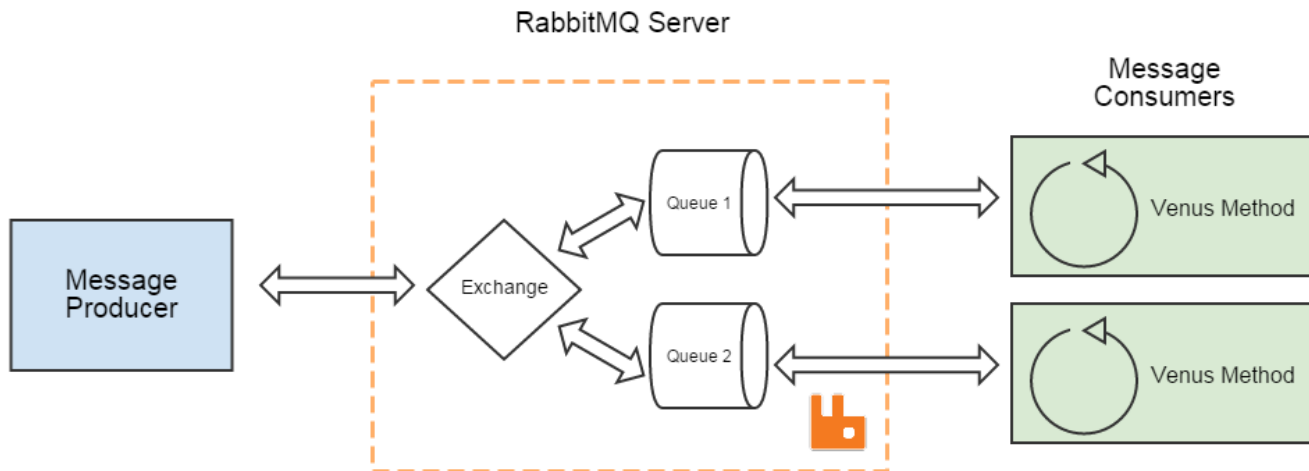


## Overview

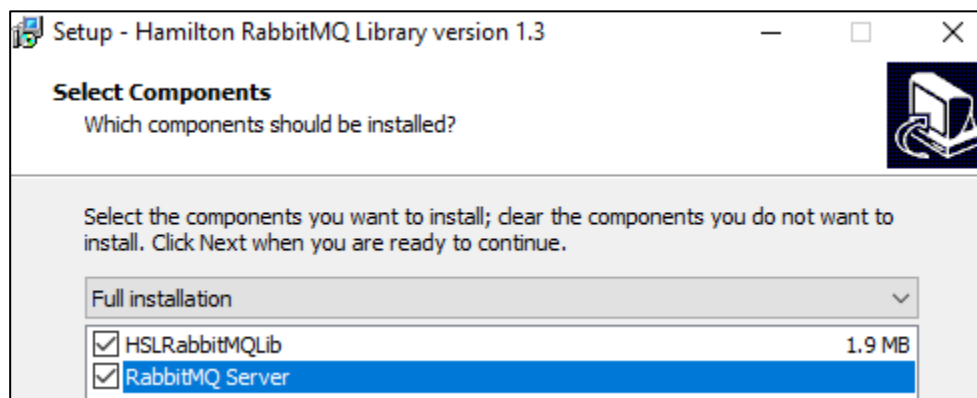
This guide will assist the user in setting up and using a RabbitMQ server alongside the VENUS software. This includes installing and configuring the RabbitMQ application as well as installing the HSLRabbitMQLib library. RabbitMQ is a robust real-time messaging platform written with the goal of facilitating the communication between distributed systems over a network or system components connected to the same computer. The HSLRabbitMQ Library enables VENUS to utilize the messaging functionality of the RabbitMQ messaging platform, allowing for the programming of complex automated systems. More specifically, automated systems which include more than one liquid handler, and one or more robotic arms benefit from improved communication and timing between the individual components.



