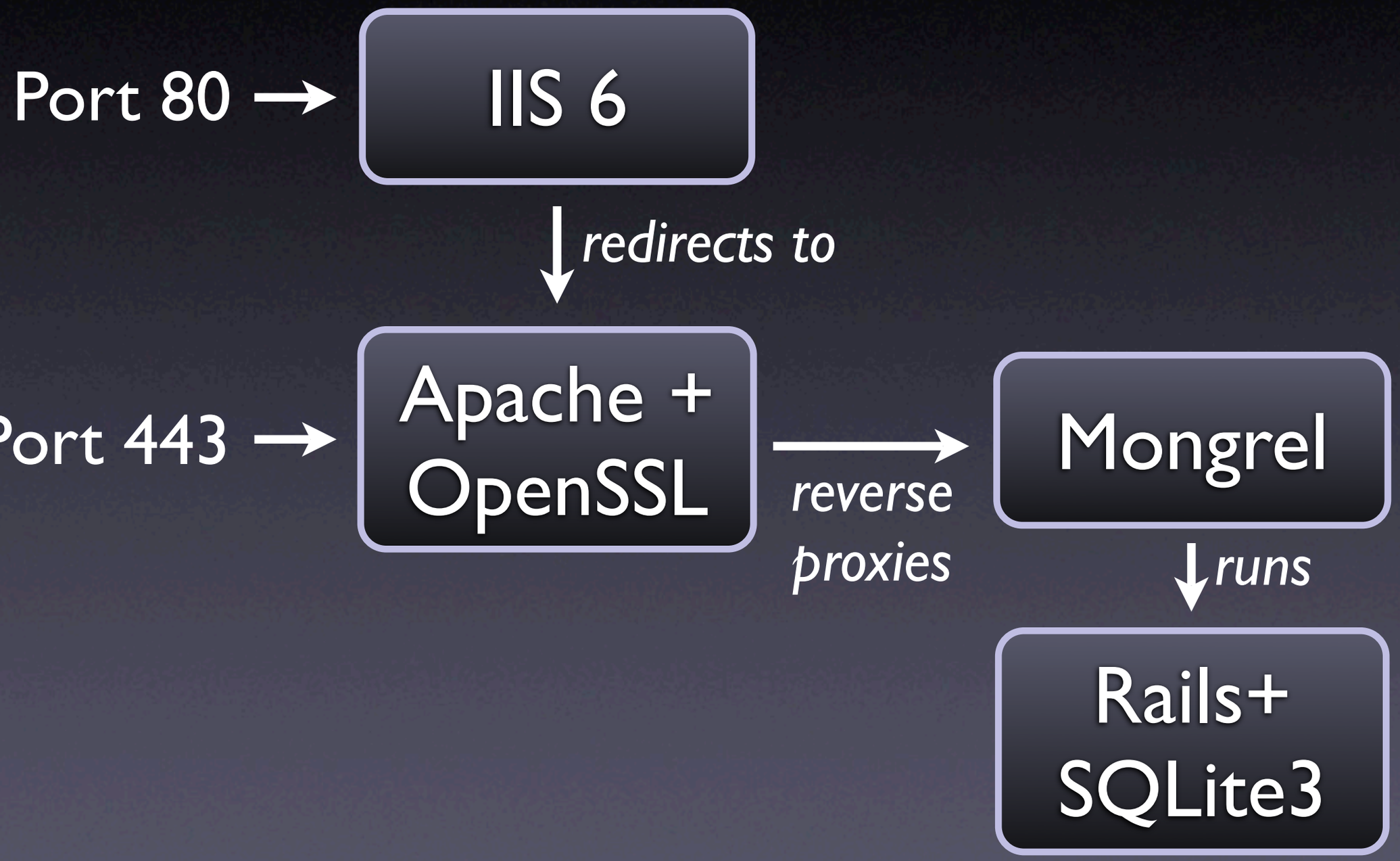
You've got a Rails app already, so use it!

```
# Send
Net::HTTP.post_form(
    URI.parse("http://127.0.0.1:3000/message/new"),
    :yaml => YAML.dump(@message))

# Receive
if (params[:yaml])
  Message::create(YAML.load(params[:yaml]))
end
```

Other options: DRb, Direct to database, Files...

