# 0-Meter Documentation

***Release 1.0.0***

**AO**

**Sep 27, 2017**

# Contents:

1 Quickstart 3

2 Configuration 5

3 Features 9

0-Meter is heavily influenced by JMeter, and is designed to be a load testing application for 0MQ-Based Applications.

The python program can read from flat files to build messages, as well as substitute variables found in a CSV file. It can transmit one or more messages to a server all at once, or at scheduled intervals.

The execution of the program is driven by a configuration file, which is read at startup.

CHAPTER 1

# Quickstart

*Go Home*

## Setup

Before downloading 0-Meter, you will need a few things installed:

- Python 2.7 - Can be downloaded [here] (https://www.python.org/)
- pyzmq
- apscheduler

Once you download Python, you can run the below commands to install the other dependencies:

- Linux

*sudo pip install apscheduler*

*sudo pip install pyzmq*

- Windows

*python -m pip install apscheduler python -m pip install pyzmq*

## Use

0-Meter is designed to be easy to use. It is a command line python script, and all of the software is contained within the .py file.

Once the dependencies are installed and the repository cloned, using the script is as easy as running the below command:

*python 0-meter.py config.xml*

## Configuration

Continue on to *Configuration* for more information!

# CHAPTER 2

# Configuration

*Go Home*

## Overview

Configuration of 0-Meter is done via editing of the configuration XML. This file contains a number of fields critical to 0-Meter's execution, all of which are described below.

This methodology of configuration was selected to allow for 0-Meter to be run in a headless environment. If a User Interface is desired, development of one would be easy so long as it simply wrote configuration XML files and then the python script can be executed independently.

Example Configuration Files can be found in the repository as well.

There are five segments to the Configuration XML.

- Behavior - This fundamentally drives the behavior of the program
- Message - This drives how we generate messages to send
- ZeroMQ - Connectivity Information for ZeroMQ
- Logging - Logging Information
- Response - Response Parser configuration

0-Meter has a response parser which can be configured independently of the sender, suggested configurations for both are supplied below. Suggested Configurations include True/False flags for Behavior Segment. It also includes descriptions for required elements in the Message Segment.

All configurations require valid entries for all elements in both ZeroMQ and Logging Segments.

## Suggested Response Parser Configurations

## Parse Responses and Fail on a Failure Response

- [x] Parse_Responses
- [] Print_Response_Keys
- [x] Fail_On_Response

0-Meter can be configured to fail when receiving a response which does not match a specified value. The below values are read from the Response Segment:

- Field_Path - The *Field Paths* string which is used to determine what field in the response is compared against.
- Success_Value - The value to compare against for the field specified

## Parse Responses and Write Keys to CSV

- [x] Parse_Responses
- [x] Print_Response_Keys
- [] Fail_On_Response

0-Meter can be configured to print values from the response message. The below values are read from the Response Segment:

- Key_Path - The *Field Paths* string which is used to determine what field in the response is written to the CSV
- Output_Csv - The name of the CSV file to create with the response keys

## Suggested Sender Configurations

## Send One Message from a Flat File

- [x] Single_Message
- [] Multi_Message
- [] Include_CSV
- [] Span_Over_Interval

Here we send a single message, read from a flat file, on the specified port. The below values are read from the Message Segment:

- Message_Location - The filename & location of the message being sent

## Send A Collection of Messages from Flat Files in a Folder

- [] Single_Message
- [x] Multi_Message

- [] Include_CSV
- [] Span_Over_Interval

Here we send a collection of messages, all at once, read from a collection of flat files in a folder, on the specified port. The below values are read from the Message Segment:

- Message_Folder_Location - The location of the folder containing the messages being sent
- Message_Extension - The extension to look for in the specified file for messages

## Send A Collection of Messages from Flat Files in a Folder, Scheduled over a specified interval of time

- [] Single_Message
- [x] Multi_Message
- [] Include_CSV
- [x] Span_Over_Interval

Here we send a collection of messages, all at once, read from a collection of flat files in a folder, on the specified port. These messages are sent periodically, with the formula being one message per (number_of_messages / interval) seconds. The below values are read from the Message Segment:

- Message_Folder_Location - The location of the folder containing the messages being sent
- Message_Extension - The extension to look for in the specified file for messages
- Interval - The overall interval for messages to be sent

## Send A Collection of Messages from a Flat File & CSV

- [] Single_Message
- [x] Multi_Message
- [x] Include_CSV
- [] Span_Over_Interval

Here we send a collection of messages, all at once, on the specified port. The messages are created by reading a base message, parsing it for defined variables, and substituting them with values from the CSV. Variables are simply names that match the headers of the CSV File, starting with the Variable_Start_Character and ending with the Variable_End_Character. The below values are read from the Message Segment:

- Message_Location - The location of the base message
- CSV_Location - The location & Filename of the CSV to read for variable data
- Variable_Start_Character - The start character of variables in the CSV
- Variable_End_Character - The end character of variables in the CSV

## Send A Collection of Messages from a Flat File & CSV, Scheduled over a specified interval of time

- [] Single_Message
- [x] Multi_Message
- [x] Include_CSV
- [x] Span_Over_Interval

Here we send a collection of messages, all at once, on the specified port. The messages are created by reading a base message, parsing it for defined variables, and substituting them with values from the CSV. Variables are simply names that match the headers of the CSV File, starting with the Variable_Start_Character and ending with the Variable_End_Character. These messages are sent periodically, with the formula being one message per (number_of_messages / interval) seconds. The below values are read from the Message Segment:

- Message_Location - The location of the base message
- CSV_Location - The location & Filename of the CSV to read for variable data
- Variable_Start_Character - The start character of variables in the CSV
- Variable_End_Character - The end character of variables in the CSV
- Interval - The overall interval for messages to be sent

## Field Paths

Field paths are a special syntax for describing document positions within the 0-Meter configuration files.

JSON fields can be contained either in lists or in objects. A field path might look like this:

*responses[0.codes[1*

The use of *[* signifies an array, and the use of . signifies an object. Here's the JSON message this would parse:

{responses: [{codes: [0, 1]}]}

The above field path would return the value '1' from the response message.

CHAPTER 3

# Features

- Send a single 0MQ Request Message with content from a flat file
- Send a set of 0MQ Request Messages with content from a set of flat files contained in a folder
- Send a set of 0MQ Request Messages with base content from a flat file, and variables substituted from a CSV file
- Send a set of 0MQ Request Messages, with content from files in a folder of generated from a CSV, scheduled periodically over a specified interval of time.
- Parse Responses and print to CSV or exit with error code based on results