ProducerFactory

exchange_name
routing_key
publish_tempo_sec
serialization_function
amqp_url
encryption_key (from subsclass)

make_amqp_url(ip)
make_producer()

Note..
keeps track of everything as raw IP addresses to avoid sending two messages to the same URL

ProducerDiscovery(
producer_discovery_queue,
mesh_routing_key,
snoop_trace_routing_key,
snoop_spy_routing_key,
mesh_exchange_name,
snoop_trace_exchange_name,
snoop_spy_exchange_name,
mesh_serialization_function,
snoop_trace_serialization_function,
snoop_spy_serialization_function)

ProducerFactoryAggregator

mesh_producers
snoop_trace_producers
snoop_spy_producers
set_of_ips
ips
amqp_ip_urls
lan_discovered
mannal_discovered
producers

make_amqp_url(ip_address)
get_ip_for_hostname()
make_amqp_url
make_mesh_producer(ip)
make_snoop_trace_producer(ip)
make_snoop_spy_producer(ip)

MeshProducerFactory

SnoopTraceProducerFactory

SnoopSpyProducerFactory

ProducerFactoryChart

Event Processor

producer discovery

```
entry /
  chart.mesh_producers = []
  chart.snoop_trace_producers = []
  chart.snoop_spy_producers = []
  chart.subscribe(Event(signals.CONNECTION_DISCOVERY))
  if not hasattr(chart, 'man_net_chart'):
    chart.manual_chart = ManNetChart(
      chart.routing_key, chart.exchange_name)

  if not hasattr(chart, 'lan_chart'):
    chart.lan_chart = LanChart(
      chart.routing_key, chart.exchange_name)

  chart.set_of_ips = set([])

CONNECTION_DISCOVERY as e /
  set_of_payload_ips = set([chart.get_ip_for_hostname(host) for host in e.payload.hosts])
  chart.set_of_new_ips = set_of_payload_ips - chart.set_of_ips
  chart.set_of_ips |= set_of_payload_ips

  if(len(chart.set_of_new_ips) > 0):
    chart.post_fifo(Event(signals.ips_discovered))

  # let garbage collector remove charts if they have done their work
  if e.payload.dispatcher == 'man_net_chart':
    del chart.man_net_chart
  elif e.payload.dispatcher == 'lan_chart':
    def chart.lan_chart
```

ConnectionDiscoveryPayload = \
namedtuple('ConnectionDiscoveryPayload',
['hosts', 'amqp_urls', 'dispatcher'])

ips_discovered

ready

ips_discovered

post_to_queue

```
CONNECTION_DISCOVERED /
  chart.defer(e)

exit /
  chart.set_of_new_ips = set([])
  chart.recall()
```

refactor_producers

```
entry /
  new_ips = list(chart.set_of_new_ips)
  new_mesh_producers = \
    [self.make_mesh_producer(ip) for ip in new_ips]
  new_snoop_trace_producers = \
    [self.make_snoop_trace_producer(ip) for ip in new_ips]
  new_snoop_spy_producers = \
    [self.make_snoop_spy_producer(ip) for ip in new_ips]
  chart.mesh_producers.append(new_mesh_producers)
  chart.snoop_trace_producers.append(new_snoop_trace_producers)
  chart.snoop_spy_producers.append(new_snoop_spy_producers)
  chart.producer_discovery_queue.put(
    ProducerQueue(
      mesh_producers=chart.mesh_producers,
    snoop_trace_producers=chart.snoop_trace_producers,
    snoop_spy_producers=chart.snoop_spy_producers))
  try:
    chart.producer_queue.put(payload, block=False)
  except:
    chart.post_fifo(
      Event(signals.ips_discovered,
        times=1, period=random.uniform(0.1,1), deferred=True)
  else:
    chart.post_fifo(Event(signals.ready))
```

ProducerQueue = \
namedtuple('ProducerQueue',
['mesh_producers', 'snoop_trace_producers', 'snoop_spy_producers'])