RenderX InGrid

User Guide

RenderX InGrid: User Guide

RenderX

228 Hamilton Ave.
Palo Alto, CA 94301
USA

Telephone: 1 (650) 328-8000
Fax: 1 (650) 328-8008
Website: https://www.renderx.com/
Email: <support@renderx.com>

Table of Contents

Chapter 1. Preface

1.1. Abstract

InGrid is a server-based product for shared document formatting. It accepts XSL-FO documents over the network and formats them by distributing the formatting tasks among multiple formatters that may run locally or on other computers in the network. Regular users publish customized documents in high volumes and varied formats. The server is accessed via straightforward controls. The users may customize server operation for their workflow.

InGrid is a direct replacement of *EnMasse*. It is completely backward compatible with *EnMasse*, i.e. it accepts the same configuration files and supports the same working modes while providing significantly better performance.

Additionally, *InGrid* provides with REST API.

1.2. A Bird's Eye View

Computation-heavy processes, like XML transformation or rendering XSL-FO to PDF, require significant resources and take more time to complete if ran on a local machine.

The conservative approach to this problem is using multiple-core and multiple-CPU configurations with server-grade amounts of memory. This introduces the need of effective and scalable task management tools to distribute the rendering tasks between the cores, detect and retry on failures, balance the load, unify the logging, compute statistics, and so on.

While multiple-core systems may be sufficient for a certain amount of rendered documents per time unit, sooner or later, any upgrade bumps into physical limits of the architecture. A really scalable solution would require the synergy of rendering engines running on multiple computers across the network. Using multiple computers, the rendering cores can achieve a data throughput at the current level of technology permits; and the price for stock single-CPU PCs is low enough to have sufficient resources to meet any business needs.

This approach also helps reliability and fail-tolerance of the grid; when the rendering engines are constantly online, the load is evenly distributed; in case when a node is down due to hardware failure or being upgraded, the entire system still processes all requests, preserves data, however, at the cost of the degraded performance. This is a well-known but not an easy task; and to really use the synergy power of the grid computing as it is known, you must resolve this issue.

RenderX XEP is a rendering engine that creates elegantly formatted print-ready template-based 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. While XEP is based on an optimized and highly-performant code, the business needs a way to **scale up** the performance to be able to handle a constantly growing load.

InGrid solves this problem for you. For an end user or an application programmer, it acts as a single access point. Through a shared folder, a Web form or a network connection (locally on the LAN, or across the Internet), *InGrid* accepts document processing requests and sends them to one of the XEP formatting engines running locally or on several computers in the local network, then delivers formatted documents back to the user. *InGrid* also monitors the performance of each formatting engines, notices when they go down and are restored, and re-adjusts the distribution of the requests according to the workload. When a server fails in the midst of processing a request, *InGrid* re-submits the request to a different server; thus the only impact is a slightly increased response time for that particular document. This provides the reliability of the entire service needed in today's world.

InGrid is both opaque and transparent. On one hand, it provides with a **single point** abstraction so that there is no need to worry about the number of engines running, or about their load; *InGrid* 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 nodes in the grid, identify problems and take an appropriate action. An *InGrid* access point takes little memory overhead and processing time, it can be deployed on a busy Intranet server, or even on a consumer-grade workstation and, as long as the processing engines run on separate machines, it does not affect the performance or throughput of the grid.

For debugging purposes, audit, and performance tuning, *InGrid* provides a logging facility. One can adjust the verbosity of the logging, or completely turn it off. The log files are easy to read and to process by programs.

InGrid runs on a wide range of hardware and operating systems, easy to install and requires a little maintenance.

Chapter 2. The User's View

Internally, *InGrid* distributes formatting jobs, records their activity to the log, and monitors the grid performance. Whatever the system around it is doing, the role of its core remains constant. For the user, *InGrid* 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*)**. Additional proxying interface ***Fork*** allows starting multiple interfaces simultaneously. It is especially useful for running multiple *InGrid* interfaces at the same time as Windows service.

