RenderX EnMasse

EnMasse User Guide

RenderX

Telephone: 1 (650) 328-8000
Fax: 1 (650) 328-8008
Website: http://renderx.com
Email: support@renderx.com

Table of Contents

Chapter 1. Preface

1.1. Abstract

EnMasse is a server-based product for shared document formatting. It accepts documents over the network and formats them by distributing the formatting tasks among multiple computers. Regular users publish customized documents in high volumes and varied formats. Server access is via straightforward controls. The users may customize server operation for their work-flow.

1.2. A Bird's Eye View

Competing programs on the same computer fight for resources, and take more time to return a result.

More memory and newer processors help applications run faster, but there are often memory limits. Processors become faster, but their speed is limited; multiple-processor configurations are expensive, and performance does not rise in line with the price and the number of processors.

A solution is to distribute processing among multiple computers. If you have as many computers as there are programs, each program runs as fast as the computer allows it and uses as much memory as you can install. Using multiple computers, programs perform as well as the current level of technology permits, and can achieve a data throughput as high as you can afford; and the price for stock single-CPU PCs is low enough to have sufficient resources to meet your business needs.

If these are connected so that they work all the time, the load is evenly distributed, and even when nodes fail, or need upgrading and are removed, the system still processes all your requests, preserves your data, and the only change noticed is a slight decrease in performance. This is a well-known but not an easy task; and to really use the joint power of grid computing as it is known, you must resolve this issue.

XEP creates elegantly formatted print-ready views of documents. In many deployments, it produces hundreds of thousands of documents on demand. For example:

- A bank printing monthly statements,
- a technical authoring department preparing documentation in twenty languages for electronic and hard-copy delivery,
- numerous users filing a loan application form on-line at the same time and requesting their own copies concurrently, in a printable form.

These are just a few examples. However fast and optimized the program's code, business needs a way to **scale up** performance so that it handles a growing load.

EnMasse solves this problem for you. For a user or an application programmer, it is a single access point. Through a shared folder, a web form or a network connection (locally on your network, or across the Internet), EnMasse accepts document processing requests and sends them to the XEP formatting engines running on several computers in the local network, then delivers formatted documents back to the user. It monitors the formatting engine performance, notices when they go down and are restored and readjusts the distribution of requests according to the workload. When a server fails in the midst of processing a request, EnMasse resubmits the request to a different server; thus the only impact is a slightly increased response time for that particular document. This provides the security of service needed in today's world.

EnMasse is both opaque and transparent. On one hand, it provides you with **single point** abstraction such that there is no need to worry about the number of engines running, or about their load; EnMasse dispatches requests to the most appropriate node. On the other hand, both the access point and the processing engines have standard, embedded servers. The system administrator can instantly check the status of the grid, identify problems and take appropriate action. An EnMasse access point takes little memory overhead and processing time, it can be deployed on a loaded intranet server, or even on a workstation and as long as the processing engines run on separate machines, it does not affect the speed or throughput of the grid.

For accounting purposes and performance tuning, EnMasse provides a logging facility. One can adjust the extent of the logging, or completely switch it off. The log files are easy to understand by humans and to process by programs.

EnMasse runs on a wide range of hardware and operating systems, easy to install and requires little maintenance. It has run for weeks on a mix of Linux, FreeBSD and Windows nodes wholly unattended, with multiple access points over the same grid when needed. It has proved to be the solution to many performance problems, providing a reliable service without high levels of support.

Chapter 2. The User's View

Internally, EnMasse distributes formatting jobs, logs activity and monitors grid performance. Whatever the system around it is doing, the role of its core remains constant. For the user, EnMasse provides a choice of ways to submit tasks and receive responses. The three current interfaces are the **active folder (Actinia)** , **network server (Toaster)** and the **SOAP server (Fairy)** .

2.1. Actinia

Actinia is revealed to the user as an active folder. When the user drops an XSL-FO file into the folder, Actinia notices it, picks it up and sends it for formatting to one of servers in the grid, then stores the formatted document in the output folder. The output folder can be the same as, or different from, the input one. This approach works when the user **sends the document for processing**, for example, when a different player needs the formatted document, or when the document leaves the system in another medium (for example is printed and delivered in hard-copy form).

