Factory

MirosRabbitLan

make_amqp_url(ip_address)
change_time_out_in_minutes(mins)

routing_key
exchange_name
dict
addresses
amqp_urls
time_out_in_minutes

LanChart

```
LanChart.DEFAULT_JSON = '''
{
  "addresses": [
  ],
  "amqp_urls": [
  ],
  "time_out_in_minutes": 0
}
'''
```

Event Processor

RecceCompletePayload = \
  namedtuple(
    'RecceCompletePayload',
      ['other_addresses', 'my_address'])

CacheWritePayload = \
  namedtuple('CacheWritePayload', ['json'])

read_or_discover_network_details

```
entry /
  if not hasattr(chart, 'cache_file_chart'):
    chart.file_path = '.miros_rabbimq_lan_cache.json'
    chart.file_name = os.path.basename(chart.file_path)
    chart.cache_file_chart = CacheFileChart(
      file_path=chart.file_path, default_json=LanChart.DEFAULT_JSON)
  if not hasattr(chart, 'rabbitmq_lan_recce_chart'):
    chart.rabbitmq_lan_recce_chart = LanRecceChart(
      chart.routing_key,chart.exchange_name)
  chart.subscribe(Event(signal=signals.LAN_RECCE_COMPLETE))
  chart.subscribe(Event(signal=signals.CACHE))
  chart.publish(Event(signals.CACHE_FILE_READ))

connection_discovered /
  payload = ConnectionDiscoveryPayload(
    ip_addresses=chart.addresses,
    amqp_urls=chart.amqp_urls,
    from=chart.name)
  chart.publish(
    Event(signal=signals.CONNECTION_DISCOVERY, payload=payload)
```

discover_network

```
entry /
  chart.publish(Event(signals.RECCE_LAN))

LAN_RECCE_COMPLETE as e /
  chart.addresses = e.payload.other_addresses
  chart.amqp_urls = [chart.make_amqp_url(a) for a in chart.addresses]
  chart.post_fifo(connections_discovered)
  chart.dict['addresses']=chart.addreses
  chart.dict['amqp_urls']=chart.amqp_urls
  chart.dict['time_out_minutes]=chart.time_out_in_minutes
  payload = CacheWritePayload(json=json.dumps(chart.dict))
  chart.publish(
    Event(signal=signals.CACHE_FILE_WRITE),
    payload=payload)
```

CACHE_FILE_WRITE(<CacheWritePayload>)

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

[e.payload.expired]

chart.addresses = e.payload.dict['addresses']
chart.amqp_urls = e.payload.dict['amqp_urls']
chart.post_fifo(Event(signals.connections_discovered))

CACHE as e: \
  [e.payload.file_name == chart.file_name]

CacheReadPayload = \
  namedtuple('CacheReadPayload',
    ['dict', 'last_modified', 'created_at', 'expired', 'file_name'])

CONNECTION_DISCOVERY(<ConnectionDiscoveryPayload>)