2.1. Actinia

Actinia is basically a service of the Active Folder within the local file system. Once a source document appears there (e.g., the user drops a file using a normal file manager), *Actinia* notices it, picks it up, and sends it for formatting to one of servers in the grid. As soon as the formatting completes, *Actinia* 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, printed and delivered in a 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 by using application-specific transforms into XSL-FO, then all documents are placed into the inbound folder of *InGrid*'s *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 is a network-activated service. It monitors a network connection, accepts source styled documents (XSL-FO), and sends back the formatted documents to the user via the same connection. 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 Online

A university server provides with a formatting facility for student projects. Students submit their documents marked up in DocBook XML via the Web interface and get them back as printable PDF. The Web server connects to the *InGrid* server via the intranet, sends the source, receives the formatted document, and then forwards it to the students' browser. This saves a considerable amount of time and effort instead of each student configuring and learning about DocBook processing locally.

2.3. Fairy

Fairy is a SOAP/REST 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 and REST 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-FO 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. Fork

Fork is basically an integrator of an arbitrary number of services listed above (*Actinia*, *Toaster*, and *Fairy*). Its configuration file contains a list of config files, each of which, in turn, configure corresponding *InGrid* instances for each list entry. *Fork* then starts each listed instance. Additionally, it provides a Web interface for monitoring status of all started instances. Using *Fork* is an alternative to manually starting multiple *InGrid* instances from the command console with their own config files passed as parameters.

2.5. XSL Transformation

InGrid, in *Actinia*, *Toaster*, and *Fairy* configurations, can apply XSL transformation to input documents. *InGrid* nodes recognize `xml-stylesheet` processing instruction with type of `"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. Installation

3.1. Required Software

InGrid runs on any operating system that has TCP sockets, and Java Virtual Machine. Since XEP also uses Java, and *InGrid* is usually installed with one of its worker engines, you may already have Java installed. The installer of *InGrid* is Java-based; it also includes `setup.bat` for Windows users convenience.

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

3.2. Installation procedure

In order to install *InGrid*, launch

```
setup.jar
```

or on Windows run:

```
setup.bat
```

Follow setup Wizard and complete the installation.

Note: Make sure that the user account who performs the installation has appropriate permissions, sufficient for modifying the system.

After the installation, *InGrid* requires proper configuring in order to run. This includes creation of a configuration file. *Actinia* mode also requires working directories to be created and properly configured (user permissions and access mode). See example conf files in `etc/`, they are in XML format and contain all options commented with detailed descriptions. Uncomment required options and adjust values according to your requirements. *InGrid* installed as Windows service uses `ingrid.conf` from the root installation directory. Installing *InGrid* as Windows service will create basic `ingrid.conf` based on `fairy.conf.example` located at `etc/` directory for testing purposes, adjust it to your requirements.

3.3. Installation directories

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

Working directories

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/ingrid/`; suggested folder names are `inp/`, `out/`, `qua/`, and `tmp/`. *InGrid* requires **write permission** to the directories. All users [1] who will be submitting files for formatting must have write permission to the `input` directory (and probably also to the `output` one so that they may delete the formatted files after retrieving them).

Program logs

InGrid writes detailed logs, to detect and resolve problems and tune performance. On Unix, `/var/log/ingrid/` may be used to store them. The user running *InGrid* must have a write permission on the logs directory.

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

Chapter 4. Running *InGrid* Access Point and Engines

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

```
java -cp ${instDir}/lib/ingrid.jar com.renderx.ingrid.XepMultiplexer etc/ingrid.conf
```

where `ingrid.conf` is the configuration file; The configuration syntax is explained in Section 4.1, “Writing Configuration Files”. On a platform which supports Bourne shell scripts, use the convenience features the script provides (execute **`bin/ingrid -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).

The `CLASSPATH` system variable should include paths to:

- `ingrid.jar`, the main executable;
- All necessary libraries of XEP engine, which *InGrid* uses for actual formatting;

Additional command-line switches are:

-port *n*

TCP port 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 unique 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.