## RabbitMQ and HSLRabbitMQLib Installation

### 1) Download and Install the HSLRabbitMQ Library and Application

- Download the HSLRabbitMQLib installer from this [link](#). Run the installer as an Administrator. When prompted, select “Full Installation” from the drop-down to install the RabbitMQ server, its prerequisites (Erlang and .NET Framework 4.5), and the HSL RabbitMQ Library.

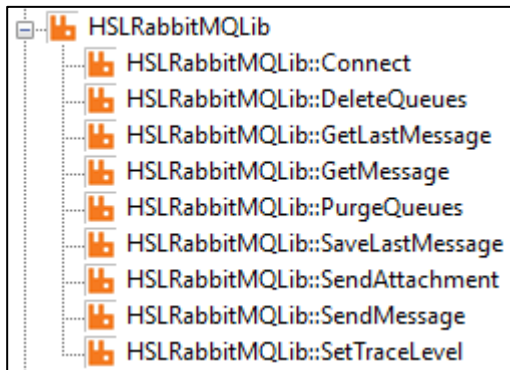


- b) For further instructions to enable the “RabbitMQ Admin Web-Interface” and set a different username and password, refer to Appendix 1. Note that these steps are not required but can be useful for security purposes or to monitor queues/messages when setting up methods in VENUS.

## Using RabbitMQ in VENUS

### 1) Programming with RabbitMQ

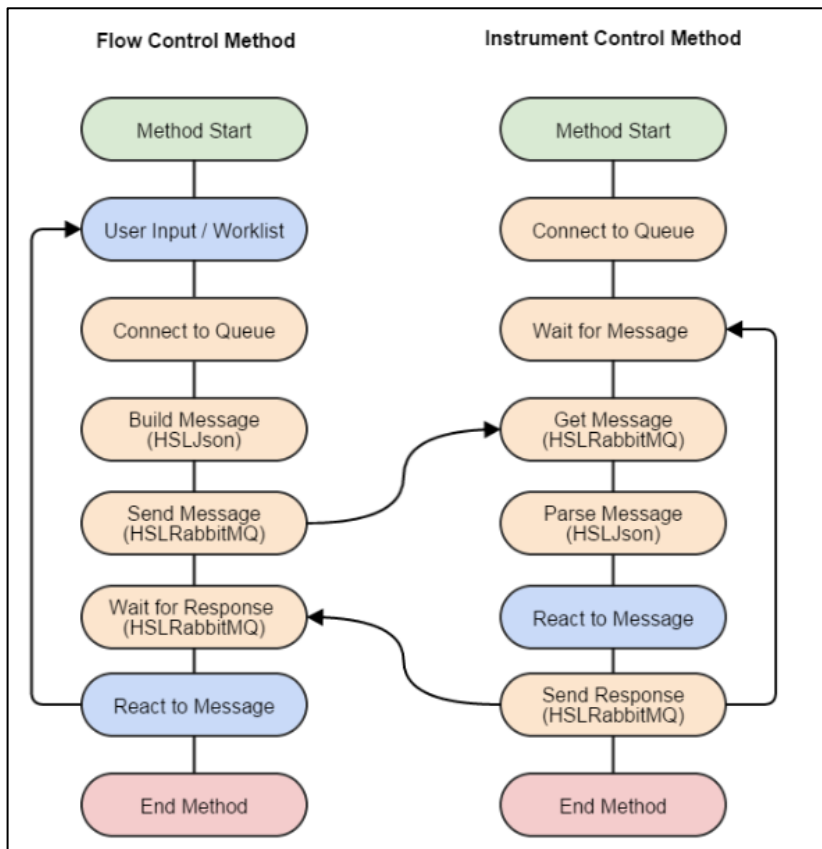
- a) Open a method in VENUS and Add the HSLRabbitMQLib Library to the method, found in the “C:\Program Files (x86)\Hamilton\Library\HSLRabbitMQ” folder.



- b) Refer to the Help file, included with the library and accessible from the individual library steps, for explanation on the individual steps and their required parameters. **NOTE:** by default, the username and password for the ‘Connect’ step are “guest” and “guest”, unless changed using Appendix 1.