2.1.1. Story: Actinia in a Bank

A typical usage is a bank generating statements, bills, invoices, personalized mails etc. Different programs installed on many servers generate different kinds of documents, each with its own styling and each with its own data retrieved from the database. The documents are generated as XML, styled using application specific transforms into XSL-FO, then all documents are placed into the inbound folder of EnMasse Actinia. Actinia picks them up and places generated postscript files into the output folder. A separate program monitors the output folder and sends the final documents to a number of print devices according to labels embedded into the documents. The service to the Bank is that of a dedicated print room!

2.2. Toaster

Toaster monitors a network connection, accepts source (XSL-FO) styled documents and sends back formatted documents via the same connection to the user. Unlike the Actinia case, the client always receives the result of processing in an electronic form for local print generation. This is suitable when the user **requesting document processing** and is both the producer of XML sources and the consumer of their formatted output.

2.2.1. Story: DocBook Formatting On-line

A university server provides a formatting facility for student projects. Students submit documents marked up in DocBook XML via a web interface and get them back as printable PDF. The web server connects to the EnMasse server via the intranet, sends the source and receives the formatted document, and then forwards it to the students browser. This saves a

huge amount of time with each student configuring and learning about Docbook processing locally.

2.3. Fairy

Fairy is a SOAP server which accepts source documents (XSL-FO) and sends formatted documents back via the same connection to the requestor (a client application). It can be easily tied with any application which supports SOAP, because writing SOAP clients is an easy task.

2.3.1. Story: Microsoft Excel to Adobe PDF conversion

A stylesheet to convert Microsoft Excel's XML output into XSL is stored on the HTTP server. Users compose their Microsoft Excel spreadsheets, press the button "Xls2Fo" in a toolbar, and a VBA program converts their spreadsheet to XML, adds to it processing instruction specifying XSL stylesheet and, with help of Microsoft Office Web Services Toolkit, sends it for formatting to Fairy SOAP web service.

2.4. XSL Transformation

EnMasse, in Actinia, Toaster and Fairy configurations, can apply XSL transformation to input documents. EnMasse nodes recognize `xml-stylesheet` processing instruction with type "text/xml" or "text/xsl" and apply the associated stylesheets to the source document. It allows to distribute both transformation and processing among the grid nodes, and fully utilize the power of the formatting framework.

For example, an installation dedicated to the formatting of DocBook documents may provide access to DocBook XSL stylesheets stored on a local server; the nodes will load and apply the stylesheets, and then format the generated XSL FO into PDF or PostScript.

Chapter 3. Package Contents

The EnMasse distribution contains:

`doc/`

documentation in DocBook XML, PDF and HTML;

`lib/`

the programs' code; `enmasse.jar` is the XEP Engine and `Python/` is the Access Point;

`bin/`

EnMasse launch scripts;

`etc/`

sample configuration files; `JavaClient/` folder contains source code of a sample client for Fairy;

`setup.py`

Distutils' setup script.

The text files use Unix-style line separator. All XML files are encoded using UTF-8. Documentation is generated from DocBook XML source using DocBook XSL stylesheets. All documentation is prepared using RenderX XEP.

Some filenames in the distribution are in mixed case, sometimes called CamelCase, e.g. ThisIsMixedCase.xml. Please take care to use suitable unpacking or unzipping tools, such as InfoZIP or Winzip which correctly handle mixed case filenames. Depending on the flavour of your operating system, your tools, and security considerations, you may want to change access permissions and ownership of the files in the distribution, but it is important to retain the case sensitive filenames

Chapter 4. Installation

4.1. Required Software

EnMasse runs on any operating system that has TCP sockets, a Java Virtual Machine, and Python 2.2 or newer (the current stable version at the time of writing is 2.3.4). Since you use XEP, you have Java. The installer of EnMasse is based on Python's Distutils module which comes with Python distribution. I recommend that you read *Installing Python* from the book *Dive Into Python* by Mark Pilgrim if you don't yet have Python on your computer.

Installing EnMasse in Toaster mode also requires a Web server able to run CGI scripts.

