Installation Guide of Mesh Framework

(NON-OSGI)

Table of Content

Contents

[1)Prerequisite 2](#_Toc445830252)

[1.1) Software Requirement: 2](#_Toc445830253)

[1.1.1) Install JDK/JRE: 2](#_Toc445830254)

[1.1.2) Installing JBoss Fuse IDE 2](#_Toc445830255)

[1.1.3) Installing Apache Maven: 2](#_Toc445830256)

[1.1.4) MySQL Installation: 2](#_Toc445830257)

[1.1.5) SOAP UI Installation: 2](#_Toc445830258)

[1.1.6) PrintNode Server Setup: 3](#_Toc445830259)

[2) Setup the Code to Eclipse/Fuse IDE: 3](#_Toc445830260)

[2.1) Get the code from Git repository: 3](#_Toc445830261)

[2.2) Project Overview: 3](#_Toc445830262)

[2.3.1) Framework: 3](#_Toc445830263)

[2.3.2) Features: 5](#_Toc445830264)

[2.3) Database setup for Application: 7](#_Toc445830265)

[2.3.1) Configuration Level database setup: 7](#_Toc445830266)

[2.3.2) Application Level database setup: 7](#_Toc445830267)

[2.4) Importing codes in Eclipse/Fuse IDE: 7](#_Toc445830268)

[3) How to test framework: 8](#_Toc445830269)

[3.1) Run all test cases at once: 8](#_Toc445830270)

[3.2) Run Test suites from eclipse/Fuse IDE: 8](#_Toc445830271)

[4) How to run framework with feature: 9](#_Toc445830272)

[4.1) Framework with LabelService feature 9](#_Toc445830273)

[4.1.1) Setup framework with feature (roi-labelService and roi-printService-printnode): 9](#_Toc445830274)

[4.1.2) Testing framework with feature (roi-labelService and roi-printService-printnode): 10](#_Toc445830275)

# 1)Prerequisite

## ***1.1) Software Requirement***:

### 1.1.1) Install JDK/JRE:

It is required to have java 7 JDK/JRE already installed into the system.

### 1.1.2) Installing JBoss Fuse IDE

1. Jboss Fuse IDE using Eclipse Indigo(v3.7):

* Red Hat JBoss Fuse IDE works with Eclipse Indigo (v3.7) and Maven m2e.

Note: You cannot install Red Hat JBoss Fuse IDE into a Spring STS Eclipse distribution because Red Hat JBoss Fuse IDE uses m2e, while Spring uses m2eclipse.

b) Direct Jboss fuse IDE archive:

* + - You can download the Red Hat JBoss Fuse IDE archive from [www.jboss.org](http://www.jboss.org/jbossfuse).
    - Red Hat JBoss Fuse IDE is packaged as a .zip file. Using a suitable archive tool, such as Zip, unpack Red Hat JBoss Fuse IDE into a directory to which you have full access.

### 1.1.3) Installing Apache Maven:

* You can download latest Apache Maven required to build projects from <https://maven.apache.org/download.cgi> .
* Using a suitable archive tool, such as Zip, unpack Apache Maven into a directory to which you have full access.
* Setup the apache maven home directory path into platform you are using, inorder to run maven based projects from command line tool.

Note: code is tested with apache maven 3.3.9 version

### 1.1.4) MySQL Installation:

* Install MySQL into the system of version 5.5. For windows machine you can download from <https://dev.mysql.com/downloads/windows/installer/5.5.html> .
* Data base schema required for application to run successfully is available in GitHub repository “roimeshrepo” under “docs/database” folder.
* Execute a data definition language script called “roi-schema.sql” available to set basic schema. Schema has table for storing configuration and test tables required by the junit test cases.
* Execute data manipulation language script called “roi-datainsert.sql” script to insert SQL statements require by the application.
* For POC and testing we have used tenant as “gap” and site as “site1”.

### 1.1.5) SOAP UI Installation:

We are using soap UI for testing cxf end point.

* Download the *platform supported binary zip (no JRE)* from [http://www.soapui.org/](https://www.soapui.org/)
* Unzip it into a preferable directory.
* Make sure that you have proper permissions on the unzipped soapUI folder.

### 1.1.6) PrintNode Server Setup:

PrintNode server is require for printing label. Create an account if you don’t have. Steps to get authentication token is given below.

* Signup from the url given below “<https://app.printnode.com/account/login>”.
* On login, click on the “API” tab to generate the api authentication key. Click on “Make New Api Key” tab to get new key.
* Enter the description and click on save.
* If already a user, you can login inside, click on the “API” tab to get the api authentication key. Click on “Make New Api Key” tab to get new key if you don’t have else you can copy from the list.

# 2) Setup the Code to Eclipse/Fuse IDE:

## 2.1) Get the code from Git repository:

Clone the Git repository containing codes into local system.

* create a folder in local system using suitable name, let say <gitFolder>.
* From command line go till the folder <gitFolder>.
* clone the codes to <gitFolder> folder using git clone command

Note:

* In case of Linux system, we need to install git. Open terminal and give command “sudo apt-get install git”.
* In case of windows machine. You need to download and install git bash and git GUI tool.

## 2.2) Project Overview:

This topic will cover the overview of the projects. We have 4 core project for framework. Other projects are the features developed over framework.

### 2.3.1) Framework:

***a)******“roi-framework” project:***

* We have written a pure java based core framework that has a service class for adding, getting, updating, enabling or disabling feature/service, deleting the configuration in database and cache.
* It supports eventing, dynastore, permastore, policy and feature Implementation configuration and can perform above specified operation using its respective service class.

***b) “mesh-core” project:***

* we have written a camel base core framework. All camel related operations are done here.
* It contains a “baseRoute.xml” which is an entry point for our application. Execution route, Implementation Route of feature are imported in “baseRoute.xml”.
* This project is responsible for initializing Mesh Header with request context object, feature group, feature, service, hazelcast transactional context object etc. Also responsible for generating unique id for each request.
* It is also used to route from base route to execution route based on featuregroup and feature name. Route from execution route to Implementation route by loading specific feature, checking if service is enable for the feature service then get the implementation route associated with it.
* It is used to call camel notifier at the end of the camel context which is responsible for dispatching component, service, system events configured for that feature and service.

***c) “features-installer” project(non-osgi):***

We have written a feature installer for non-osgi environment whose responsibility is to load all the features configuration (configuration like event configuration, feature Implementation configuration etc.) in database and cache before starting the application. This is done as follows:

* It searches for “featureMetaInfo.xml file” in all the jar available in its class path. If found, see the resource defined at feature level (resource like configurable xml for event, permastore, dynastore, featureImpl etc.). Load, parse and add its configuration into database as well as in data grid(hazelcast).
* Once all the configuration is loaded, it will load, parse the “baseRoute.xml” file available in “mesh-core” project and start the camel context defined in it.

***d) “feature-extender” project(osgi):***

We have written a feature extender for osgi environment whose responsibility is to load all the features configuration (configuration like event configuration, feature Implementation configuration etc.) in database and cache when a feature bundle is install and remove the configuration if feature bundle is removed.

* The project is provided with Activator class which has start and stop method implementation. On start the activator starts bundle tracker which listen and trigger an event when a bundle is get installed/start/stop/uninstalled. Also it has a custom service tacker class which maintain a registry of new service added or removed. On stop, it cleans the service registry and remove all registered services.
* Whenever any new bundle is installed and started in karaf. It checks if the new bundle contains “featureMetaInfo.xml” file or not. If found, see the resources defined at feature level (resources means configurable xml for event, permastore, dynastore, policy etc.). Then it loads, parse and add its configuration into database as well as in data grid(hazelcast).
* Whenever any bundle is stopped or uninstalled from karaf. It checks if the bundle getting uninstalled or stopped contains “featureMetaInfo.xml” file or not. If found, see the resources defined at feature level (resources means configurable xml for event, permastore, dynastore, policy etc.). Then removes its configuration from database as well as in data grid(hazelcast).

***e) “MultipleEndpointCustomPlugin” project:***

We have written a custom maven plugin to generate “soap endpoint” and its “route definition” for the feature.

* It has to be added to feature project as plugin in pom.xml. During the build time, the plugin read the “featureservice.xml” file, get the data required to create “endpoint” and its “route definition”.
* Once “endpoint” and “route definition” is created, write it into “\*Execution.xml” available in “resources/META-INF/spring” folder of the project where plugin is added in pom.xml.
* Plugin should be added to all the feature project which support “soap endpoint”.

### 2.3.2) Features:

***a) “roi-labelService” project:***

We have written a code which will generated the batchid, add label template by generating labelId when a request is made for add label and create an actual value substituted label at produce label request and store it in dynastore against generated batchid, triggers another feature to print label by passing batchid.

* It is created on top of core framework “roi-framework” and “mesh-core”. It is roi implemented label service.
* Feature project must contain a xml file name ‘featureMetaInfo.xml” into resource folder which defines all the resource (event, featureimpl, permastore etc.) supported by feature.
* Event xml file configure the channel, system event, component event and service event for the feature. Permstore xml file contain configuration to store data required by application in permastore. featureImpl xml to specify the service supported by feature, endpoints where these service are exposed and implementation route name associated with service.
* Label Template, Printer config loaded from text file to permastore and on service call template or printer details taken from permastore instead of loading it from text file
* Services supported by label service are “startlabel” – which returns batchid, “addlabel” which returns you success message if added successfully. It can be called any number of time. “producelabel” which returns all label in concatenated with value replaced for a batchid. “voidlabel” which trigger another feature as event using rest http using “Post” with JSON as the payload.

Note: “voidlabel” service is not mean to call printservice instead it supposed to empty the cache if called. But here we just want to show how one feature can call another through event.

***b) “roi-label-niceLabelService” project:***

We have written a code which will generated the batchid, add label template and labelId when a request is made for add label and create an actual value substituted label at produce label request and store it in dynastore against generated batchid, triggers another feature to print label by passing batchid and printer id.

* It is created on top of core framework “roi-framework” and “mesh-core”. It is nice-label implemented label service.
* Feature project must contain a xml file name ‘featureMetaInfo.xml” into resource folder which defines all the resource (event, featureimpl, permastore etc.) supported by feature.
* Event xml file configure the channel, system event, component event and service event for the feature. featureImpl xml to specify the service supported by feature, endpoints where these service are exposed and implementation route name associated with service.
* Services supported by label service are “startlabel” – which returns batchid, “addlabel” which returns you success message if added successfully. It can be called any number of time. “producelabel” which returns all label in concatenated with value replaced for a batchid then trigger another feature as event using rest http using “Post” with JSON as the payload.