4.1. Writing Configuration Files

To run *InGrid*, you must configure it. A configuration file determines both how *InGrid* 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).

InGrid is packaged with a set of initial configuration files called `actinia.conf.example`, `toaster.conf.example`, `fairy.conf.example`, and `fork.conf.example`. Copying and editing example configs from `${instDir}/etc/` is the fastest and the easiest way to start.

The configuration file is in the following XML format (in Relax NG). The complete specification of *InGrid* configuration file follows:

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

toaster = element toaster {
 toaster-folders  & settings
}

fairy = element fairy {
 fairy-folders & settings
}

fork = element fork {
 fork-tines & element option {
  attribute name {"http-port"},
  attribute value {string}
 }?
}

fork-tines = element tine {
  attribute conf {string}
 }+

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 *InGrid* mode is set by the top-level element; it is either `toaster` (network server), `actinia` (active folder), `fairy` (SOAP server) or `fork`(proxying mode). All modes except `fork` require `folders` and `servers` elements.

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

Here's a sample `folders` section:

```
<folders
    temporary="'C:/InGrid/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 turns off compression.

4.2. Configuration Options and Tuning Performance

InGrid allows tuning its performance through a number of options. The default configuration 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 available options, with their data types and default values in parentheses:

4.2.1. Actinia Options

pickup-interval (seconds:1)

interval to check for new source documents;

pickup-delay (seconds:2)

delay after 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|\.xml|\.xep)$)

regular expression for file names treated as source documents in the input directory;

4.2.2. Toaster Options

data-port (int:6575)

TCP port which *Toaster* accepts data and return results on;

data-backlog (int:0)

backlog for data connections, default is no backlog.

4.2.3. Fairy Options

soap-port (int:6577)

TCP port which *Fairy* SOAP server accepts 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 to submit a formatting request.

stop-method (string:'stop')

Remote method name called to stop Service.

accept-path (string:'/fairy')

Service alias, used in HTTP requests.

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. If enabled, formatting requests over plain HTTP will be rejected. Requires specifying `server-ssl-certfile` and `server-ssl-keyfile`.

TLS-version (string:'TLSv1_2')

Deprecated. Specifies TLS version. Default version is TLSv1.2. Can be used to downgrade TLS version to TLSv1 or TLSv1_1, which is strongly discouraged for security considerations.

server-ssl-certfile (string)

Path to the file with 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. A non-zero value makes *InGrid* reject the request until the working queue frees an allocation slot. See Section 7.5, “Load Balancing ” for details. Default value is `0`, which means that the queue is unlimited.

service-temporarily-unavailable-page (string)

Specifies a path to a file containing a 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 the available memory), otherwise negative impact on performance may occur.

4.2.4. Fork Options

Fork accepts one or more `tine` elements (one per *InGrid* instance to run) and an optional `option` element with `http-port` attribute specifying port number for *Fork* Web interface to reside at. Each `tine` element should have a `conf` attribute which string value should contain a pathname to a valid *InGrid* config.

http-port (integer:6600)

HTTP port for connecting to the Web interface.

4.2.5. Common Options

agents-count (integer: number of servers)

number of agents to launch, default is equal to 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, *InGrid* starts sending documents to them again;

socket-timeout (seconds: indefinite)

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

log-path (string: None)

path to the log file; if omitted, *InGrid* prints to the standard error stream (`stderr`) 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:'InGrid:Actinia' or 'InGrid:Toaster')

default heading for HTTP report (change it for each *InGrid* 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 instances of *InGrid* on the same machine, different ports need to be used for **both data and HTTP**.

Chapter 5. Using InGrid in Actinia mode

Using *InGrid* in *Actinia* mode is no more complex than copying a file within the file system itself. In fact, this mode is designed for simplicity and it does not require any complex management or maintenance. The only required configuration option is the full path to the folder to monitor. Another caveat is that designing your system around *Actinia* requires the client software to properly maintain the filesystem permissions and locks. Once the file is copied to the Active Folder, it must be released from any locks so that *Actinia* could process it. This also includes scenarios involving network-accessible folders (network drives, network shares) mounted to the local file system, and the network protocol should be able to properly release the write locks and support access permissions.

Chapter 6. Using InGrid in Toaster mode

Toaster is one of the parts of *InGrid* which requires to be integrated into the customer's business process by writing a client program. 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.

InGrid distribution includes an example client application. It is written as a CGI script and Python. It contains a sample Web page `${instDir}/etc/toaster.html` which contains an HTTP form. A user attaches an XSL-FO document and submits the form to a `${instDir}/etc/wet.cgi` script which, in turn, calls the *Toaster* to format the document. Additionally, examples of Java Server Pages and ASP.NET are included into the distribution.

Before using *InGrid* in *Toaster* mode, it must be configured accordingly. See Chapter 3, *Installation* for further details.

Setting up *Toaster* with a simple Web Server

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

1. Copy the file `InGrid/etc/wet.cgi` to the Web server's working directory.
2. Copy `InGrid/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 whether it is 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 as follows:

```
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 *InGrid* is unable to 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 InGrid in Fairy mode

In *Fairy* mode, *InGrid* works as a network server. It accepts SOAP or REST requests, takes source documents along with their resources, and performs formatting via one or more instances of XEP formatters. In *Fairy* mode, *InGrid* also performs load balancing between the underlying formatters.

InGrid in *Fairy* mode is supposed to be accessed from within the client's custom software which is a part of your company's document processing environment. Simply speaking, the user (you) are required to write a relevant piece of Client program and configure it to access the *Fairy*. Since *Fairy* accepts SOAP requests, there are very few limitations on the platforms or programming languages to use; virtually any SOAP toolkit would fit, depending on your needs.

Before using *InGrid* in *Fairy* mode, it must be configured accordingly. See Chapter 3, *Installation* for further details.

7.1. SOAP API

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

`format`

Method takes two arguments: `systemId` and `xml`. `systemId` is the document's system identifier. `xml` is XSL-FO document, or XML with an embedded stylesheet (see Section 2.5, “XSL Transformation”). Also see Section 7.6, “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 with a WSDL document describing the service. For example, if *Fairy* is running at host `yourhost` with the default configuration, you can get the WSDL document by sumbitting `GET /fairy?WSDL` HTTP request to `yourhost:6577` (e.g., just by typing `http://yourhost:6577/fairy?WSDL` in browser's address field).

Fairy, as a 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.

See Section 7.3, “API Endpoint list ” for further details on commands and their arguments.

7.2. REST API

REST API is de-facto the standard approach for modern network APIs. In recent years, SOAP has been rarely used for new projects and is often replaced with the REST API because of the simplicity of the latter. REST API works in a pretty much the same way as SOAP over the same HTTP connection, but it does not require XML wrapping and Base64 encoding for both request and response. Avoiding double encoding and wrapping results in smaller network traffic and lower CPU usage and therefore provides with better overall performance (on small documents 7-10%). Writing a client code for REST API and supporting it appears simpler and faster because it consists of sending HTTP request with a raw document data and receiving a raw formatting result without requiring any additional wrapping or encoding.

A simple HTML form example for submitting a document for formatting via the Web browser:

```
<html><body><div>
    <form action="http://localhost:6577/format" method="POST"
             enctype="multipart/form-data">
        <label>Document to format:</label>
        <input type="file" name="upload_file[]"/>
        <label>SystemId:</label>
        <input type="url" name="systemId[]" value=""/>
        <input type="submit" value="Submit"/>
    </form>
</div></body></html>
```

This example can be saved as an `.html` file and opened in a Web browser to submit documents (assuming that *InGrid* is running locally), or the form part can be integrated in any other Web site. A bit more user-friendly version of this example is included in the *InGrid* distribution and is accessible at `/format` directory.

Note: If the API is used by the JavaScript code (Ajax) instead of submitting a form, *`Access-Control-Allow-Origin`* option must be set to `*`.