- c) RabbitMQ is best used to have two separate methods communicate with each other, meaning two instances of Run Control will be active, potentially on two separate computers. A typical workflow will have the methods connecting to the same server/queue and sending/receiving messages to each other. To facilitate this, one method will often act as a primary which sends a message to the secondary and waits for a response (success or error). The secondary method will wait in an infinite loop checking for new messages, receive the new message, act on the message, and send a response when complete. Refer to the following image.

**NOTE:** “React to Message” refers to the actual instrument steps to be performed (i.e. pipetting, transport, scanning, etc.); also, messages do not need to be in JSON format, however it can be useful to send large amounts of data with multiple, varying tasks.



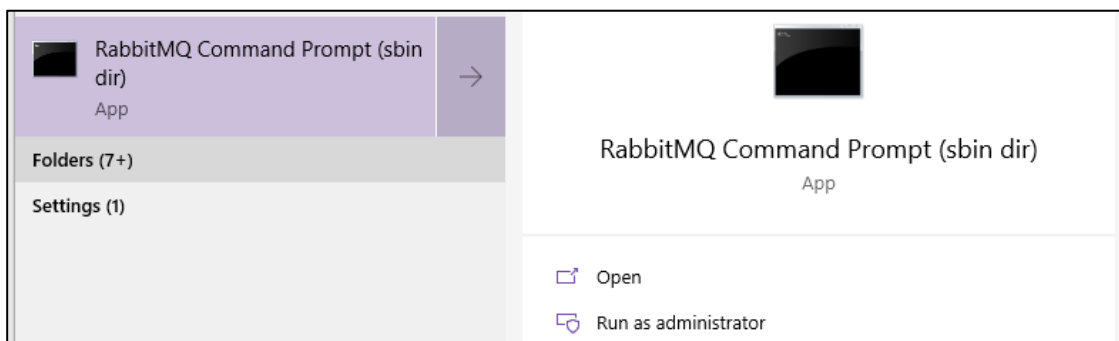
## 2) Determine a Programming System Model

- a) While the programmer is free to use RabbitMQ how they see fit, there are a preexisting communication models that are worth understanding and implementing based on need. Two are outlined below, but there are other options available (i.e. Peer-to-Peer and Coupled Clusters)
  - **Client - Server (Request - Response):** This model is outlined in the workflow above. One method acts as primary sending messages to the secondary, which reacts according to the request, and sends back a response (success/error/data). An example of this in Hardware terms would be a Worklist Processing Method sending a Message to a STAR Method with data on how to proceed. This effectively splits up the data handling and the liquid handling / sample processing functions into two simpler discrete methods.
  - **Actor Model:** This model is more complex than the first. In this case there are multiple Queues and multiple methods interacting with these Queues. An example of this would be a Worklist method sending data to two separate STAR methods, which in turn send messages to a shared HMotion method which retrieves and delivers plates to/from an incubator. While a single STAR and a HMotion can be scheduled and managed with Scheduler, it's difficult to share a resource between two separate schedules. In this case, the HMotion acts as a tertiary, and the STARs (secondaries) connect to multiple queues according to the worklist sent by the primary Worklist Method.

