POLITECNICO
MILANO 1863

Computer Science and Engineering

Code Inspection

February 4, 2017

Prof. Luca Mottola

Authors:

- ZHOU YINAN(Mat. 872686)
- ZHAO KAIXIN(Mat. 875464)
- ZHAN YUAN(Mat. 806508)

Contents

1 Classes Assigned

We have been assigned two classes :

- ProductionRun.java
- ViewerServletRequest.java

The namespace patten is :

```
../apache-ofbiz-16.11.01/applications/manufacturing/src/main/java/org/apache/ofbiz
/manufacturing/jobshopmgt/ProductionRun.java
```

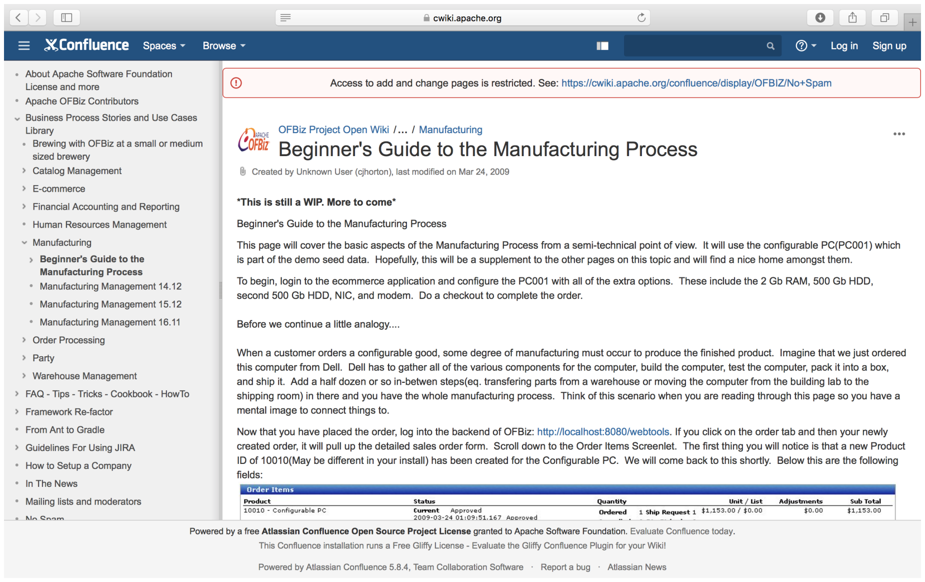
```
../apache-ofbiz-16.11.01/specialpurpose/birt/src/main/java/org/apache/ofbiz/birt
/report/servlet/ViewerServletRequest.java
```

2 Functional Role

2.1 ProductionRun.java

Instead of directly looking into the code, we first examine the online ofbiz document to get information of this class. This class belongs to **Manufacturing** section. The link to the Online Documet.

The situation is described here. After a client makes order, configurable goods which our company provide require some type of manufacturing or production. If we do not have the requiring parts in our inventory, a production run is generated.


cwiki.apache.org
Confluence
Spaces
Browse
Log in
Sign up
About Apache Software Foundation License and more
Apache OFBiz Contributors
Business Process Stories and Use Cases Library
Brewing with OFBiz at a small or medium sized brewery
Catalog Management
E-commerce
Financial Accounting and Reporting
Human Resources Management
Manufacturing
Beginner's Guide to the Manufacturing Process
Manufacturing Management 14.12
Manufacturing Management 15.12
Manufacturing Management 16.11
Order Processing
Party
Warehouse Management
FAQ - Tips - Tricks - Cookbook - HowTo
Framework Re-factor
From Ant to Gradle
Guidelines For Using JIRA
How to Setup a Company
In The News
Mailing lists and moderators
No Spam

Access to add and change pages is restricted. See: https://cwiki.apache.org/confluence/display/OFBIZ/No+Spam

OFBiz Project Open Wiki /… / Manufacturing

Beginner's Guide to the Manufacturing Process

Created by Unknown User (cjhorton), last modified on Mar 24, 2009

This is still a WIP. More to come

Beginner's Guide to the Manufacturing Process

This page will cover the basic aspects of the Manufacturing Process from a semi-technical point of view. It will use the configurable PC(PC001) which is part of the demo seed data. Hopefully, this will be a supplement to the other pages on this topic and will find a nice home amongst them.

To begin, login to the ecommerce application and configure the PC001 with all of the extra options. These include the 2 Gb RAM, 500 Gb HDD, second 500 Gb HDD, NIC, and modem. Do a checkout to complete the order.

Before we continue a little analogy....

When a customer orders a configurable good, some degree of manufacturing must occur to produce the finished product. Imagine that we just ordered this computer from Dell. Dell has to gather all of the various components for the computer, build the computer, test the computer, pack it into a box, and ship it. Add a half dozen or so in-betwen steps(eq. transfering parts from a warehouse or moving the computer from the building lab to the shipping room) in there and you have the whole manufacturing process. Think of this scenario when you are reading through this page so you have a mental image to connect things to.