Server Setup



Authentication + Authorization

Windows shops use Active Directory

Authentication: `LogonUser()`
(Win32 API)

Authorization: attempt to use resource only accessible by that user

Authentication + Authorization *(eek!)*

```
require 'Win32API'

class Login
  # These constants from Win32 API headers (winbase.h)
  #
  LOGON32_LOGON_NETWORK = 3
  LOGON32_PROVIDER_DEFAULT = 0

  def raise_with_error_code(message)
    get_last_error = Win32API.new("kernel32", "GetLastError", [], 'L')
    code = get_last_error.call

    raise SecurityError, "#{message} (#{code})"
  end

  def authorize(user, domain, password)
    RAILS_DEFAULT_LOGGER.info("Attempting to authorize user #{user}")

    # Set up Win32 API calls
    #
    logon_user = Win32API.new("advapi32", "LogonUser", ['P', 'P', 'P', 'L', 'L', 'P'], 'L')
    close_handle = Win32API.new("kernel32", "CloseHandle", ['P'], 'V')
    impersonate_logged_on_user = Win32API.new("advapi32", "ImpersonateLoggedOnUser", ['P'], 'L')
    revert_to_self = Win32API.new("advapi32", "RevertToSelf", [], 'L')

    # Get user impersonation token if user/domain/password are valid
    #
    token = '    ' # space for token
    ret = logon_user.call(user, domain, password, LOGON32_LOGON_NETWORK, LOGON32_PROVIDER_DEFAULT, token)
    token = token.unpack('L')[0]
    raise_with_error_code("Could not authenticate user") if (ret == 0)

    # Switch current thread to the security context of that user
    #
    ret = impersonate_logged_on_user.call(token)
    raise_with_error_code("Could not switch to user context for authorization") if (ret == 0)

    # Attempt to open our magic file which determines who can access BlueScale
    #
    authorized = false
    begin
      File.open("your-magic-file-here.txt") do |file|
        RAILS_DEFAULT_LOGGER.info("Authorization successful")
        authorized = true
      end
    rescue
      RAILS_DEFAULT_LOGGER.info("Failed authorization")
    end

    # Revert back to original security context
    #
    revert_to_self.call
    close_handle.call(token)

    raise(SecurityError, "User #{domain}\\#{user} is not authorized to access BlueScale") unless authorized
  end
end
```

and that brings us to...