***c) “roi-printservice-printnode” project:***

We have written a code which get the batchid from request data, get the label data from the dynastore using batchid as key and send to create a “printJobId” to printing server. Also used to get status of “printJobId” for print node server

* It is created on top of core framework “roi-framework” and “mesh-core”. It is printnode implemented print service.
* Feature project must contain a xml file name ‘featureMetaInfo.xml” into resource folder which defines all the resource (event, featureimpl, permastore etc.) supported by feature.
* Event xml file configure the channel, system event, component event and service event for the feature. featureImpl xml to specify the service supported by feature, endpoints where these service are exposed and implementation route name associated with service.
* This feature is triggered either from “roi-labelservice” or “roi-label-niceLabelService” as an event. Its responsibility is to generate “printJobId” for the label store in dynastore for key as batchid. And Once “printJobId” is generated get the job status.
* To connect to printnode service, require printnode api key.

***d) “roi-printservice-cups” project:***

We have written a code which get the batchid from request data, get the label data from the dynastore using batchid as key and send to create a “printJobId” to printing server. Also used to get status of “printJobId” for cups printer.

* It is created on top of core framework “roi-framework” and “mesh-core”. It is cups implemented print service.
* Feature project must contain a xml file name ‘featureMetaInfo.xml” into resource folder which defines all the resource (event, featureimpl, permastore etc.) supported by feature.
* Event xml file configure the channel, system event, component event and service event for the feature. featureImpl xml to specify the service supported by feature, endpoints where these service are exposed and implementation route name associated with service.
* This feature is triggered either from “roi-labelservice” or “roi-label-niceLabelService” as an event. Its responsibility is to generate “printJobId” for the label store in dynastore for key as batchid. And Once “printJobId” is generated get the job status.

## 2.3) Database setup for Application:

Application is configured with 2 types of database schema. One used for configuration level and other used for application level.

Note: write now we are using same database schema for both configuration and application level.

### 2.3.1) Configuration Level database setup:

A property file named “configurationDB.properties” available in “roi-framework” project inside “resources” folder. Change it to the database properties of your system

### 2.3.2) Application Level database setup:

Application database setup is done in “baseRoute.xml” file available in “resources/META-INF/spring” folder of “mesh-core” project for bean declaration whose id is “dataSourceA”. Change it according to the database properties of your system.

## 2.4) Importing codes in Eclipse/Fuse IDE:

Inorder to Import all the code into eclipse/Fuse IDE, go the location where to have clone all codes using git command or tool. Inside you will see a folder called “RoiMeshRepo”, inside “RoiMeshRepo” folder you can see pom.xml.

Therefore, from command line go to the location of pom.xml available in “RoiMeshRepo” folder.

* To build all project specified in parent pom.xml along use command given below. This command will build all project in a sequence its mentioned in module.

“mvn clean install” – run all test cases

OR

“mvn clean install -DskipTests” – skip test cases

* To clean eclipse environment file like .class, .project, .setting use below specified command:

“mvn eclipse:clean”

* To recreate .class, .project, .setting use below specified command so that you can get project on eclipse.

“mvn eclipse:eclipse”

* Import projects in this order to avoid any project build error (this is optional, you can import in any sequence), roi-framework, mesh-core, custom plugin (MultipleEndpointCustomPlugin), features (roi-labelservice, roi-label-niceLabelService, roi-printserivce-print node etc.) and then features-installer.

Note:

* Custom plugin for soap end points only tested with roi-label service (but It will work same for all features). Make sure there is no “cxf route definition” or “end point declaration” defined in “labelServiceExecution.xml” before “mvn clean install” is called because it has to be dynamically generated at build time.

# 3) How to test framework:

Framework is provided with test classes and test suites which runs the test classes. It is used to test the framework is running in expected manner or not. There are different ways we can run this test cases:

Note: Test cases is written only for “roi-framework” project to check the behaviour of configuration service class for event, permatsore, dynastore etc. We have not provided any test case for camel based “mesh-core” framework”.

## 3.1) Run all test cases at once:

In this, all the test cases written for framework (“roi-framework”) is executed at once.

To do this, follow the below steps:

* Go the “roi-framework” project location from command line. You can see the pom.xml file written for the project in that folder.
* Type “mvn clean install” and press enter button from command line. This will run all the test case written for framework one by one and return you the result containing success or failure of test cases. Also build the project.

## 3.2) Run Test suites from eclipse/Fuse IDE:

Test suites are written to run the group of test class from eclipse.

To do this, follow the below steps:

* From eclipse/fuse IDE, open “src/test/java” then open a package name “com.getusroi.test.suite” of project “roi-framework”.
* You can see the class available in the package. You can run this just by right click on the selected class and select “run as junit” on open popup.
* In order to run configuration DAO, permastore or policy related test classes run class named “RoiTestSuite”, to run dynastore related test class run class named “DynastoreTestSuite”, to run feature related test classes, run class named as “FeatureTestSuite”, to sun eventing based test classes, run class named “EventingTestSuite”.

# 4) How to run framework with feature:

## 4.1) Framework with LabelService feature

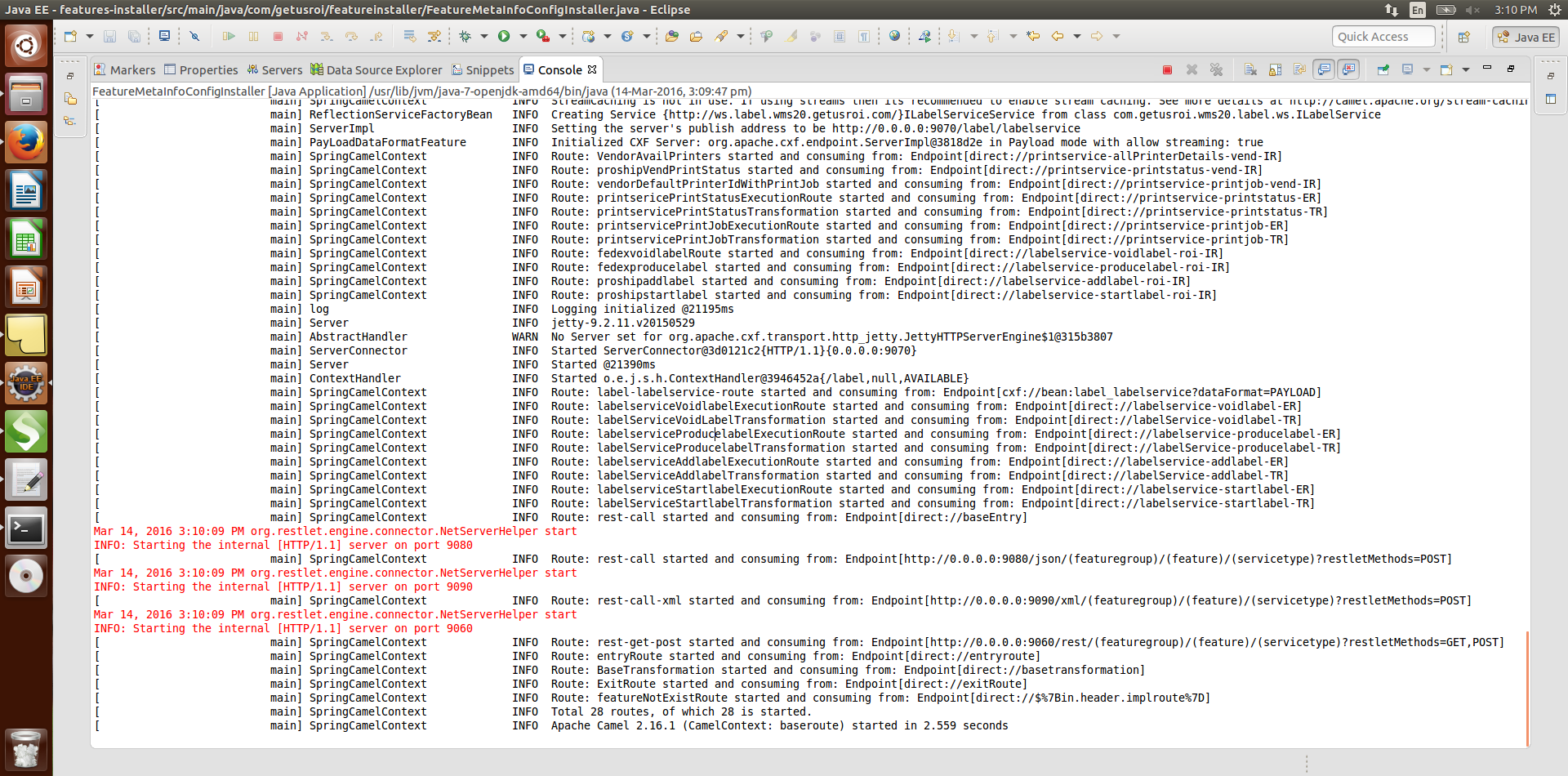
### 4.1.1) Setup framework with feature (roi-labelService and roi-printService-printnode):

To test “roi-framework” with feature (roi-labelService and roi-printService-printnode) you have to follow these steps:

1. In “labelservice-roi-eventing.xml” and “printservice-printnode-eventing.xml”available in resources folder of project “roi-labelservice” and “roi-printservice-printnode”, we have configured File dispatcher channel and provided the local file system path. You need to change it according your suitable file location.
2. You need to change the path of template file available in “TEMPLATE\_LIST.properties”.
3. Make sure “cxf endpoint” and its “route definition” is not available in labelServiceExecution.xml.
4. If available, remove “cxf-endpoint” declaration and its “route definition” form labelServiceExecution.xml but not the declaration of <cxf:bus>.
5. In “roi-printservice-printnode” project “resources” folder for the file “printNodeProperty.properties” changes the “PRINTNODE\_TOKEN” value with one you get during “PrintNode server Setup”.
6. First maven build custom plugin called “MultipleEndpointCustomPlugin”.
7. On maven install on roi-labelService it will freshly add “cxf-endpoint” and its “route definition” in “labelServiceExecution.xml”. Then maven install project “roi-printservice-printnode”.
8. In features-installer project, make sure only “roi-labelservice”, “roi-printservice-printnode” maven dependency is added, all other dependency of feature like “roi-label-niceLabelService”, “roi-printservice-cups” are commented.
9. Perform sequence of command from command line on features-installer project.

* From command line go the features-installer project where you can see pom.xml.
* First do “mvn clean install” followed by “mvn eclipse: clean” and then “mvn eclipse: eclipse”.

1. From eclipse run class contain main () “FeatureMetaInfoConfigInstaller” class. This class will load all the configuration required for the application into the cache and database, then start camel.



### 4.1.2) Testing framework with feature (roi-labelService and roi-printService-printnode):

Application is exposed to 4 different url. LabelService is provided with “startlabel”,” addlabel”,” producelabel”,” voidlabel” service. Let’s look into request url and data need to send for them.

Note:

* “voidlabel” service is not mean to call printservice instead it supposed to empty the cache if called. But here we just want to show how one feature can call another through event.
* Use the batchId which you get from startlabel service response for other service (addlabel, producelabel etc.).