4.2. Installation procedure

In order to install EnMasse, launch

```
python setup.py install
```

command from the command line. By default the package content will be installed in `/usr/local/RenderX/EnMasse` or `C:\Program Files\RenderX\EnMasse` directories, respectfully on Unix and Windows systems. You can change the default installation directory with `--prefix` option. For example:

```
python setup.py install --prefix=~/tmp
```

From now on the installed directory will be referred to as ${instDir}.

After installation, in order to use EnMasse you will need to configure it by creating working directories, log and configuration files. You can configure EnMasse either by hand or using a configuration script.

```
python setup.py configure
```

command creates all necessary working directories, log and configuration files. The complete list of working directories, as well as log and configuration files are described in the next sections. You can customize the configuration with `--mode`, `--spool`, `--logs` and `--conf` options. For example:

```
python setup.py configure --mode=toaster
                          --spool=~/tmp/EnMasse/spool
                          --logs=~/tmp/EnMasse/logs
                          --conf=/usr/local/RenderX/EnMasse/enmasse.conf
```

Note: An empty log file must be created during the initial configuration procedure. If the log file does not exist, it will **not** be created during the runtime.

4.3. Installation directories

EnMasse installation has three types of installed content: programs and configuration files, working directories and program log.

Working directories.

EnMasse needs one folder as an internal working directory; additionally, Actinia requires three folders for user files: `input` , `output` and `quarantine` . It monitors the `input` folder for new work, places the formatted result into the `output` folder, and keeps a copy of all files not yet formatted in the `quarantine` folder so that if the system fails, all files are preserved and can be reprocessed. A temporary working directory, `tmp` is also used.

On Unix, these folders are in `/var/spool/enmasse/`; suggested folder names are `inp/`, `out/`, `qua/`, and `tmp/`. EnMasse must be able to write to the directories. All users[1] who will be submitting files for formatting must be able to write to the input directory (and probably also to the output one so that they may delete the formatted files after retrieving them).

Program logs.

EnMasse writes detailed logs, to detect and resolve problems and tune performance. On Unix, `/var/log/enmasse/` may be used to store them. The user running *EnMasse* must have write permissions for the logs' directory.

[1] "users" in the Unix sense, that is, owners of processes; users do not have to access these folders directly.

Chapter 5. Running EnMasse Access Point and Engines

To run EnMasse, launch an **access point** on one of the servers and several **XEP engines**, usually on separate computers. **`${instDir}/bin/enmasse`** is a shell script that launches the access point; it issues the following command:

```
python ${instDir}/lib/Python/enmasse.py etc/enmasse.conf
```

where `enmasse.conf` is the configuration file; The configuration syntax is explained in the following section. On a platform which supports Bourne shell scripts, use the convenience features the script provides (execute **`bin/enmasse -help`** for usage instructions), otherwise just run the command above.

`${instDir}/bin/engine` launches an XEP engine; it is a call (to a Java program):

```
java com.renderx.xepx.cliser.Engine -DCONFIG=/path/to/xep.xml
```

(replace the path to `xep.xml` with the actual location of XEP configuration). Additional command-line switches are:

-port *n*

TCP port number for data communications (6570 by default), several engines may be run in separate Java Virtual Machines on the same computer if they use different ports, or change the default port value if you have a reason to do so;

-hcport *n*

HTTP port. The engine has a built-in web server, the server displays the current status of the engine, as well as allowing it to switch the engine off or suspend it. The HTTP server may be disabled by specifying hcport=-1.

-label *name*

The engine's name is displayed on the monitor's web page; it is convenient to assign a separate name to each of the engines so that you can easily see which one you've connected to.

Both the access point and the engines run embedded HTTP servers; the servers display the current state, to monitor activity and help performance tuning. By default, the HTTP ports are 6570 for Actinia, 6575 for Toaster, 6577 for Fairy, and 6580 for XEP.

5.1. Writing Configuration Files

To run EnMasse, you must configure it. A configuration file determines both how EnMasse interacts with the outside world and how it manages XEP engines and distributes the load.

While for many parameters the default values are satisfactory, some values are required to be set explicitly to describe your local environment (the network and the computer).

