pyimq Documentation

Release 0.0.5

Michael van der Westhuizen

October 21, 2016

Contents

This documentation is a work-in-progress, and will be updated frequently

pyimq is a set of Python bindings to Open Message Queue (also known as the Glassfish Message Queue, formerly known as the Sun Java System Message Queue and informally known as IMQ).

This software wraps the Sun GlassFish Message Queue C runtime library in Python, modelling the implied object oriented design of the C interface.

The most common message queue usage patterns are supported, including asynchronous message consumers.

CHAPTER 1

Contents

1.1 Getting Started

If you are already familiar with the OpenMQ C runtime library you should still read this section to familiarise yourself with the mqcrt object hierarchy as exposed to Python.

If you are not familiar with the OpenMQ C runtime library you may want to open a copy of the mqcrt documentation in another tab.

1.1.1 Prerequisites

To use the *pyimq* module you'll need access to an OpenMQ installation (or Sun Java System Message Queue, or Glassfish Message Queue). If you do not yet have access to such a beast, this document will help you to get a simple OpenMQ instance up and running.

You'll also need to OpenMQ C runtime library, as the pyimq module uses this C library to access the message queue broker.

The OpenMQ C runtime library (mqcrt) is distributed along with platform specific builds of OpenMQ, but is often not quite what you really need to build a Python extension module against, so if you need to rebuild mqcrt you'll need a C++ compiler and some patience.

Finally, if you're running the OpenMQ broker (server) on your machine you'll need a Java installation. If you plan to build mqcrt you'll also need a Java JDK and Ant.

1.1.2 Obtaining OpenMQ

notes on downloading and running an instance of OpenMQ

1. Download...
2. Unzip...
3. Run...

1.1.3 Building the Message Queue Runtime Library

notes on building, when you need to, and where to find more information

prereqs

1. Extract
2. Patch
3. Configure
4. Build
5. Install

1.1.4 Running OpenMQ

a quick introduction to running an instance

1.1.5 Writing a Producer

how to write a producer

1.1.6 Writing a Consumer

how to write a consumer

CHAPTER 2

A Quick Example

This example shows a simple consumer, which simply retrieves messages and prints them.

```
#!/usr/bin/env python

import sys, os, optparse

if not 'MQ_LOG_LEVEL' in os.environ:
    os.environ['MQ_LOG_LEVEL'] = '-1'

from imq import MQConnection, MQMessage
from imq import MQ_BROKER_HOST_PROPERTY, MQ_BROKER_PORT_PROPERTY, \
    MQ_CONNECTION_TYPE_PROPERTY, MQ_CLIENT_ACKNOWLEDGE, MQ_SESSION_SYNC_RECEIVE, \
    MQ_QUEUE_DESTINATION, MQ_TOPIC_DESTINATION, MQ_TEXT_MESSAGE
from imq import MQ_STRING_TYPE, MQ_INT32_TYPE

def consumer(host, port, name, type):
    conn = MQConnection({MQ_BROKER_HOST_PROPERTY: (MQ_STRING_TYPE, host),
            MQ_BROKER_PORT_PROPERTY: (MQ_INT32_TYPE, port),
            MQ_CONNECTION_TYPE_PROPERTY: (MQ_STRING_TYPE, "TCP")}, "guest", "guest")
    sess = conn.create_session(False, MQ_CLIENT_ACKNOWLEDGE, MQ_SESSION_SYNC_RECEIVE)
    cons = sess.create_consumer(sess.create_destination(name, type))
    conn.start()

    while True:
        mess = cons.receive_message_wait()

        if mess.type() != MQ_TEXT_MESSAGE:
            mess.acknowledge_messages()
            continue

        text = mess.get_text()
        sys.stdout.write("Received message: " + mess.get_text() + "\n")
        sys.stdout.flush()
        mess.acknowledge_messages()


if __name__ == '__main__':
    parser = optparse.OptionParser()
    parser.add_option("--host", action="store", dest="host", default="localhost")
    parser.add_option("--port", action="store", dest="port", default=7676, type="int")
    parser.add_option("--destination_name", action="store", dest="destination_name", default="example
    parser.add_option("--type", action="store", dest="type", choices=("q", "t",), default="q")
    (options, args) = parser.parse_args()
```

```
    args = {'host' : options.host, 'port' : options.port, 'name' : options.destination_name,
        'type' : 't' == options.type and MQ_TOPIC_DESTINATION or MQ_QUEUE_DESTINATION}

    consumer(**args)
```

CHAPTER 3

Indices and tables

- genindex
- search