## Appendix

### 1) Enable the “RabbitMQ Admin Web-Interface”

- a) Open the RabbitMQ Command Prompt from the Start Menu as an Administrator



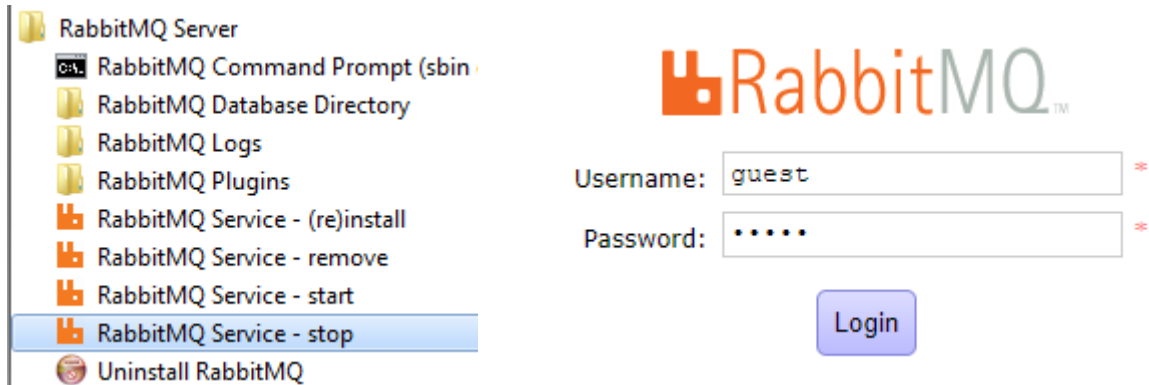
- b) In the window that pops up, type in “rabbitmq-plugins.bat enable rabbitmq\_management” and hit Enter

```
Microsoft Windows [Version 10.0.18362.418]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Program Files (x86)\RabbitMQ Server\rabbitmq_server-3.2.4\sbin>rabbitmq-plugins.bat enable rabbitmq_management
The following plugins have been enabled:
  mochiweb
  webmachine
  rabbitmq_web_dispatch
  amqp_client
  rabbitmq_management_agent
  rabbitmq_management
Plugin configuration has changed. Restart RabbitMQ for changes to take effect.

C:\Program Files (x86)\RabbitMQ Server\rabbitmq_server-3.2.4\sbin>
```

- c) Restart the RabbitMQ Service by using the Stop and Start programs found in the Start Menu, then open an internet browser, and go to "http://127.0.0.1:15672" or "http://localhost:15672". Login with Username: guest; Password: guest.



RabbitMQ Server

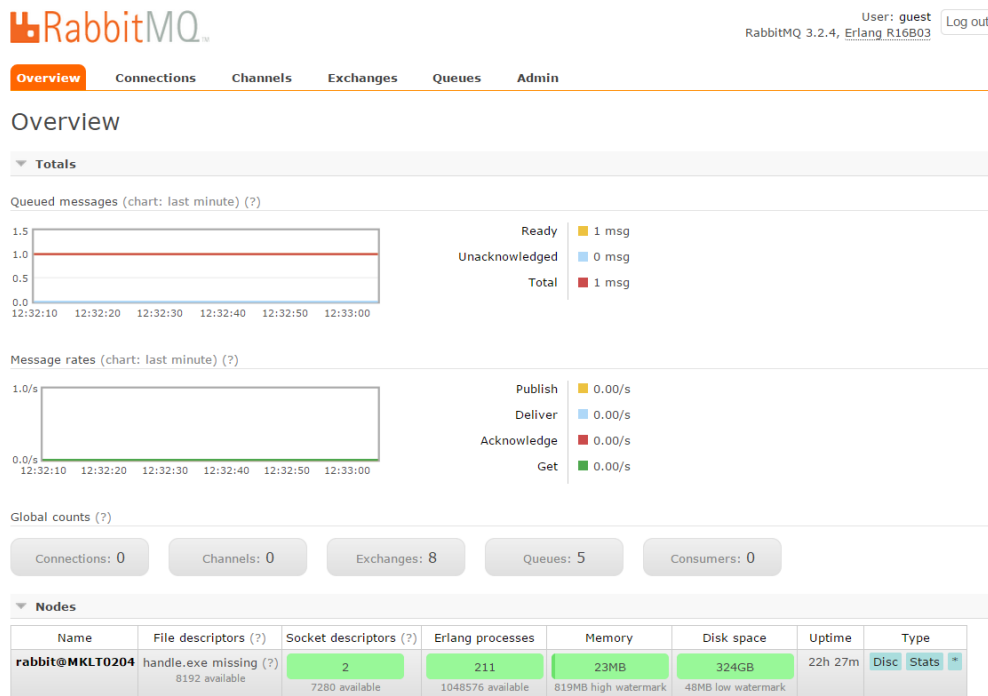
- RabbitMQ Command Prompt (sbin)
- RabbitMQ Database Directory
- RabbitMQ Logs
- RabbitMQ Plugins
- RabbitMQ Service - (re)install
- RabbitMQ Service - remove
- RabbitMQ Service - start
- RabbitMQ Service - stop**
- Uninstall RabbitMQ

