Schedule::Pluggable

My first CPAN module
Built on MooseX::Workers

Provides a simple interface to run processes

Easily customisable
Provides just three methods

run_in_series
run_in_parallel
run_schedule

run_in_series and run_in_parallel are utility methods which call run_schedule to run the jobs
Interface

each method expects either :-

• a list of jobs to run
• a reference to a list of jobs to run
• a reference to a hash keyed on job name

each job can be :-

1. a scalar value containing the path to an executable to run or a code reference to some code to be run

2. an anonymous hash containing at least on value - ‘command’ containing the details as per 1.
Specifying Jobs by a hash

- name => 'A Job name', # defaults to Job$n where $n is Job number
- command => '<path to an executable> [<param>] [<param> ..]' or a code ref
- params => (list of parameters),
- groups => (list of groups),
- prerequisites => (list of jobs or groups which must succeed first),
- dependencies => (list of jobs or groups which await this job succeeding),

Example

use Schedule::Pluggable;
my $s = Schedule::Pluggable->new();
$s->run_schedule( [ { name => 'First', command => 'echo Hello' },
                    { name => 'Second', command => 'echo World' } ]);
Examples

use Schedule::Pluggable;
my $s = Schedule::Pluggable->new();

$s->run_in_series([ 'echo Hello', 'echo World' ]); 

$s->run_in_series( [ { name => 'First', command => 'echo Hello' },
                     { name => 'Second', command => 'echo World' } ] );

$s->run_in_series( { First => 'echo Hello', Job2 => 'echo World' } );
More Examples

```php
$s->run_schedule([  
    { name => 'First',  
      command => 'echo',  
      params => ['Hello'],  
      dependency => 'Second',  
    },  
    { name => 'Second',  
      command => 'echo World' },  
  ]);

$s->run_schedule([  
    { name => 'First',  
      command => 'echo',  
      params => ['Hello'],  
    },  
    { name => 'Second',  
      command => 'echo World' },  
    prerequisite => 'First',  
  ]);
```
Using Groups

```perl
$s->run_schedule(
    [ { name => 'First', command => 'echo Hello', },
      { name => 'Second', command => 'echo World', prerequisite => 'First', },
      { name => 'Third', command => 'echo Something else', prerequisite => 'First', },
    ];

$s->run_schedule(
    [ { name => 'First', command => 'echo Hello', dependency => 'Rest', },
      { name => 'Second', command => 'echo World', groups => ['Rest'], },
      { name => 'Third', command => 'echo Something else', groups => ['Rest'], },
    ];
```
But why Schedule::Pluggable?

The default behaviour can easily be overridden by using Plugins. There are two Plugin types available:

JobsPlugin - controls where the jobs configuration comes from
EventsPlugin - controls what happens when an event occurs

The JobsPlugin is required to provide a single method - `get_job_config` which is expected to return a reference to either a hash of an array containing the jobs to run. By default JobsPlugin is set to `JobsFromData` which means that the plugin Schedule::Pluggable::Plugins::JobsFromData is loaded.

There are two alternative JobsPlugin provided:

JobsFromXML and JobsFromXMLTemplate both of which obtain the jobs configuration from a file containing XML the latter also passes the file through Template Toolkit before processing allowing you to make the definition dynamic.
Jobs from XML

use Schedule::Pluggable (JobsConfig => 'JobsFromXML');
my $p = Schedule::Pluggable->new;
my $status = $p->run_schedule({XMLFile => 'path to xml file'});

XMLFile in following format :-
<?xml version="1.0"?>
<Jobs>
  <Job name='Job1' command='<command1>'>
    <params>3</params>
    <dependencies>second</dependencies>
  </Job>
  <Job name='Job2' command='<command2> '>
    <params>3</params>
    <group>second</group>
  </Job>
  ...
</Jobs>
EventsPlugin

Enables handling of any event which occurs. By default the event handler simply outputs what has occurred to stdout, a supplied file handle or Log4perl handle.

By supplying your own plugin you can make it do whatever you want.

e.g.

Update a database, update memcached for dynamic display on an ajax web page or send emails on error
package Schedule::Pluggable::Plugin::DefaultEventHandler;
use Moose::Role;

# event_handler is passed a Schedule::Pluggable object ref and a has of parameters including :-
#

sub event_handler {
  my $self = shift;
  my %params = @_; 
  return if exists $params{JobName} and $params{JobName} =~ m/^MonitorJobs$/i;
  return if $self->EventsToReport =~ m/^none$/i;
  my $event = $params{Event};
  return if $self->EventsToReport !~ m!^all$!i and $self->EventsToReport !~ m!evento\b!i and
  $self->EventsToReport !~ m!$event\b!i;
  my %whatreport = ( 
    JobQueued => [qw/ Event JobName Command /], 
    JobStarted => [qw/ Event JobName Command /],
    JobDone => [qw/ Event JobName Command /],
    JobStderr => [qw/ Event JobName Stderr /],
    JobStdout => [qw/ Event JobName Stdout /],
    JobFailed => [qw/ Event JobName Command ReturnValue Stderr /],
    JobSucceeded => [qw/ Event JobName Command /],
    MaxJobsReached => [qw/ Event /],
    ManagerStart => [qw/ Event /],
    ManagerStart => [qw/ Event /],
  );
  1;