***tclrmq: A pure Tcl library for RabbitMQ***
***Garrett McGrath***

## Introduction

*tclrmq*, a new library for using the Advanced Message Queueing Protocol (AMQP) v0.9.1 with RabbitMQ, provides a pure Tcl interface for this flavor of distributing messaging. Currently the open-source Tcl ecosystem provides support for several varieties of distributed messaging, e.g., *kafkatcl* and *tclzmq*, but nothing regularly maintained or feature-full for AMQP despite it being one of the most common, stable and well known options. With the creation of this library, though, that is now no longer the case. Although *tclrmq* is tightly coupled with RabbitMQ, this is no real limitation: its stability, long history, extensions, and active upkeep make it one of the most trusted open source products, and the open source leader for AMQP. Tcl applications can now make use of it without relying on any external bindings or C libraries. With a standard 8.6.* distribution, Tcl is now capable of utilizing AMQP in any application.

## Why Pure Tcl

When designing a library in Tcl, its high quality C API and the availability of widely used and long developed open source C code for many tasks often makes the writing of a C extension the best and most obvious solution. Hearkening back to Tcl's original use case, in these situations, it is clear that right thing to do is to use Tcl as a wrapper over low level code. The performance and ease of development that results commonly renders the writing of a pure Tcl library from a scratch an infeasible task. However, in the case of RabbitMQ, the primary C library *rabbitmq-c* leaves a bit to be desired. For one, it is still technically in beta, although it has been around for several years. In addition, it is not possible to use it out of the box in a non-blocking way. One can potentially combine its use with an additional library for asynchronous networking like *libevent*, but this adds further and unnecessary complexity. Given these reasons, *tclrmq* is written from scratch, completely in Tcl, supporting all available RabbitMQ protocol extensions and only with non-blocking, asynchronous operations.

## Getting Started

Getting work done with *tclrmq* requires the use of three simple TclOO classes: *Connection, Login and Channel*. The *Connection* class is responsible for establishing the network connection with the RabbitMQ server, both standard and using TLS, while *Login*, which is a helper class for the connection, encapsulates the necessary authentication mechanism for making the connection. Once connected, the *Channel* class is the main workhorse of the library. The vast majority of methods are defined in this class: all logical operations supported by RabbitMQ's implementation of the AMQP v0.9.1 standard are found here. Every *Channel* object refers to a particular *Connection*, and multiple channels can be defined for a given connection if desired by the application creator. Example usage is as follows:

```
package require rmq
```

```
# Arguments: -user -pass -vhost
# All optional and shown with their defaults
set login [Login new -user "guest" -pass "guest" -vhost "/"]

# Pass the login object created above to the Connection
# constructor
set conn [Connection new -host rabbitmq.domain.com -port 5672
-login $login]

# Set TLS options: all available options shown
$conn tlsOptions -cafile "/path" -certfile "/path" -keyfile
"/path" -require 1

# Set a callback for when the connection is ready to use
# which will be passed the connection object
$conn onConnected rmq_conn_ready
proc rmq_conn_ready {conn} {
    puts "Connection ready!"
    # with the connection established, create a channel
    set rChan [Channel new $conn]
}

# Initiate the connection handshake and enter the event loop
$conn connect
vwait die
```

**Setting Callbacks**

Central to the design of *tclrmq* is the use of callbacks. Since everything in the library happens asynchronously, it is necessary to set procs for responding to events as they occur. Most callbacks are invoked on *Channel* objects, but *Connection* objects also provide them for the opening and closing of a connection and whenever an error occurs. *Connection* objects provide callbacks for the same three events, but also for every AMQP method documented in the standard that returns information back to a client application. The electronic proceedings for *tclrmq* documents all arguments required for every possible callback proc, including all the relatively obscure methods that do not find use in the majority of applications. However, the two most common tasks—publishing and consuming—can be accomplished quickly and with little code. Example usage for both these tasks is as follows:

```
set conn [Connection new]
$conn onConnected connected_rmq

proc connected_rmq {conn} {
```

```
    set rChan [Channel new $conn]
    $rChan onOpened publish_to_queue
}

# the channel object created above is passed in
proc publish_to_queue {ch} {
    set pubFlags [list $::rmq::PUBLISH_MANDATORY]
    set pubProps [dict create content-type text/plain]
    set data "textual content"
    # content, exchange name, routing key, flags, properties
    $ch basicPublish $data "tcl_test" hello $pubFlags $pubProps
}

# consuming needs to be setup after a channel has been defined
and fully opened
proc setup_consuming {ch} {
    $ch basicConsume consumer_proc
}

proc consumer_proc {ch methodD frameD data} {
    # do some work with the data consumed and
    # send an acknowledgment
    $ch basicAck [dict get $methodD deliveryTag]
}
```

**Limitations and Future Work**

Currently the only protocol supported by the library is AMQP v0.9.1. This is the most stable version of the protocol in widespread use. It will be supported by the maintainers of RabbitMQ for the foreseeable future. Although AMQP v1.0 has been out since 2011, it breaks functionality with v0.9.1 and earlier versions and implements an entirely separate type of protocol that focuses entirely on the messaging layer. Therefore, this library does not make an attempt to support it, nor has it been designed with the ability to easily adapt the code-base to v1.0. Other protocols supported by RabbitMQ such as STOMP and MQTT are also not supported. Unlike AMQP v1.0, though, future work on these is a possibility and could easily leverage the current code without requiring a substantial redesign.

Future work will focus primarily on improving the reliability of the library. One of the main tasks for this area of improvement is to increase the number of test cases available. In the AMQP standard, a number of test scenarios are detailed. Every one will be eventually covered by forthcoming commits. Supplying code for the complete suite of test scenarios will dramatically increase the current code coverage. In addition, a more formal benchmarking of the library's performance is part of upcoming work. The benchmark will measure the speed of producing and consuming under a variety of workloads.