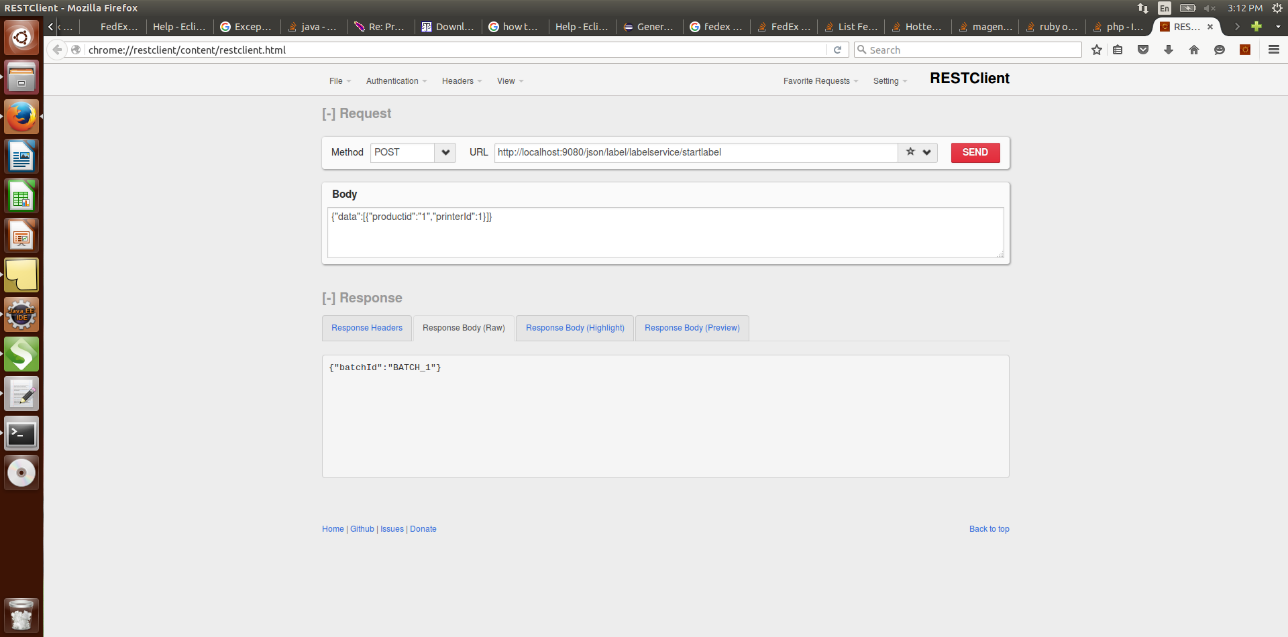
#### a) Generic-json-baseRoute: expose each service as “http” over “post” method with the “JSON” as the payload.

* For start Label service:

URL: <http://localhost:9080/json/label/labelservice/startlabel>

HTTP METHOD: “POST”

REQUEST DATA : {"data":[{“printerId":1}]}

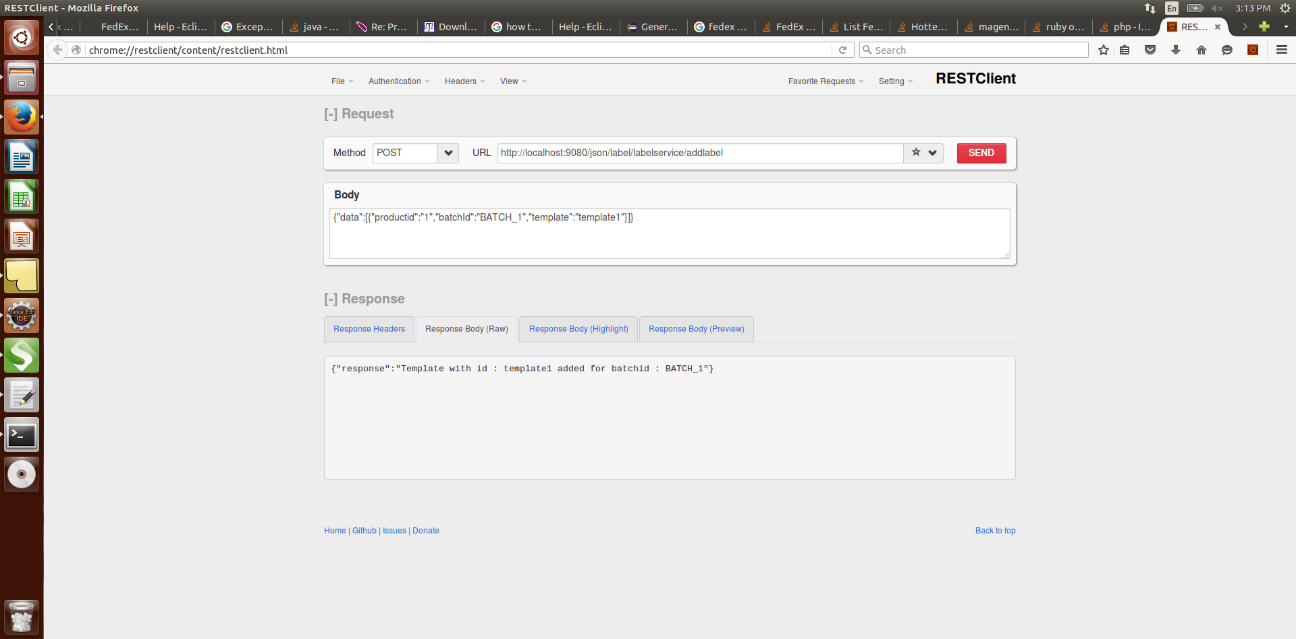


* For Add Label Service: request for add label service can be made any number of time.

URL : <http://localhost:9080/json/label/labelservice/addlabel>

HTTP METHOD: “POST”

REQUEST DATA : {"data”:[{"batchId":"BATCH\_1","template":"template3"}]}

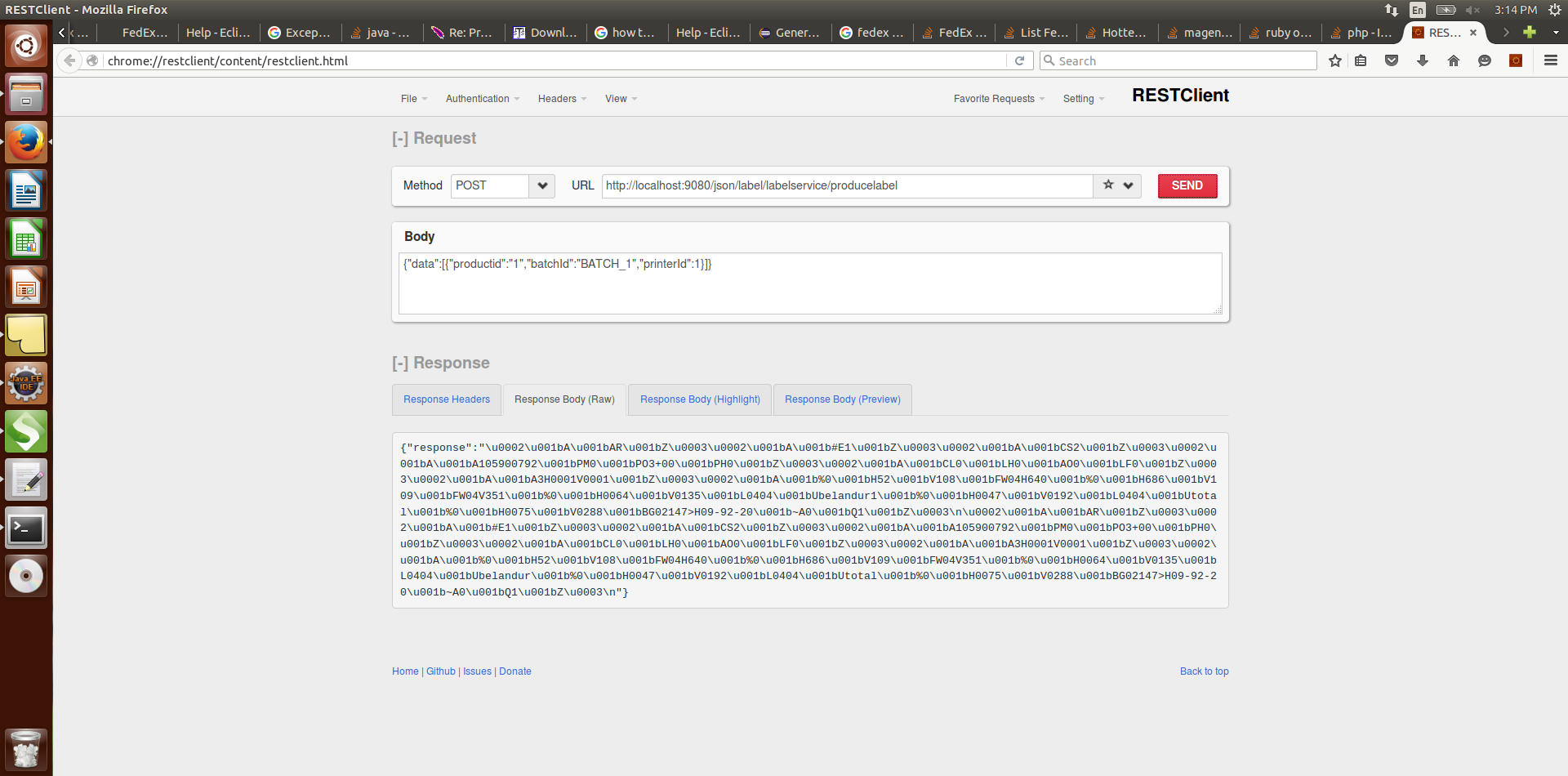


* For Produce Label Service:

URL : <http://localhost:9080/json/label/labelservice/producelabel>

HTTP METHOD: “POST”