The configuration file is in the following XML format (in Relax NG).

```
config = actinia | toaster | fairy
actinia = element actinia {
 actinia-folders & settings
}

toaster = element toaster {
 toaster-folders  & settings
}

fairy = element fairy {
 fairy-folders & settings
}
settings = options & servers & cliser

actinia-folders = element folders {
 attribute input {string},
 attribute output {string},
 attribute quarantine {string},
 attribute temporary {string}
}

toaster-folders = element folders {
 attribute temporary {string}
}

fairy-folders = element folders {
 attribute temporary {string}
}

options = element option {
 attribute name {token},
 attribute value {string}
}*

servers = servers {
 element server {
  attribute host {token}?,
  attribute port {token}?
 }+
```

```
}

cliser = element cliser {
 attribute format {token}?,
 options
}
```

The *EnMasse* mode is set by the top-level element; it is `toaster` (network server), `actinia` (active folder) or `fairy` (SOAP server); the required elements are `folders` and `servers`.

Attributes of `folders` are `temporary`, `input`, `output`, and `quarantine` which specify paths to the temporary folder (*EnMasse* kernel needs it), and, only for *Actinia*, to the `input`, `output` and `quarantine` folders. Actinia looks for XSL-FO sources in the `input`folder and writes formatted results to `output`. It keeps a copy of each XSL-FO source in the `quarantine` folder until processing is finished. This allows the user to re-submit documents after a system failure (for example, due to a power or hardware problem). Both `input` and `output` attribute values can point to the same location: Actinia picks files ending in '.fo' by default. You can set a different input filter. See the list of options below.

Here's a sample `folders` section:

```
    <folders
        temporary="'C:/EnMasse/tmp'"/>
```

`servers` defines servers available to the access point. For each `server`, `host` is the server's host name or IP address ('localhost' by default), and `port` is the XEP engine's port (the default value is 6570). You can list the same server multiple times if you want it to load it more heavily. XEP engines are multi-threaded and handle concurrent sessions efficiently.

Here's a sample `servers` section:

```
    <servers>
        <server host="'localhost'" port="6570"/>
    </servers>
```

CLISER, RenderX XEP Client-Server protocol, is the underlying protocol layer; element `cliser` sets the required document format (`pdf` is the default value, `ps` (for PostScript), or `xep` may be used), and can contain CLISER options (see the documentation on XEP for the list of option names). Use the same names as are available for XEP and prepend core options with 'FRM:' and generator options with 'GEN:' (optionally followed by the format's name and a colon). For example, the following fragment:

```
<cliser format="pdf">
```

```
  <option name="FRM:VALIDATE" value="'true'"/>
  <option name="GEN:pdf:COMPRESS" value="'false'"/>
</cliser>
```

sets output format to PDF, enables validation and switches off compression.

5.2. Configuration Options and Tuning Performance

You can tune EnMasse' performance through a number of options. Default values are fine for most applications. By changing them you can build the exact configuration you want and fine-tune the load on the grid,, the throughput, and the response time. Here is the list of all the available options, with their data types and default values in parentheses.

5.2.1. Actinia Options

pickup-interval (seconds:1)

interval to check for new source documents;

pickup-delay (seconds:2)

delay since the last modification of a source document, Actinia needs it to avoid picking up documents while they are being written;

end-of-input (string:'stop')

when Actinia finds a file with this name in the input folder, it shuts down;

input-filter (regular expression: \.fo$)

regular expression for source names in the input directory;

5.2.2. Toaster Options

data-port (int:6575)

TCP port Toaster accepts data and return results on;

data-backlog (int:0)

backlog for data connections, default is no backlog.

5.2.3. Fairy Options

soap-port (int:6577)

TCP port Fairy SOAP server accepts SOAP requests and return results on.

data-backlog (int:0)

backlog for data connections, default is no backlog.

format-method (string:'format')

Remote method name called for formatting.

stop-method (string:'stop')

Remote method name called to stop Service.

accept-path (string: '/fairy')

Service alias, used in HTTP requests.

not-found-page (string)

HTML page, which is returned, if requested path was not *`accept-path`*

welcome-page (string)

HTML page, which is returned for 'GET' HTTP requests with *`accept-path`* resource URI.