Username:

Password:

Login

- d) Use the “Overview” and “Queues” tabs to monitor messages passing to/from the server. Use the “Admin” tab to change/add/remove usernames/passwords.



RabbitMQ

User: guest  
RabbitMQ 3.2.4, Erlang\_R16B03 [Log out](#)

**Overview** | Connections | Channels | Exchanges | Queues | Admin

### Overview

**Totals**

Queued messages (chart: last minute) (?)

Ready: 1 msg  
Unacknowledged: 0 msg  
Total: 1 msg

Message rates (chart: last minute) (?)

Publish: 0.00/s  
Deliver: 0.00/s  
Acknowledge: 0.00/s  
Get: 0.00/s

Global counts (?)

Connections: 0 | Channels: 0 | Exchanges: 8 | Queues: 5 | Consumers: 0

**Nodes**

Name	File descriptors (?)	Socket descriptors (?)	Erlang processes	Memory	Disk space	Uptime	Type
rabbit@MKLT0204	handle.exe missing (?) 8192 available	2 7280 available	211 1048576 available	23MB 819MB high watermark	324GB 48MB low watermark	22h 27m	Disc   Stats

## FAQ

### **Q: What do I include in a message?**

**A:** A string of information that can be parsed by the receiving method, which is unique and specific to ensure the correct reaction is performed by the receiving method. Consider using JSON and the HSLJSON Library to generate/parse large amounts of data/instructions with multiple parameters. Additionally, file attachments can be sent as part of a message (i.e worklists).

### **Q: What are message headers?**

**A:** Headers are used as fields that describe the content of a message, but do not hold the actual content of the message. An example of this would be sending a message with a "type" Header to tell the destination method how to respond. They are not required, and often go unused in simpler processes.

### **Q: What is a Correlation ID?**

**A:** A Correlation ID is an alphanumeric identifier of a message (GUID). Correlation IDs are generated automatically if the field is passed an empty string variable. Correlation IDs can be useful to check whether a received message corresponds to one that was sent. This same functionality can be programmed in using headers and responses, or even message content, but with complex systems, or ones where the order of messages matters, Correlation IDs can become a useful tool for keeping track of messages and their responses.

### **Q: Why are old messages sent in previous methods being retrieved and messing things up?**

**A:** Messages are stored in a queue until they are received or purged. It is prudent to use the PurgeQueues or DeleteQueues steps at the end of a workflow to ensure message crossover does not occur. Queues can also be purged through the Admin Web-Interface.

### **Q: Why am I receiving “left hand side of expression” errors?**

**A:** The variables passed in/out of the HSLRabbitMQ Library steps are data-type specific and require declaration/assignment to the given data type before being used in the step itself. For example, “o\_strErrorMessage”, which returns a description of an error must be declared as a string-type variable using an Assignment step (i.e. t\_strRabbitMQ\_ErrorMessage = “”, in which empty double quotes denotes an empty string).

### **Q: How do I prevent additional RabbitMQ traces from filling my Trace File?**

**A:** Use the SetTraceLevel step from the HSLRabbitMQ Library (or ASW TraceLevel Library) to set the trace level to RELEASE (1) or NONE (0) to limit the information written into the Trace File. It is suggested to use the DEBUG level (2) when testing and developing a method, then setting to a lower level when the additional information is no longer required.

### **Q: Can I send messages to other RabbitMQ computers/servers on the network?**

**A:** Yes, IP Addresses of other computers hosting RabbitMQ can be configured in the RabbitMQ Connect step by changing the “strHostName” parameter to the target IP Address instead of the default “localhost”.