REQUEST DATA : {"data":[{“batchId":"BATCH\_1","printerId":1}]}

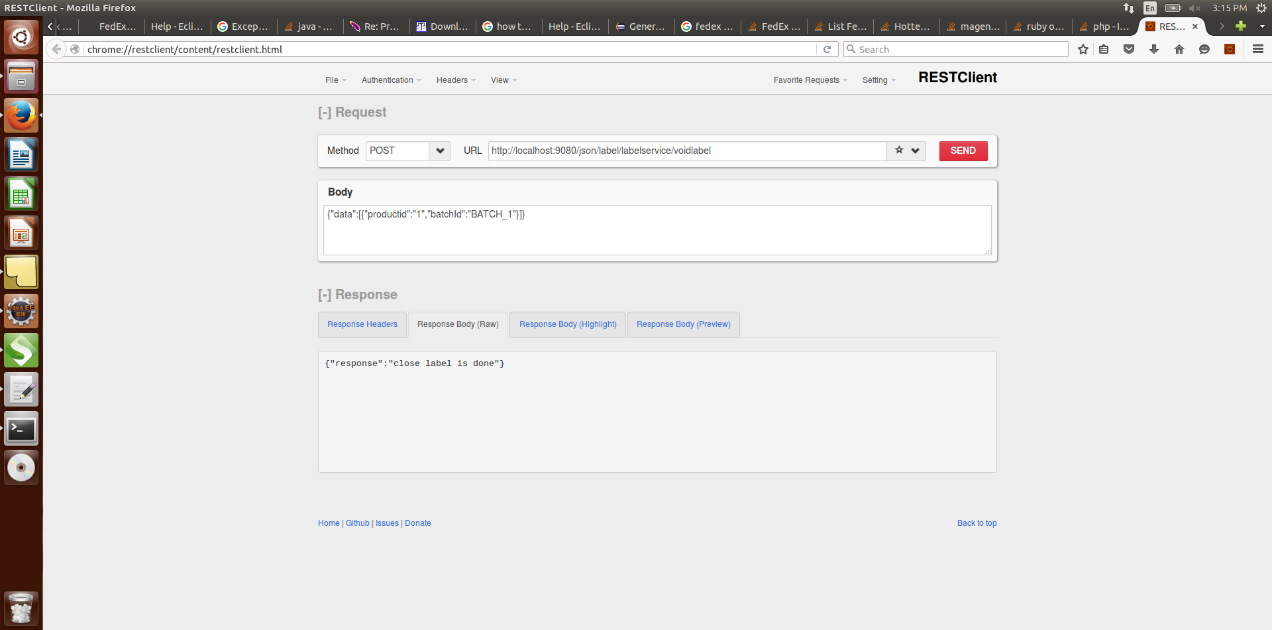


* For void Label Service:

URL : <http://localhost:9080/json/label/labelservice/voidlabel>

HTTP METHOD: “POST”

REQUEST DATA : {"data":[{“batchId":"BATCH\_1"}]}

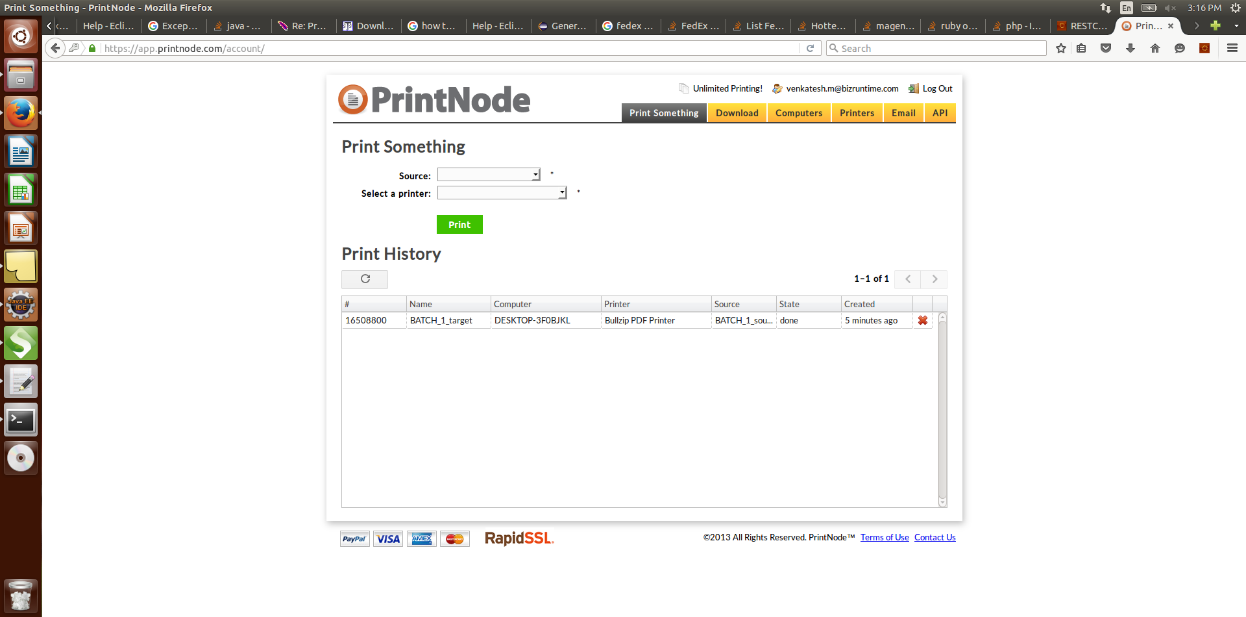


Note: void label call printservice-printnode as an event using Http rest client:

URL : <http://localhost:9080/json/print/printservice/printjob>

HTTP METHOD: “POST”

REQUEST DATA : {"data":[{“batchid":"BATCH\_1"}]}



* For PrintJob Status Service:

URL : [http://localhost:9080/json/ print/printservice/printstatus](http://localhost:9080/json/%20print/printservice/printstatus)

HTTP METHOD: “POST”

REQUEST DATA: {"data":[{“printJobId":"986678"}]}

#### b) Generic-xml-baseRoute: expose each service as “http” over “post” method with the “XMLRPC” as the payload.

* For start label service: Request data

URL: <http://localhost:9090/xml/label/labelservice/startlabel>

HTTP METHOD: “POST”

REQUEST DATA:

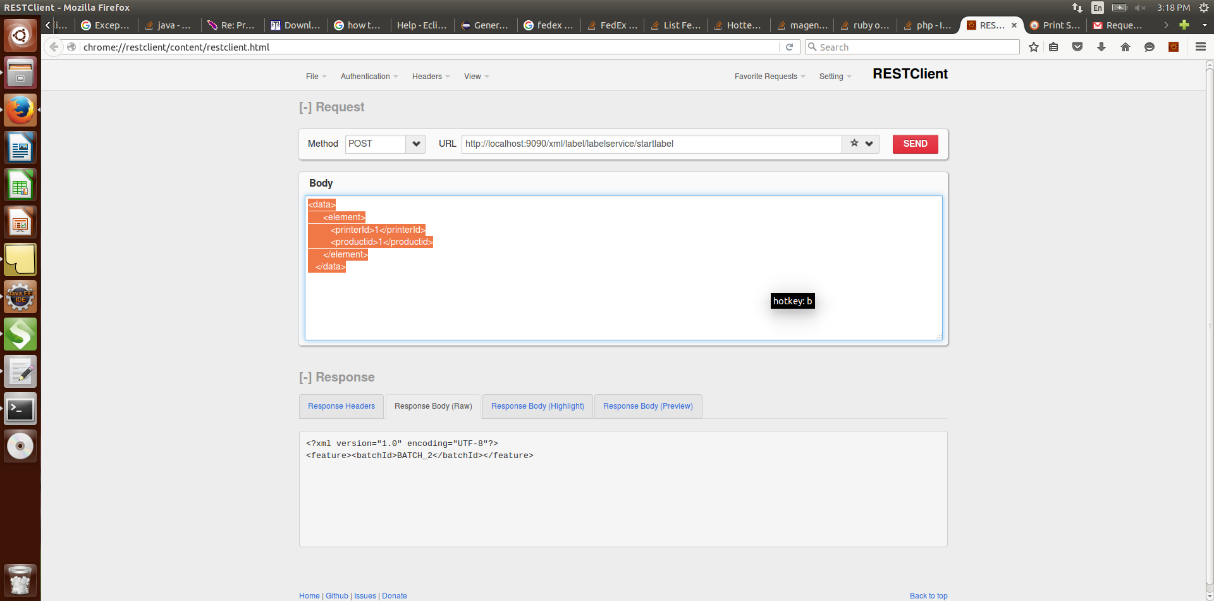
<data>

<element>

<printerId>1</printerId>

<description>start label service</description>

</element>

</data>

* For add label service:

URL: <http://localhost:9090/xml/label/labelservice/addlabel>

HTTP METHOD: “POST”

REQUEST DATA:

<data>

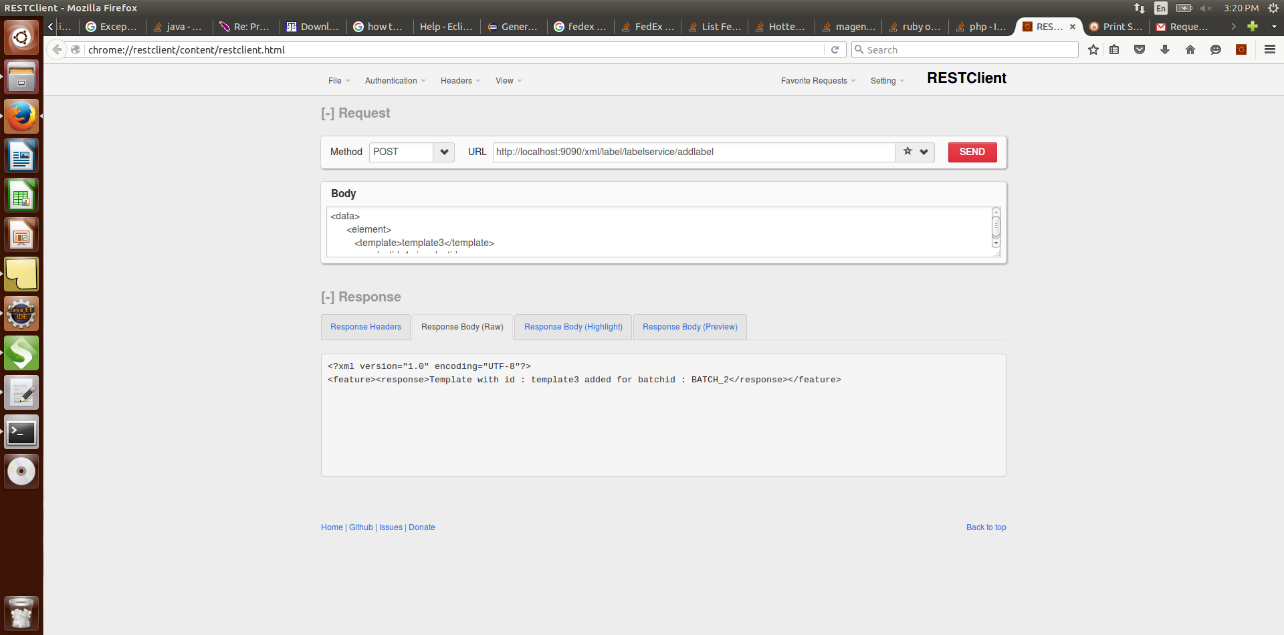
<element>

<batchId>BATCH\_1</batchId>

<template>template3</template>

</element>

</data>



* For produce label service:

URL: <http://localhost:9090/xml/label/labelservice/producelabel>

HTTP METHOD: “POST”

REQUEST DATA:

<data>

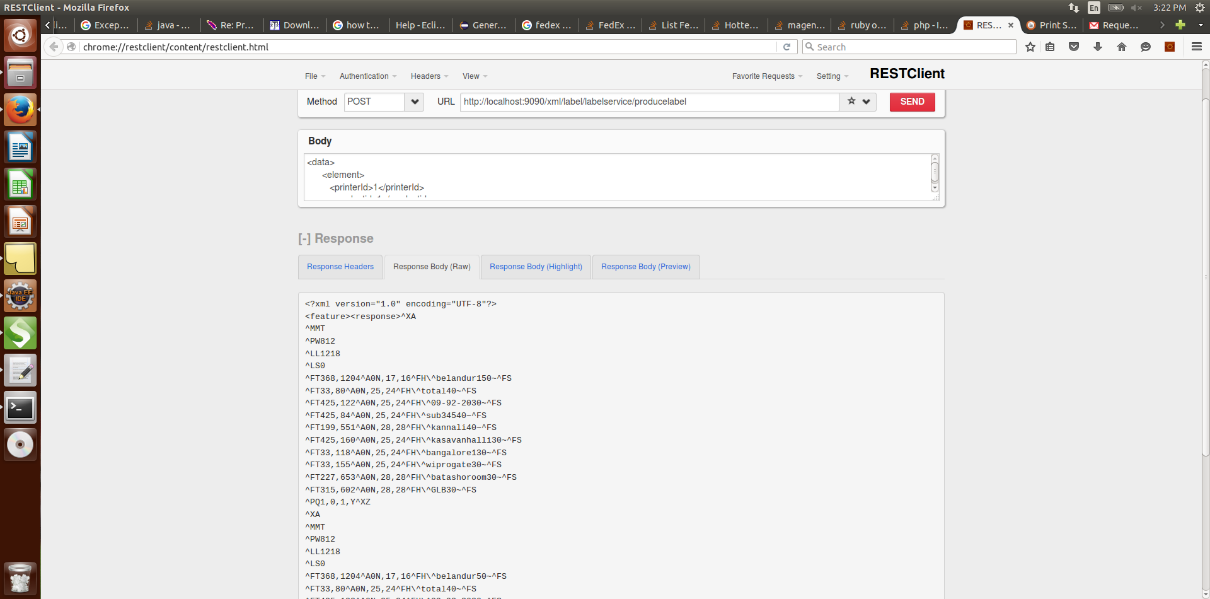
<element>

<batchId>BATCH\_1</batchId>

<printerId>1</printerId>

</element>

</data>



* For close label service:

URL: <http://localhost:9090/xml/label/labelservice/voidlabel>

HTTP METHOD: “POST”

REQUEST DATA:

<data>

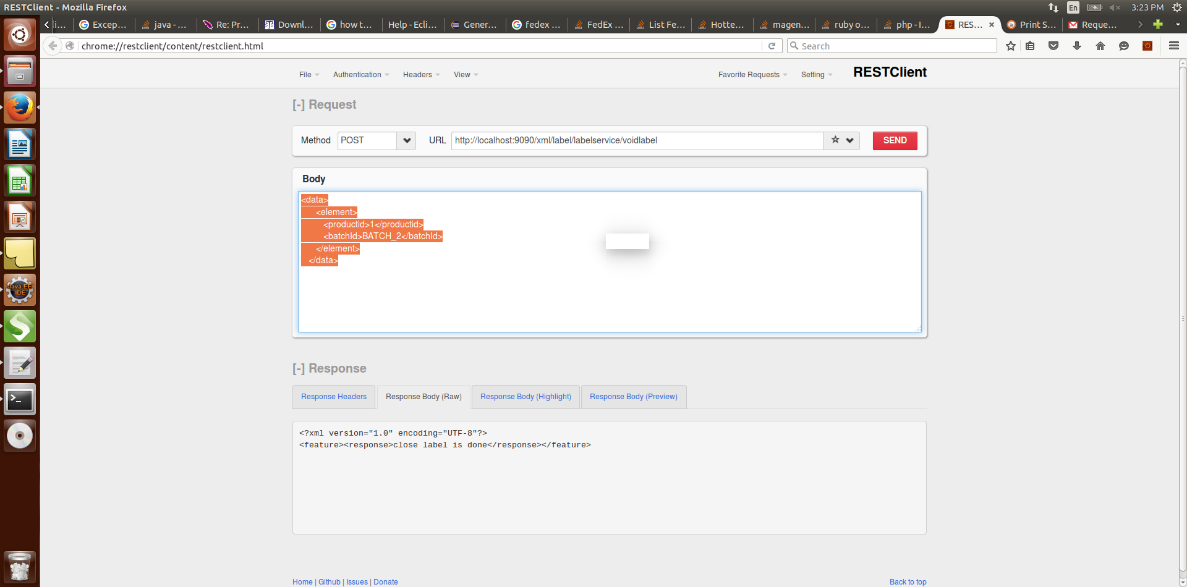
<element>

<batchId>BATCH\_1</batchId>

<description>its close label</description>

</element>

</data>

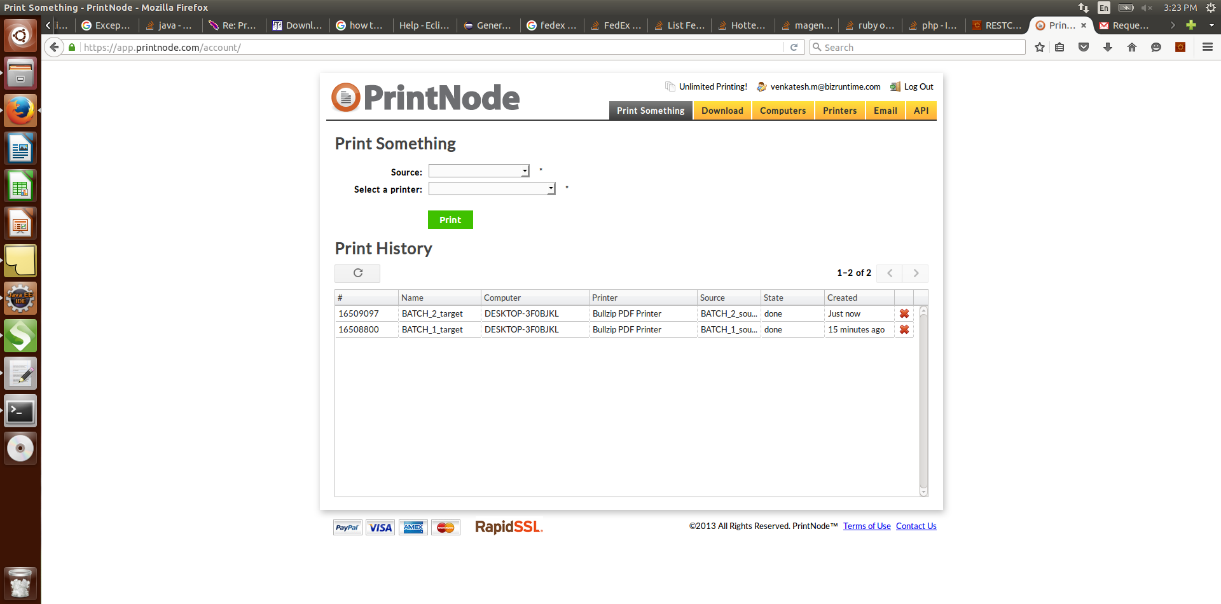


Note: void label call printservice-printnode as an event using Http rest client:

URL: <http://localhost:9080/json/print/printservice/printjob>

HTTP METHOD: “POST”

REQUEST DATA: {"data":[{“batchid":"BATCH\_1"}]}



* For PrintJob Status Service:

URL: [http://localhost:9080/xml/ print/printservice/printstatus](http://localhost:9080/xml/%20print/printservice/printstatus)

HTTP METHOD: “POST”

REQUEST DATA:

<data>

<element>

<printJobId >986678</printJobId>

<description>its print status </description>

</element>

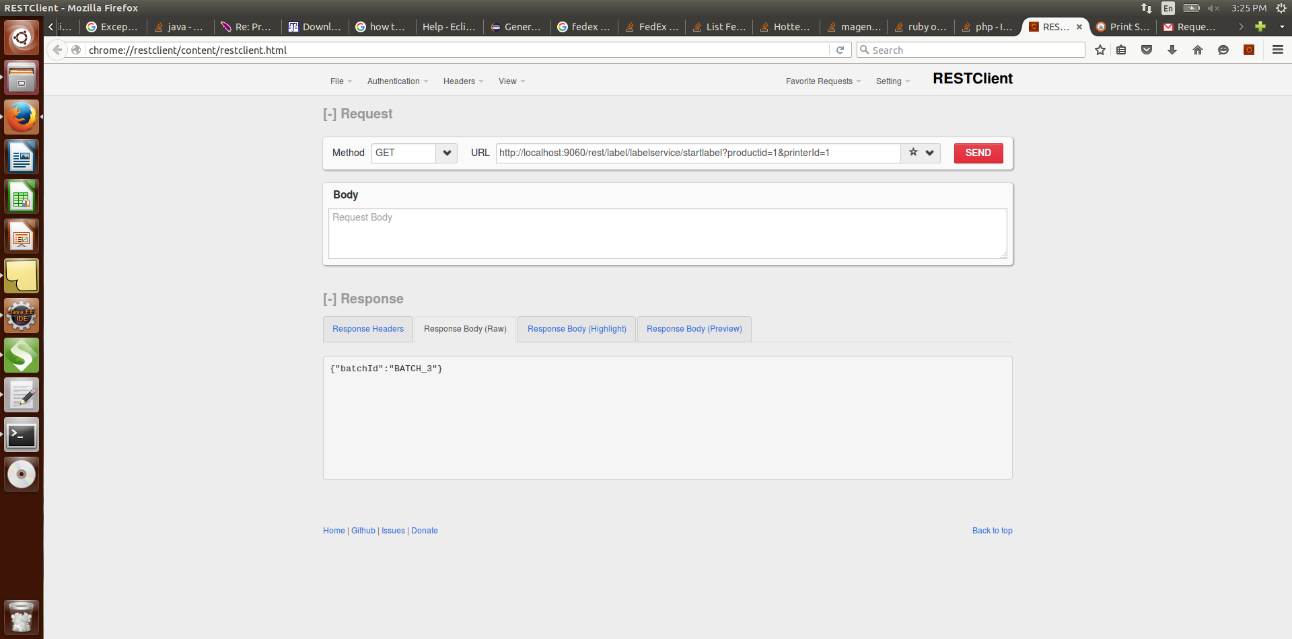
</data>

#### c) Generic-rest-baseRoute: expose each service as “rest” over “GET/POST” method with the “JSON” as the payload.

* For start Label service:

URL: <http://localhost:9060/rest/label/labelservice/startlabel?printerId=1>

HTTP METHOD: “GET”

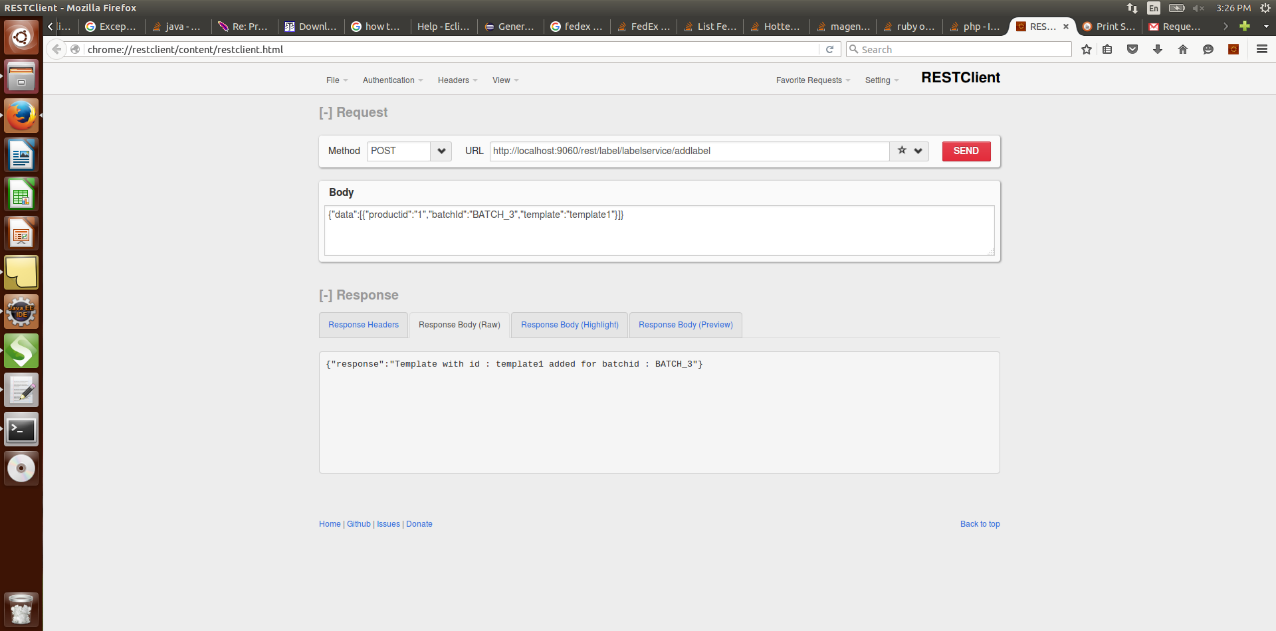


* For Add Label Service: request for add label service can be made any number of time.

URL: <http://localhost:9060/rest/label/labelservice/addlabel>

HTTP METHOD: “POST”

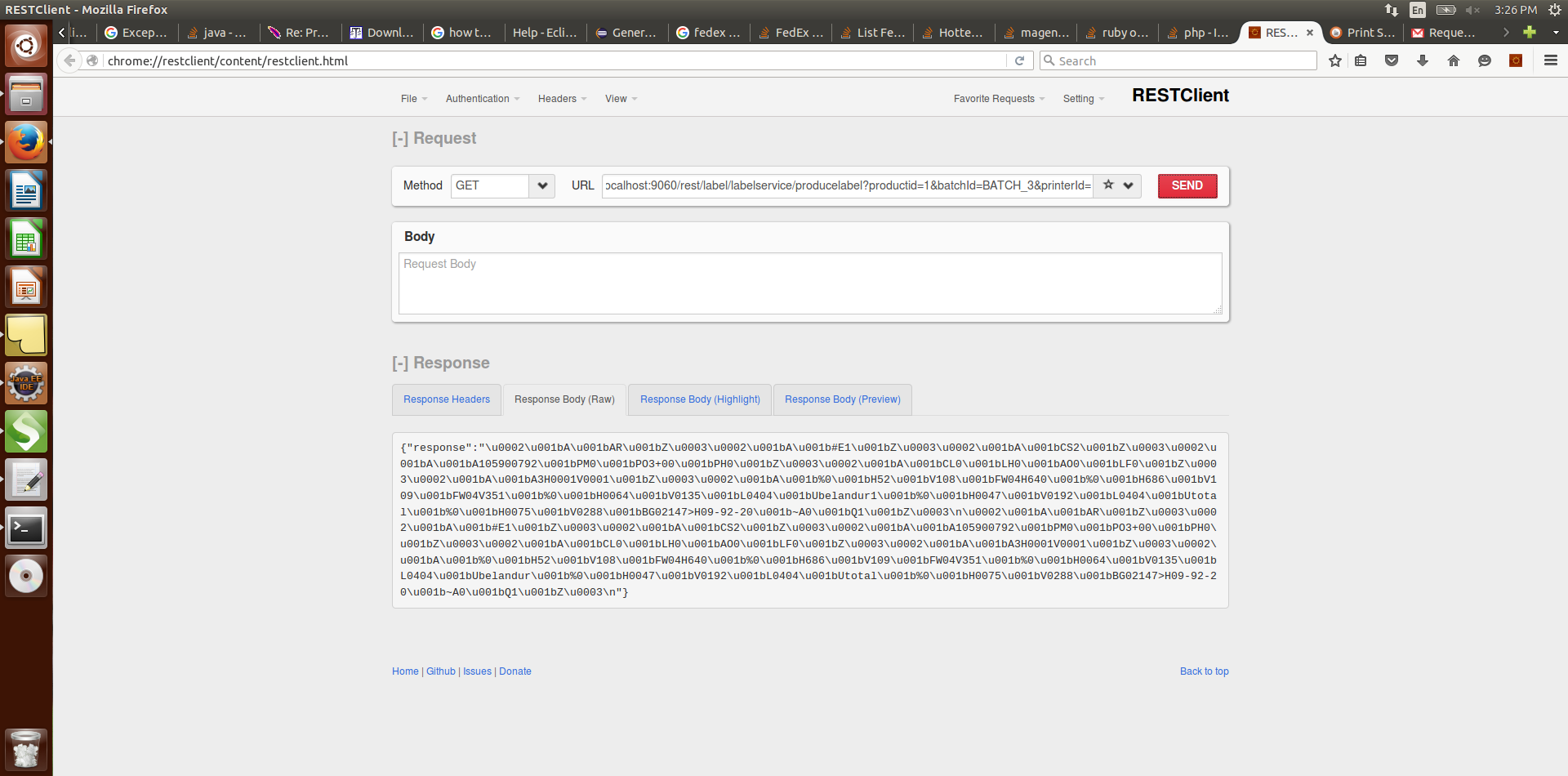
REQUEST DATA: {"data”:[{"batchId":"BATCH\_1","template":"template1"}]}



* For Produce Label Service:

URL: <http://localhost:9060/rest/label/labelservice/producelabel?batchId=BATCH_1&printerId=1>

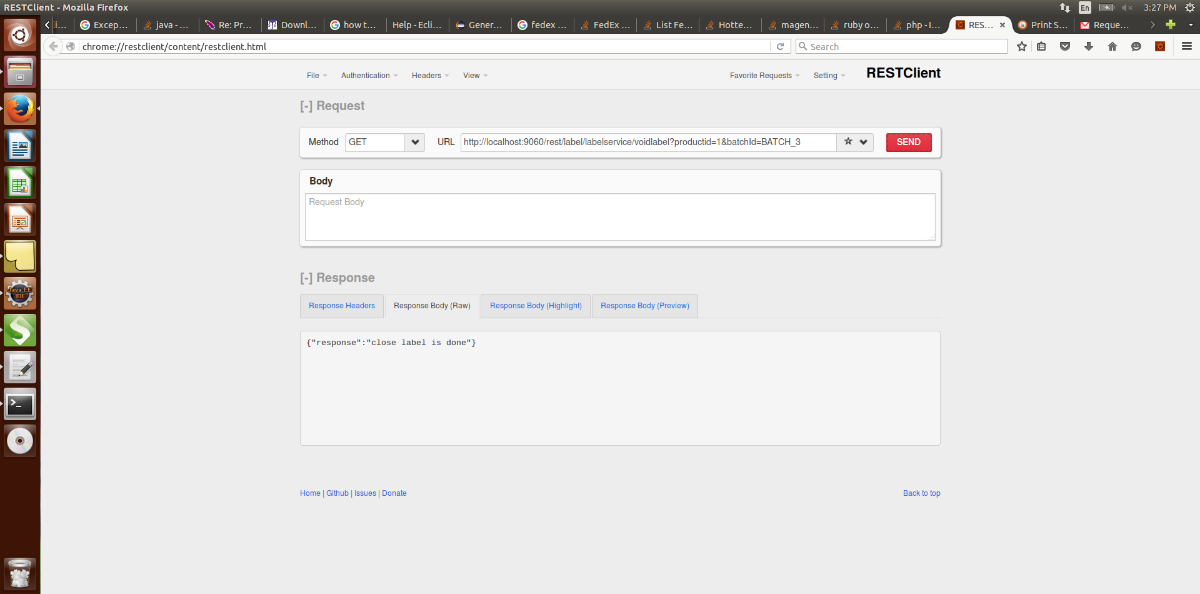
HTTP METHOD: “GET”



* For void Label Service:

URL : <http://localhost:9060/rest/label/labelservice/voidlabel?batchId=BATCH_1>

HTTP METHOD: “GET”

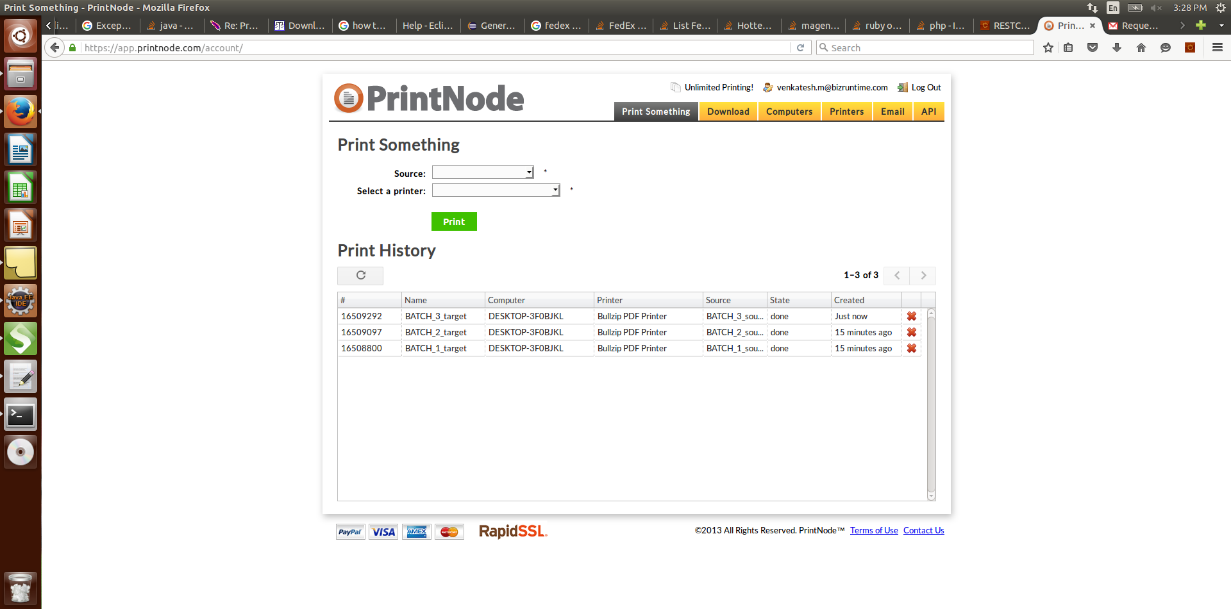


Note: void label call printservice-printnode as an event using Http rest client:

URL: <http://localhost:9080/json/print/printservice/printjob>

HTTP METHOD: “POST”

REQUEST DATA : {"data":[{“batchid":"BATCH\_1"}]}



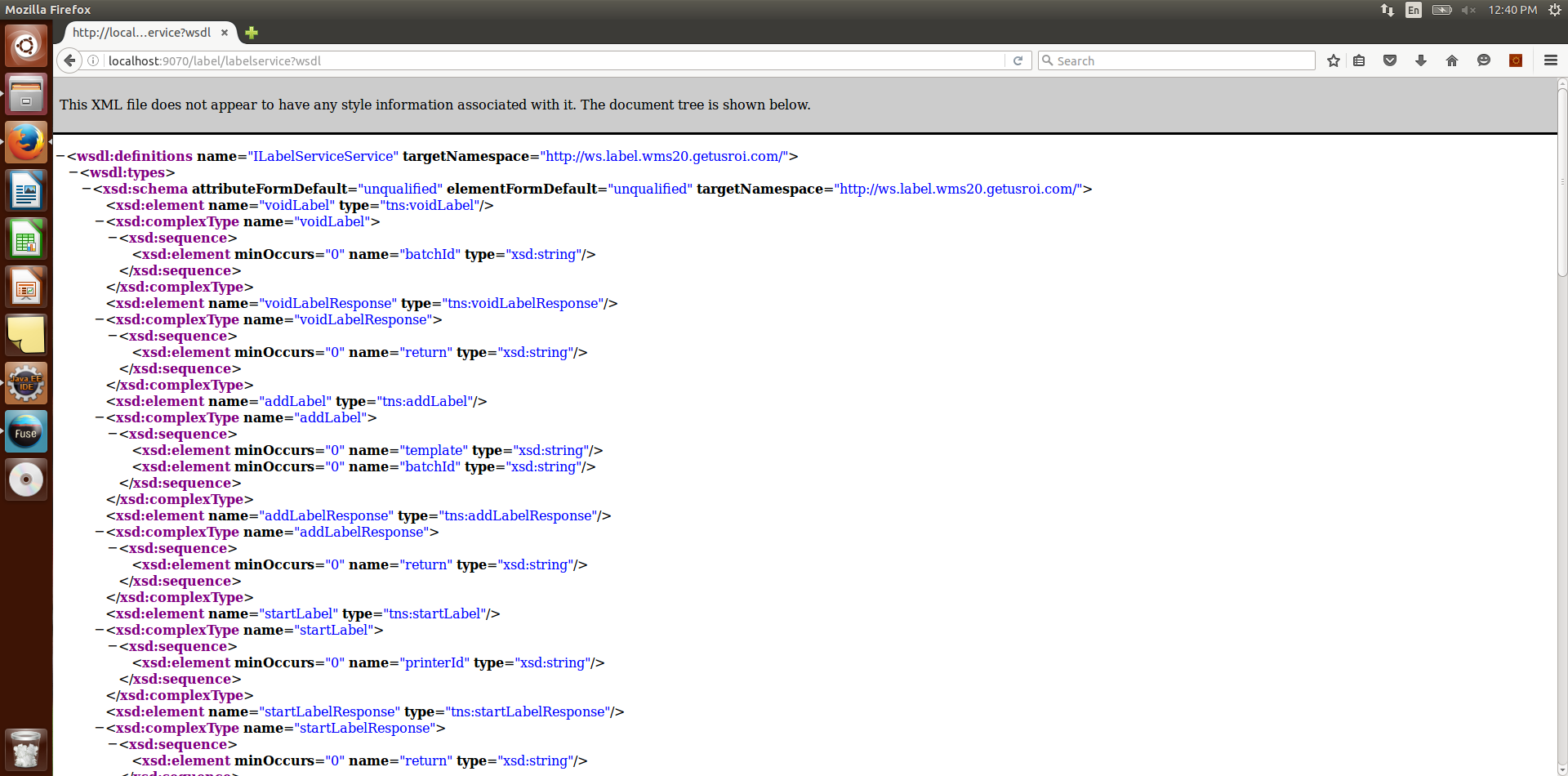
* For PrintJob Status Service:

URL : [http://localhost:9060/rest/ print/printservice/printstatus?printJobId=986678](http://localhost:9060/rest/%20print/printservice/printstatus?printJobId=986678)

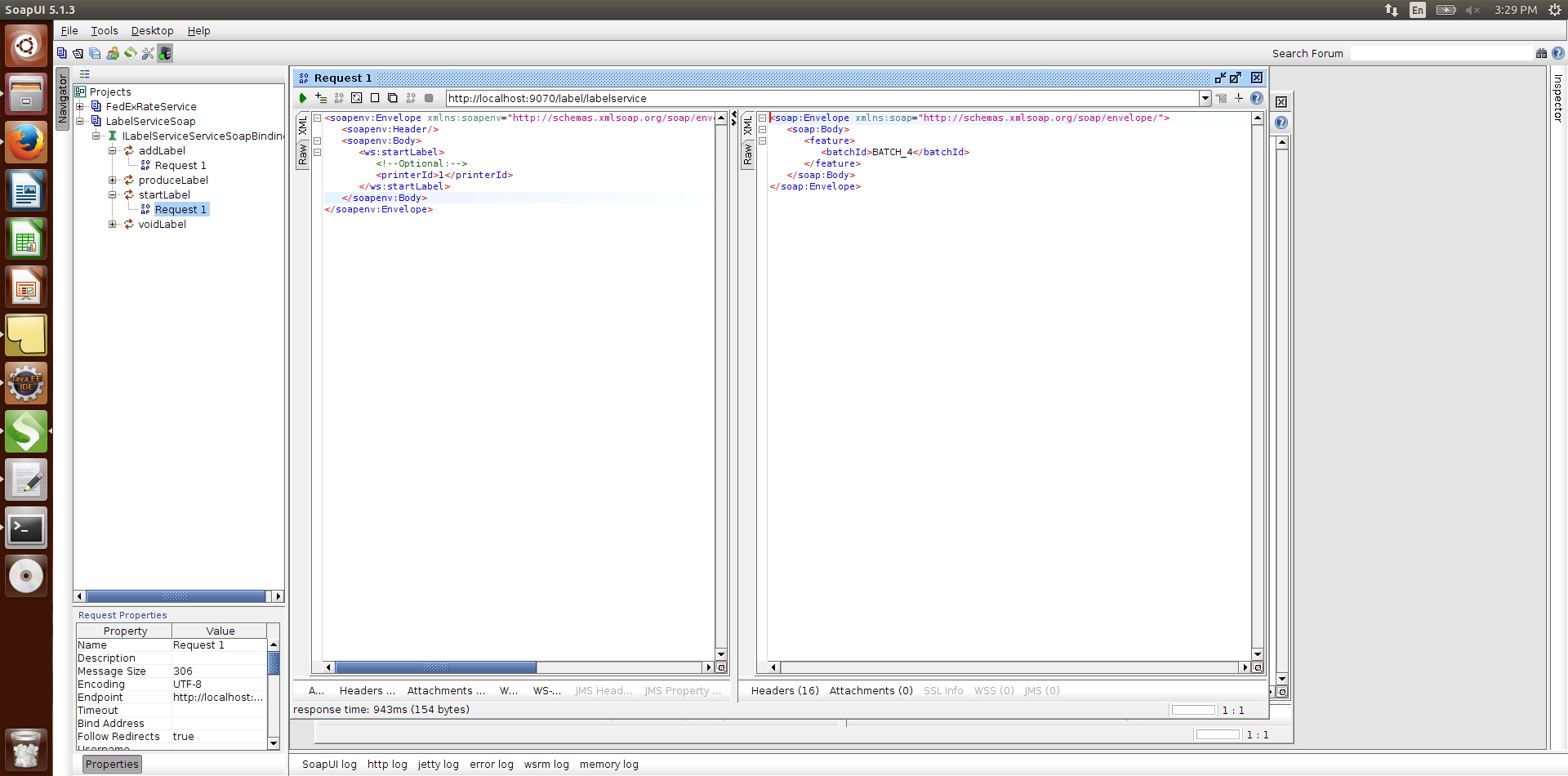
HTTP METHOD: “GET”