Access-Control-Allow-Origin (string)

required for sending formatting requests to Fairy from a Web site via the Web Browser. This option specifies the value of *`Access-Control-Allow-Origin`* parameter in HTTP header returned by SOAP server (Fairy). A more detailed information is available at W3C Recommendation on Cross-Origin Resource Sharing.

use-https (boolean:False)

Enables HTTPS. Formatting requests over plain HTTP will be rejected. Requires specifying *`server-ssl-certfile`* and *`server-ssl-keyfile`*.

TLS-version (string:'TLSv1')

Specifies TLS version. Default version is TLSv1. With python 2.7.9 or later you can also use TLSv1_1 or TLSv1_2. Available values are: TLSv1, TLSv1_1, TLSv1_2. The older versions of SSL are deprecated and therefore aren't supported.

server-ssl-certfile (string)

Path to the server's certificate file in PEM format.

server-ssl-keyfile (string)

Path to the file with server's private key in PEM format.

mutual-authentication (boolean:False)

Enables mutual authentication (two-way TLS). All clients submitting requests must provide a valid certificate. Fairy server verifies those against the CA certificate (or certificate chain) specified in *`ca-certs-file`* file. Requests that failed to provide with one, will be rejected.

ca-certs-file (string)

Path to a file with CA certificate (or certificate chain) in PEM format.

max-request-queue-size (integer:0)

Controls how load balancing work. This option specifies maximum size of this queue. Default value is 0, which means that the queue is unlimited.

service-temporarily-unavailable-page (string)

Specifies a path to custom Error 503 page. This can be any document in plain text or HTML.

use-disk-cache (boolean:False)

Enables disk caching for XSL-FO files. This option should be only enabled for processing very large documents (larger than available memory), otherwise negative impact on performance may occur.

5.2.4. Common Options

agents-count (integer: number of servers)

number of agents to launch, default is the number of servers;

putback-interval (seconds: 1)

dead servers are brought back periodically; a separate thread tries to re-connect to them, and if it succeeds, *EnMasse* starts sending documents to them again;

socket-timeout (seconds: indefinite)

connections to servers time out after this interval, thus even if a server went down without properly closing its socket, EnMasse will notice its outage and temporarily unregister it;

log-path (string: None)

path to the log file; if omitted, EnMasse prints to the standard error stream by default

log-level (string:'errors')

logging level, one of `none`, `errors`, `all`;

Note: Remember that specifying `log-level` option to `all` negatively affects on performance. It is recommended to use `log-level` option set to `errors`, unless for troubleshooting purposes.

report-label (string:'EnMasse:Actinia' or 'EnMasse:Toaster')

default heading for http report (change it for each EnMasse instance if you have several ones)

http-port (integer:6590 for Actinia, 6595 for Toaster, and 6597 for Fairy)

http port the logger listens on.

Note: When running multiple EnMasse's on the same machine, different ports need to be used for **both data and http**.

Chapter 6. Using EnMasse in Toaster mode

Toaster is one of the parts of EnMasse which requires that you write a program to use it. Since Toaster accepts requests over a network TCP socket, and implements a simple protocol, you must implement the protocol in your language of choice and embed it in your client-side application, such as a web form, or an authoring tool. An example of protocol CGI script calling toaster to format a document submitted via a WWW page is provided in `${instDir}/lib/Python/wet.cgi`, and a sample WWW page is in `${instDir}/etc/toaster.html`. Additionally, examples of Java Server Pages and ASP.NET are included into the distribution.

Before using EnMasse in Toaster mode, it must be configured accordingly. See Installation procedure for further details.

Setting up Toaster with a simple Web Server

The standard distribution of EnMasse contains a sample configuration for a simple Web-service setup.

1. Copy the file `EnMasse/lib/Python/wet.cgi` to the Web server's working directory.
2. Copy `Enmasse/etc/toaster.***.example` to the Web server's working directory.

   Note: Note. The format of example page depends on the specific Web server.
3. If necessary, change path to Python interpreter by editing `wet.cgi` and changing the path in the topmost line.
4. If the Toaster service is supposed to run on another machine, also specify its address and port in the line:

   ```
   TOASTER = ...
   ```
5. Start the web server and test its working by formatting any FO document.

6.1. Connecting to Toaster

The protocol involves one request and one response. The client sends the request, in the form

```
RECEIVE data-size systemId
```

followed by a zero byte ('\0' in C), and then by the data of `data-size` bytes in length itself. Toaster transforms the document into PDF or PostScript and returns it: it sends

```
RECEIVE data-size format
```

followed by a zero byte and by the formatted document.

