

# Supervisor as a Platform

OSCON 2008  
Mike Naberezny



<http://maintainable.com>

# About Me

- <http://mikenaberezny.com>
- <http://maintainable.com>
- <http://ohloh.net/accounts/mnaberez>

# Goals

- Quickly learn the basics of Supervisor and how to get it up & running.
- Explore the advantages of writing programs specifically designed to take advantage of Supervisor's capabilities.

# Agenda

- Supervisor Basics
- Remote Control via XML-RPC
- XML-RPC Interface Extensions
- Event Notification System
- Q&A

# Supervisor Basics

# Supervisor Basics

- Supervisor is a Python program that allows you to start, stop, and restart other programs on UNIX systems. It can restart crashed processes. Built on Medusa.
- Comparable programs: daemontools, launchd, runit.
- Not focused on being "pid 0". Instead, focused on supporting ad-hoc projects and extensibility.

# Supervisor Components

- 'supervisord' is the daemon program. Runs arbitrary programs as child processes.
- 'supervisorctl' is a client program. Allows users to control state of supervisord children and view logs.
- Web interface: start/stop/restart/view logs.
- XML-RPC interface: arbitrary commands.

# Supervisor Config File

```
[inet_http_server]
port=127.0.0.1:9001

[supervisord]
logfile=/tmp/supervisord.log

[program:cat]
command=/bin/cat
autostart=True

[rpcinterface:supervisor]
supervisor.rpcinterface_factory = ...
```

# Security

- By default, users cannot start **arbitrary** processes. They can stop, start, restart, get info on the set of predefined processes.
- Username and password auth can be configured for through-the-web manipulation (XML-RPC and web interface).
- UNIX sockets can be used rather than TCP sockets (or both at the same time).

# Starting Supervisord

```
[mnaberez@box sup]$ bin/supervisord -n -c sample.conf
2007-11-12 12:11:32,581 INFO RPC interface 'supervisor' initialized
2007-11-12 12:11:32,581 CRIT Server 'inet_http_server' running without any
HTTP authentication checking
2007-11-12 12:11:32,582 INFO RPC interface 'supervisor' initialized
2007-11-12 12:11:32,583 INFO supervisord started with pid 13353
2007-11-12 12:11:33,586 INFO spawned: 'cat' with pid 13355
2007-11-12 12:11:34,589 INFO success: cat entered RUNNING state, process has
stayed up for > than 1 seconds (startsecs)
```



# Demo

# Remote Control via XML-RPC

# XML-RPC Interfaces

- All common supervisor operations are scriptable via XML-RPC.
- `supervisorctl` communicates with `supervisord` using XML-RPC. You do everything that `supervisorctl` can do.
- e.g. `supervisor.startProcess()`,  
`supervisor.stopProcess()`,  
`supervisor.readProcessLog()`

# Example: Inspecting a Process

```
>>> import xmlrpclib
>>> s = xmlrpclib.ServerProxy('http://localhost:9001')

>>> s.supervisor.getProcessInfo('cat')

[{'statename': 'RUNNING',
  'group': 'cat',
  'name': 'cat',
  'stop': 0,
  'stderr_logfile': '/path/to/the/log'
  # ...
}]
```

# Example: Stopping a Process

```
>>> import xmlrpclib
>>> s = xmlrpclib.ServerProxy('http://localhost:9001')

>>> s.supervisor.stopProcess('cat')
True
```

# Example: System Introspection

```
>>> import xmlrpclib
>>> s = xmlrpclib.ServerProxy('http://localhost:9001')

>>> s.system.methodHelp('supervisor.getAllProcessInfo')
'Get info about all processes\n\n
@return array result An array of process status results\n'
```

# XML-RPC Namespaces

- Supervisor's XML-RPC interface is divided into namespaces: built-in or your extensions
- Built-in Namespaces
  - `supervisor` namespace has all of the core functions for controlling processes.
  - `system` namespace has introspection functions. Try `system.listMethods()`

# Extending the XML-RPC Interface

# Extending Supervisor With New RPC Interfaces

- RPC interface 'namespace interfaces' may be plugged in to Supervisor.
- Arbitrary functionality may be added.
- Functionality usually "meta-process"
- Code can be difficult to write because it cannot block.

# Registering a Namespace

In the `supervisord.conf` configuration file:

```
[rpcinterface:thenamespace]
supervisor.rpcinterface_factory = <callable>
a_config_option = foo
another_config_option = bar
```

The `<callable>` is a factory that returns your custom RPC namespace instance.

# Registering a Namespace

```
supervisor.rpcinterface_factory =  
    supervisor_twiddler.make_twiddler_rpcinterface
```

- The `<callable>` uses a dot notation that specifies any module in the `PYTHONPATH`.
- Easily package, install, and enable/disable your own extensions without modifying Supervisor.

# RPC Namespace Factory

```
def make_twiddler_rpcinterface(supervisord, **config):  
    return TwiddlerNamespaceRPCInterface(supervisord, **config)
```

- Supervisor injects its instance as the first argument. Everything important is available through this.
- Keyword arguments (\*\*config) are built from the options in the supervisord.conf under your rpcinterface section.
- Methods of the instance returned by the factory are then available through RPC. Pseudo-private (“\_”) methods are not accessible through the interface.

# Async Means No Blocking!

- Supervisor is single-threaded. Methods of custom RPC interfaces cannot block.
- `NOT_DONE_YET` sentinel allows functions that would block to be called periodically until they are done working.
- Example: `supervisor.rpcinterface's SupervisorNamespaceRPCInterface` class' `stopProcess()` method.

# Extensions Available

- supervisor\_twiddler
  - Manipulate Supervisor's program definitions in arbitrary ways without restarting
  - Probably not for high security environments
- supervisor\_cache
  - Store data in Supervisor as key/value pairs
  - Good starting point - very simple extension

# Demo

# Event Notification System

# Supervisor Events

- Events happen as Supervisor runs normally.
- Supervisor itself defines all event types, e.g. `PROCESS_STATE_CHANGE`, when a process changes its state.
- All important state changes of Supervisor and its processes are fired as events.

# Event Listeners

- An event listener is a process that runs under Supervisor.
- This process can be written in any language. Communication with Supervisor is a simple text protocol.
- A module (`childutils`) is packaged with Supervisor to help writing these in Python.

# Event Listener Config

In the `supervisord.conf` configuration file:

```
[eventlistener:listen_for_proc_state_change]
  command=/bin/on_state_change
  process_name=%(program_name)s_%(process_num)02d
  numprocs=5
  events=PROCESS_STATE_CHANGE
```

The event listener can subscribe to any or all of the Supervisor event types.

# Event Listener Sample

Python example, using childutils helper module:

```
import os
from supervisor import childutils

def main():
    while 1:
        headers, payload = childutils.listener.wait()
        ename = headers['eventname']
        if ename.startswith('PROCESS_COMMUNICATION'):
            pheaders, pdata = childutils.eventdata(payload)
            print pheaders, pdata
            childutils.listener.ok()
```

# Practical Uses

- Monitor subprocess memory usage and kill off or restart a process consuming "too much" memory.
- Provision new instances of programs "on the fly" based on usage statistics.
- Bidirectional communications between "normal" supervisor-managed processes and event listener processes.

# Q&A

# Thanks!

- Supervisor & Extensions
  - <http://supervisord.org>
  - <http://maintainable.com/software>
- Contacts
  - [chrism@agendaless.com](mailto:chrism@agendaless.com)
  - [mike@maintainable.com](mailto:mike@maintainable.com)