7.3. API Endpoint list

The following API endpoints are available:

Table 7.1. API Endpoints

HTTP method	URI path	Request Content Type	Description
POST	**/fairy**	text/xml	Accepts SOAP formatting requests.

HTTP method	URI path	Request Content Type	Description
GET	**/fairy?WSDL**		Retrieves WSDL for *Fairy* SOAP service.
POST	**/format**	multipart/form-data	The Request must contain a `systemId[]` and `upload_file[]` entries. Returns a formatted document from `upload_file[]` entry with all relative resources resolved relatively URL from the `systemId[]` entry.
GET	**/format**		Retrieves a Web page with a form for submitting documents to process.
POST	**/formatFO**	text/xml	Accepts an XSL-FO file for formatting. If the FO file contains relatively-referenced resources, the URL must contain a query with a `systemId` parameter, having the value of an URL-encoded `systemId` (base URL for resolving relative URLs in the document. For example: `/formatFO?systemId=file%3A%5CC%3A%5CProgram+Files%5CRenderX%5CXEP%5Cexamples%5Cbasic%5Cbgimage.fo`
POST	**/formatZIP**	application/zip	Accepts a ZIP binary file containing a source FO or XML+XSL or XEPOUT file. *InGrid* will automatically detect the file to start from. The ZIP archive may have any name. The archive can also contain bundled resources such as images. If the `systemId` is not specified via the URL query, the `systemId` will be set to the ZIP file contents; therefore allowing resource bundling.
GET	**/**	text/xml	Retrieves a page containing a brief description of *Fairy* service and links to a page containing a document submitting form and page with API description.

If the formatting request fails, the server will return `500 Content-Type: "text/plain"` with the detailed description of the failure.

7.4. Transport Layer Security

Fairy supports TLS (Transport Layer Security) that allows sending sensitive documents securely via insecure 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.

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

7.5. 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.

The queue length may be configured. If the limit is reached, e.g. the queue is full, and a new request arrives, it gets rejected with Error 503 Service Temporary Unavailable status code and a 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 hardware configuration is insufficient for the purpose of effective load balancing.

7.6. Supported Encodings

Fairy will accept SOAP request without the 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. Using InGrid in Fork mode

Fork is basically a service integrator (or proxying) mode, and it serves to aid running and maintenance of multiple instances of other *InGrid* modes. Other modes are specified via `tine` elements, where each tine element corresponds to one *InGrid* instance. Each `tine` element has its own `conf` attribute pointing to a valid *InGrid* configuration file.

Note: Since *Fork* mode involves many instances of *InGrid*, log reports may grow rapidly, thus making troubleshooting complicated. To simplify initial configuring and testing as well as troubleshooting, we recommend configuring and testing each instance **separately first**. Once all needed instances work flawlessly **alone**, go ahead and configure *Fork* to run them together.

Chapter 9. Running as Windows service

InGrid can be installed to run as a Windows service.

If run on a Windows machine, the *InGrid* installer registers it automatically.

Note: Just after the installation, *InGrid* 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 *InGrid* by creating a `ingrid.conf` file. Once a good, valid configuration is created, the system administrator may then change *InGrid* to start "automatically".

InGrid searches the configuration file named `ingrid.conf` in the root installation directory.

Alternatively, *InGrid* can be run using a batch script `ingrid.bat`:

```
java -classpath "c:\path\to\InGrid\lib\ingrid.jar" \
    com.renderx.ingrid.XepMultiplexer c:\path\to\InGrid\ingrid.conf
```

See more details in Chapter 4, *Running InGrid Access Point and Engines*.

To uninstall *InGrid* installed as Windows service use "Add or remove programs", "Apps & features" or "Programs and Features".

Chapter 10. Frequently Asked Questions

10.1. The simplest way to set up two-way TLS

Setting up set up two-way TLS

The following steps show one possible way to set up two-way TLS on a single machine.

1. Creating certificates

   use `openssl` to generate certificates, if `openssl` isn't in the path then use full path to `openssl.exe`.