#### d) Generic-soap-baseRoute: expose each service as “SOAP” over “POST” method with the “soap” as the payload:

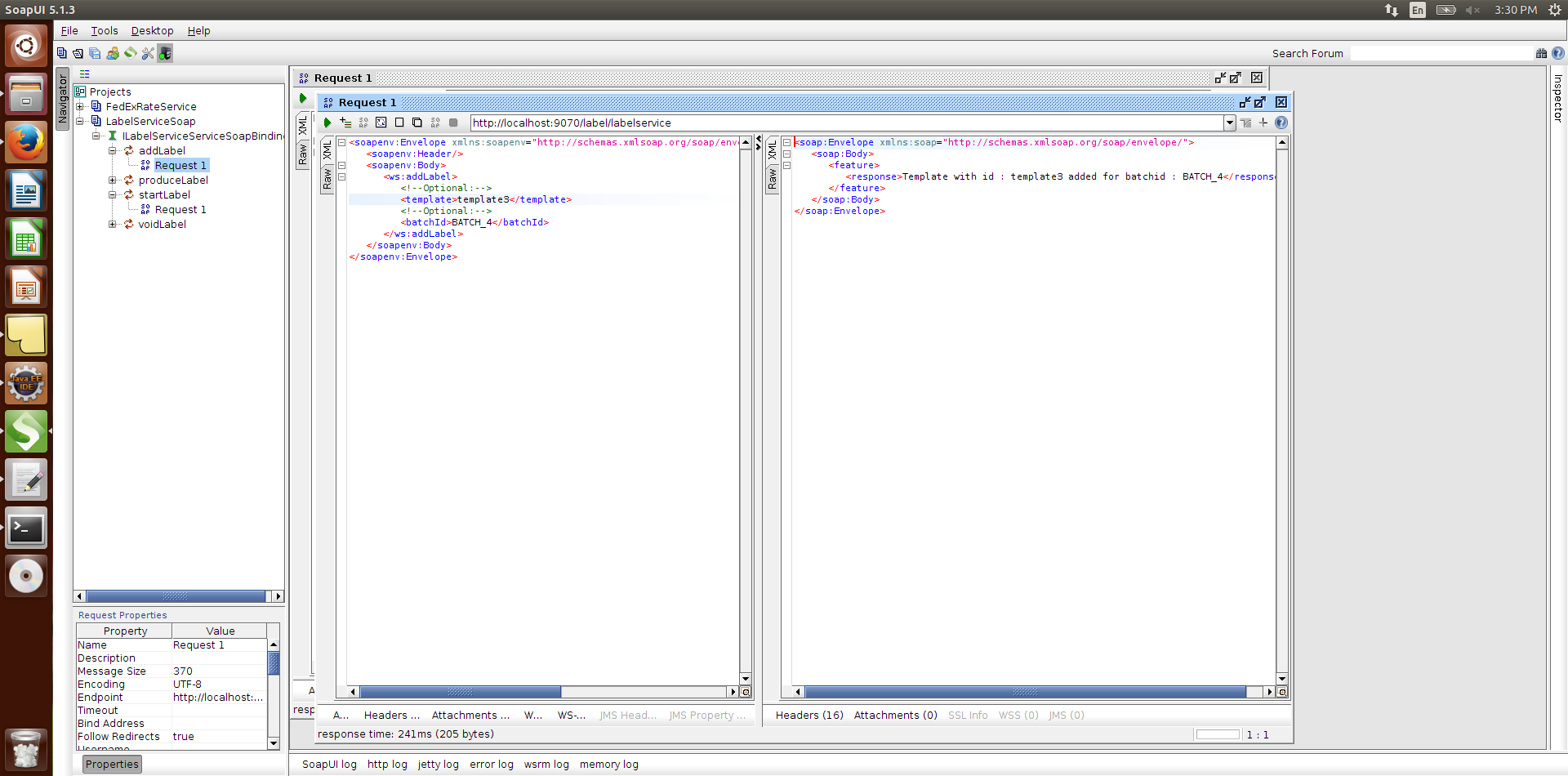
On the browser, put the url as “http://<hostname>:9070/soap/<featuregroup>/<feature>?wsdl. It will generate the wsdl for the feature supporting all service required for the feature. For example, for label service feature the wsdl would open at “http://<hostname>:9070/soap/label/labelservice?wsdl” as shown in below screenshot.



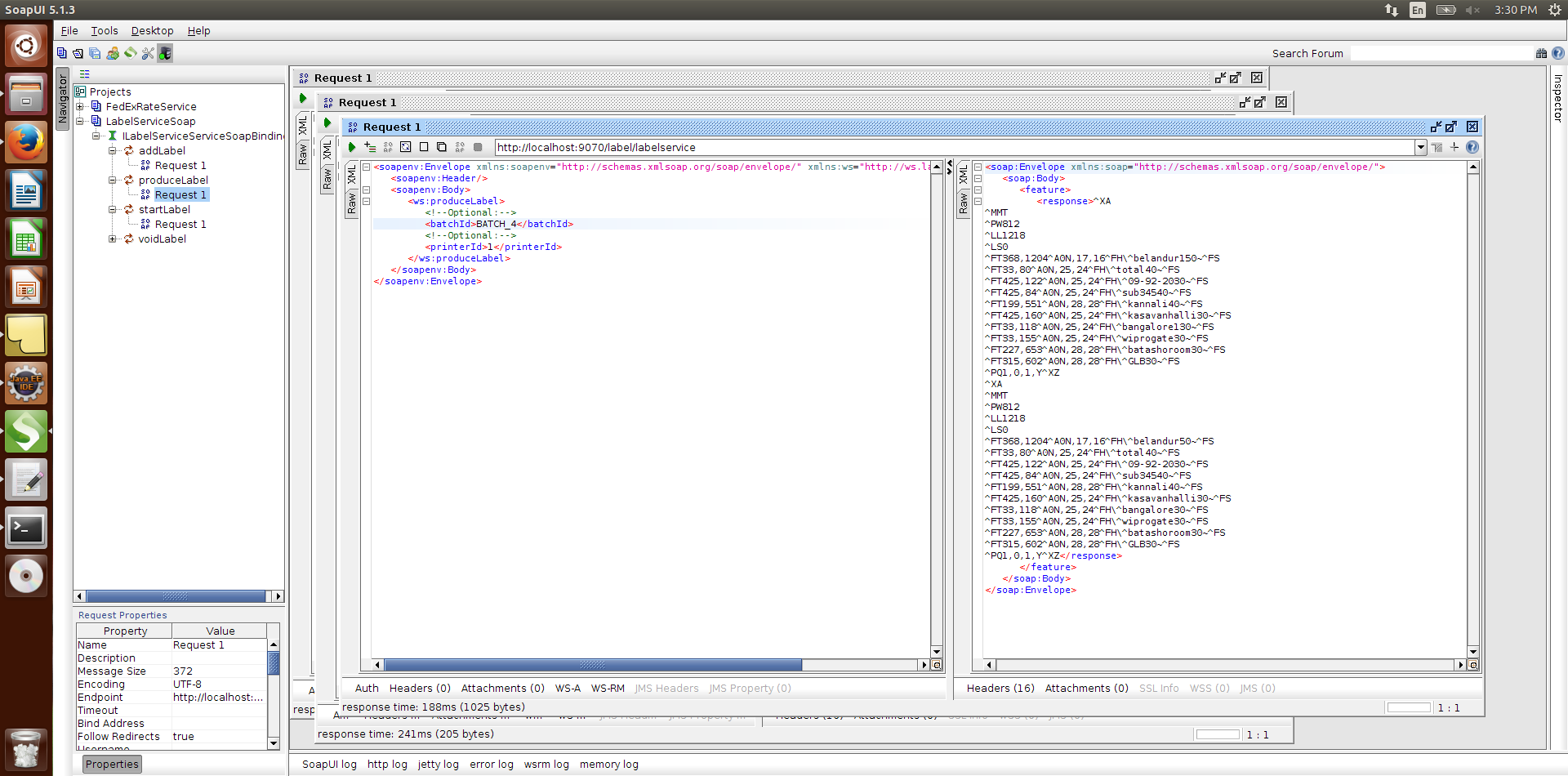
* For start label service:



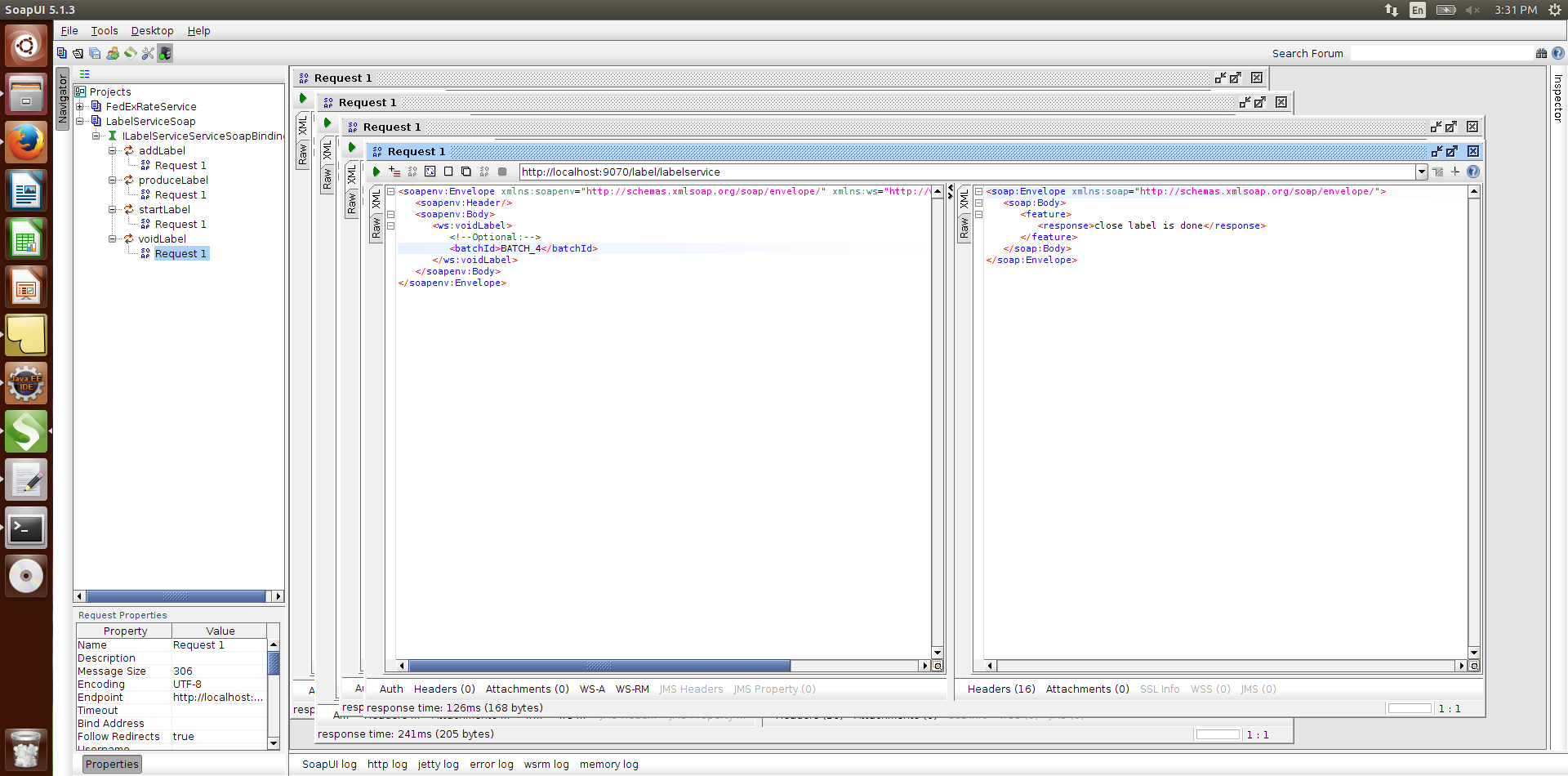
* For add label service:



* For produce label service:



* For void label service:



Note: void label call printservice-printnode as an event using Http rest client:

URL: <http://localhost:9080/json/print/printservice/printjob>

HTTP METHOD: “POST”

REQUEST DATA : {"data":[{“batchid":"BATCH\_1"}]}