Ruby on Windows

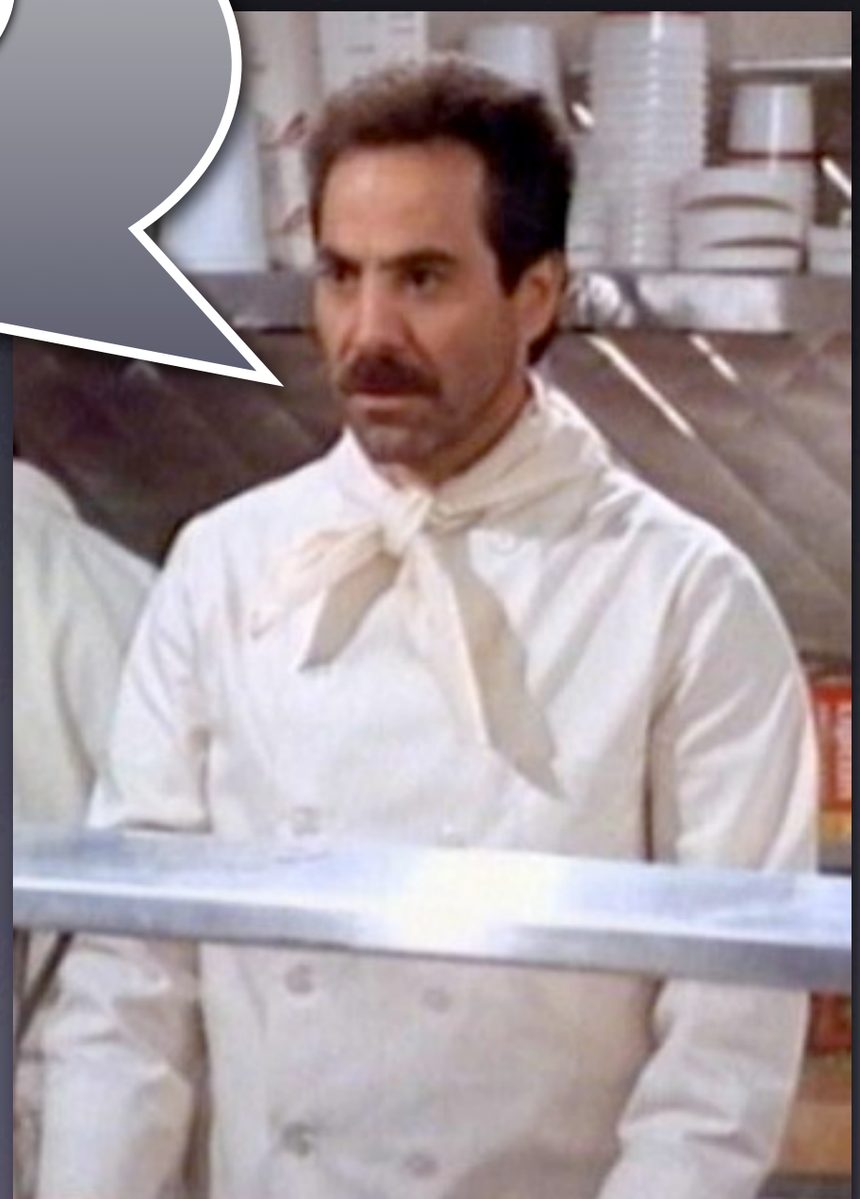
Good news: *most* things work

Bad news: but not everything

(plus there's Windows)

Ruby on Windows

No `fork()`
for you!



(and threads suck for our purposes, too.)

Ruby Threads vs. IO

Blocking IO call on one thread

==

frozen Ruby app

(therefore, you need processes)

Ruby Win32::Service Flaky

Services start but don't stop

Bonus feature:

No way to nicely (but assuredly) kill a Ruby process from a Windows service

Creating a Ruby Process

```
STARTUPINFO         startupInfo;
PROCESS_INFORMATION processInfo;
memset(&startupInfo, 0, sizeof(STARTUPINFO));
memset(&processInfo, 0, sizeof(PROCESS_INFORMATION));
startupInfo.dwFlags = STARTF_USESHOWWINDOW;
startupInfo.wShowWindow = SW_HIDE;
```

```
BOOL success = CreateProcess(
    NULL,                           // pointer to name of executable
    (...),                          // pointer to command line string
    NULL,                           // process security attributes
    NULL,                           // thread security attributes
    FALSE,                          // handle inheritance flag
    CREATE_NEW_PROCESS_GROUP,       // creation flags
    NULL,                           // pointer to new environment block
    (...),                          // pointer to current directory name
    &startupInfo,                   // pointer to STARTUPINFO
    &processInfo                    // pointer to PROCESS_INFORMATION
);
```

Killing a Ruby Process

Works in normal application land:

```
CloseHandle(processInfo.hThread);
CloseHandle(processInfo.hProcess);
GenerateConsoleCtrlEvent(CTRL_BREAK_EVENT,
                         processInfo.dwProcessId);
```

Needed in service land (a.k.a. *kill -9*):

```
TerminateThread(processInfo.hThread, 0);
CloseHandle(processInfo.hThread);
TerminateProcess(processInfo.hProcess, 0);
CloseHandle(processInfo.hProcess);
```

Windows Registry

```
require 'win32/registry'

keyname = 'SYSTEM\CurrentControlSet\Control\Session Manager\Environment'
access  = Win32::Registry::KEY_ALL_ACCESS

Win32::Registry::HKEY_LOCAL_MACHINE.open(keyname, access) do |reg|
  # Add to path
  ['c:\foo\bin', 'c:\bar\bin'].each do |directory|
    unless (reg['path'].include? directory)
      reg['path'] += ';' + directory
    end
  end

  # Add environment variable
  reg['foobar'] = '42'

  # Note: won't take effect until restart!
end
```

Windows Management Instrumentation (WMI)

```
require 'win32ole'
WIN32OLE.connect("winmgmts:\\\\.").
InstancesOf("Win32_Service").each do |s|
  puts s.Caption
  puts s.Description
  puts s.StartMode
  puts s.State
  # ...
end
```

(See MSDN for all the stuff available via WMI)

Win32 vs. Cygwin

Cygwin == *danger!*

(at least that was my experience)

Questions?