If EnMasse cannot format the document, it sends

```
ERROR message-size None
```

followed by a zero byte and then by the error message. The message contains XEP's diagnostics and helps identify the problem.

To shutdown Toaster, send message `STOP` to the data port.

Chapter 7. Using EnMasse in Fairy mode

Fairy also requires you to write a program to use it. Since Fairy accepts SOAP requests you can use any SOAP toolkit to access it.

Before using EnMasse in Fairy mode, it must be configured accordingly. See Installation procedure for further details.

Fairy, as SOAP service, provides two methods: to format and to stop. If not otherwise specified in configuration methods names are respectively `format` and `stop`.

`format`

Method takes two arguments: `systemId` and `xml`. `systemId` is the document's system identifier. `xml` is XSL-FO document, or XML with embedded stylesheet (see XSL Transformation). See section Supported encodings

`stop`

The call of this method stops Fairy as soon as it becomes free. Any following requests will not be served.

Fairy provides WSDL document, describing the service. For example, if Fairy is running at host `yourhost` with default configuration, you can get WSDL document, with `GET /fairy?WSDL` HTTP request to `yourhost:6577` (say, just by typing `http://yourhost:6577/fairy?WSDL` in browser's address field).

Fairy, as SOAP service, returns one of the following status codes :

200 OK

The request has succeeded. The response body contains formatted document.

503 Service Unavailable

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.

7.1. Transport Layer Security

Fairy supports TLS (Transport Layer Security) that allows sending sensitive documents securely via unsecure networking transport. In this case, the clients are ensured they are connecting to an authentic server.

Fairy also supports two-way TLS, limiting access to authorized clients only. In this case, it stores public keys of all authorized clients and verifies the keys on each request.

EnMasse distribution contains a sample of Fairy client application. One may find it in `etc\JavaClient\` within the EnMasse installation directory. The application shows how to use one-way and two-way TLS with Fairy.

7.2. Load Balancing

Fairy also acts as a load balancer. Upon arrival, all new incoming formatting requests are internally queued. Once an Agent is available, it starts processing the first queued formatting request.

Queue length may be configured. If the limit is reached, e.g. the queue is full, and a new request arrives, it is rejected with Error 503 Service Temporary Unavailable and warning is logged. Since the queue keeps open connection sockets, the queue length is actually limited by the amount of free sockets in OS' TCP stack. Normally, this limit should not be reached, and the fact that it is being actually reached usually means that the configuration is insufficient for the purpose of effective load balancing.

7.3. Supported Encodings

Fairy will accept SOAP request without type specification for method's arguments. In this case `format` method's first argument will be interpreted as 'string' (and will be used without any conversion) and the second argument as 'base64Binary', but for greater compatibility it suggests the following types for arguments:

'base64' or 'base64Binary'

data is base64-encoded.

'arrayType'

data is represented as array of bytes.

'string'

data is represented as is (and will be used without any conversion).

Here are examples of SOAP requests:

```
...
<format>
 <systemId>SYSTEMID</systemId>
 <xml>XML_DATABASE64ENCODED</xml>
</format>
...
```

```
...
<format>
 <systemId xsi:type="xsd:string">SystemId</systemId>
 <xml xsi:type="xsd:base64Binary">XML_DATA_BASE64_ENCODED</xml>
</format>
...
```

Chapter 8. WinMasse

WinMasse is the same EnMassse application besides it runs as a windows service. Before installing WinMasse as Windows service, you must have installed Python Win32 Extensions. Winmasse is installed and uninstalled with

```
winmasse.py install
```

and

```
winmasse.py remove
```

commands, respectively.

Note: Just after the installation, WinMasse registers itself as a **"manually starting" service**. The reason for that is that after the installation, the administrator must choose an appropriate configuration according to the business needs and configure EnMasse by creating a `winmasse.conf` file. Once a good, valid configuration is created, the system administrator may then change WinMasse to start "automatically".

WinMasse searches the configuration file named `winmasse.conf` in the same directory as the `winmasse.py`. The configuration file has the same structure as the EnMasses's.

Index