   Use OpenSSL ≥ 1.1.1 Binaries for Windows: download link.

   The variable `subjectAltName` required for accessing *Fairy* via Chrome; Chrome will complain if TLS version <1.2 or certificate doesn't contain `subjectAltName`.

   ```
   cd "C:\Program Files (x86)\RenderX\InGrid"
   # create selfsigned certificate and private key for server
   openssl req -new -x509 -days 365 -nodes -out serverCert.pem \
     -keyout serverKey.pem -subj "/CN=localhost" \
     -addext "subjectAltName=DNS:localhost,IP:127.0.0.1" \
   # create selfsigned certificate and private key, don't care
   # about any info for client so use -subj to avoid questions
   openssl req -new -x509 -days 365 -nodes -out clientCert.pem \
     -keyout clientKey.pem -subj "/CN=FairyClient"
   ```

   Note: For different machines the procedure is the same, but files need to be copied and change `'localhost'` to corresponding hosts names.

2. Configure *InGrid* to use created certificates.

   InGrid config should contain the following:

   ```
   <option name="use-https" value="True"/>
   <option name="server-ssl-certfile"
     value="C:\Program Files (x86)\RenderX\InGrid\serverCert.pem"/>
   <option name="server-ssl-keyfile"
     value="C:\Program Files (x86)\RenderX\InGrid\serverKey.pem"/>

   <option name="mutual-authentication" value="True"/>
   <option name="ca-certs-file"
     value="C:\Program Files (x86)\RenderX\InGrid\clientCert.pem"/>
   ```

3. Start *InGrid*

Start *InGrid* as Windows service by invoking the following command:

```
net start ingrid
```

InGrid should be started as Windows service (it will use `C:\Program Files (x86)\RenderX\InGrid\ingrid.conf` as its config)

4. Use this Sample Client

   This sample Client can be build to Client.jar by provided `build.bat`

   Format single document from examples using this sample Client:

```
java -jar Client.jar -url https://localhost:6577/fairy \
  -trustedCertsFile "C:\Program Files (x86)\RenderX\InGrid\serverCert.pem" \
  -clientCert "C:\Program Files (x86)\RenderX\InGrid\clientCert.pem"
  -clientPrivateKey "C:\Program Files (x86)\RenderX\InGrid\clientKey.pem" \
  "C:\Program Files\RenderX\XEP\examples\basic\linebreak.fo"
```

   Do not forget to disable detailed logging after verifying that everything works and before doing performance tests

   Format the same document from examples 1000 times and do not save formatted pdfs to disk(useful for performance testing):

```
java -jar Client.jar -url https://localhost:6577/fairy \
  -trustedCertsFile "C:\Program Files (x86)\RenderX\InGrid\serverCert.pem" \
  -clientCert "C:\Program Files (x86)\RenderX\InGrid\clientCert.pem" \
  -clientPrivateKey "C:\Program Files (x86)\RenderX\InGrid\clientKey.pem" \
  -amount 1000 -writeFormattedDocsOnDisk false
  "C:\Program Files\RenderX\XEP\examples\basic\linebreak.fo"
```

Example for generating certificates

```
openssl req -new -x509 -days 365 -nodes -out clientCert.pem \
  -keyout clientKey.pem
```

req - PKCS#10 certificate request and certificate generating utility.

Read full documentation at OpenSSL.org.

-x509 option outputs a self signed certificate instead of a certificate request. Otherwise the certificate request will be generated.

Note: ATTENTION!

When creating a certificate or submitting a certificate request you will be asked for several parameters. Pay attention to `CN` field. By default you'll be asked for "Common Name (e.g. server FQDN or YOUR name)", and your answer should correspond to the server's name you're going to use this certificate on. Certificate won't be valid otherwise.

Certificate requests can be signed using x509 utility, download link

Example:

Sign a certificate request using the CA certificate above and add user certificate extensions:

```
openssl x509 -req -in req.pem -CA cacert.pem -CAkey key.pem \
  -CAcreateserial -out signedCert.pem
```

Index