Now that you have placed the order, log into the backend of OFBiz: http://localhost:8080/webtools. If you click on the order tab and then your newly created order, it will pull up the detailed sales order form. Scroll down to the Order Items Screenlet. The first thing you will notice is that a new Product ID of 10010(May be different in your install) has been created for the Configurable PC. We will come back to this shortly. Below this are the following fields:

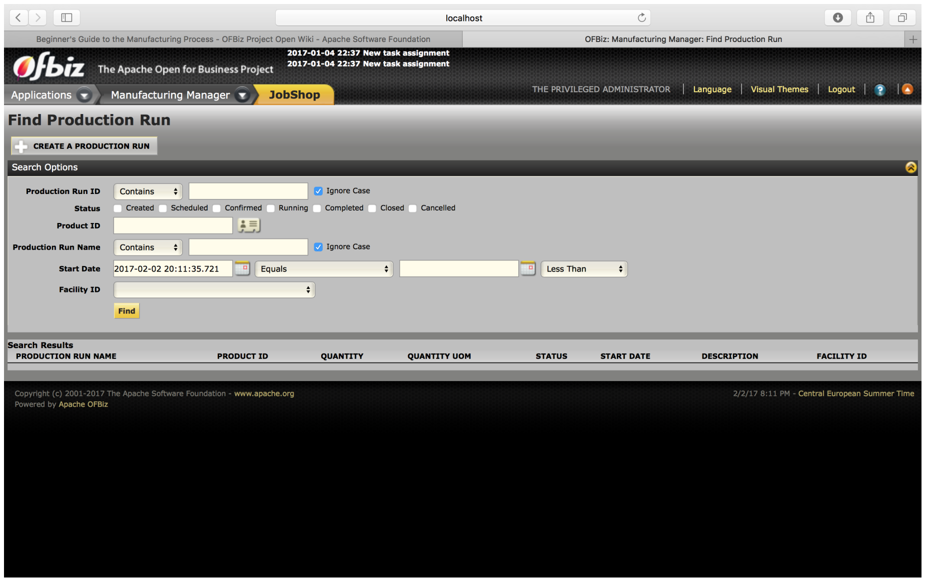
Order Items

Product	Status	Quantity	Unit / List	Adjustments	Sub Total
10010 - Configurable PC	Current Approved 2009-03-24 01:09:51.167 Approved	Ordered 1 Ship Request 1	$1,153.00 / $0.00	$0.00	$1,153.00

Powered by a free **Atlassian Confluence Open Source Project License** granted to Apache Software Foundation. Evaluate Confluence today.
This Confluence installation runs a Free Gliffy License - Evaluate the Gliffy Confluence Plugin for your Wiki!

Powered by Atlassian Confluence 5.8.4, Team Collaboration Software · Report a bug · Atlassian News

If we log into the ofbiz web application, we can examine the production run section. The ProductionRun class manages all the information of a certain production run activity.


localhost
Beginner's Guide to the Manufacturing Process - OFBiz Project Open Wiki - Apache Software Foundation
OFBiz: Manufacturing Manager: Find Production Run
The Apache Open for Business Project
2017-01-04 22:37 New task assignment
2017-01-04 22:37 New task assignment
Applications
Manufacturing Manager
JobShop
THE PRIVILEGED ADMINISTRATOR
Language
Visual Themes
Logout
Find Production Run
CREATE A PRODUCTION RUN
Search Options
Find
Search Results
Copyright (c) 2001-2017 The Apache Software Foundation - www.apache.org
Powered by Apache OFBiz
2/2/17 8:11 PM - Central European Summer Time

Production Run ID	Contains; Ignore Case ✓
Status	Created, Scheduled, Confirmed, Running, Completed, Closed, Cancelled
Product ID	
Production Run Name	Contains; Ignore Case ✓
Start Date	2017-02-02 20:11:35.721; Equals; Less Than
Facility ID	

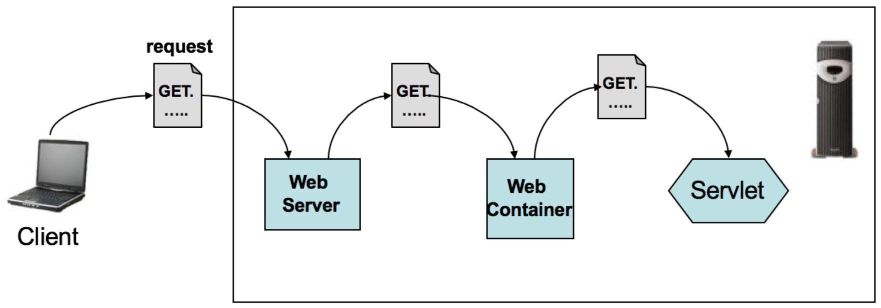
PRODUCTION RUN NAME	PRODUCT ID	QUANTITY	QUANTITY UOM	STATUS	START DATE	DESCRIPTION	FACILITY ID

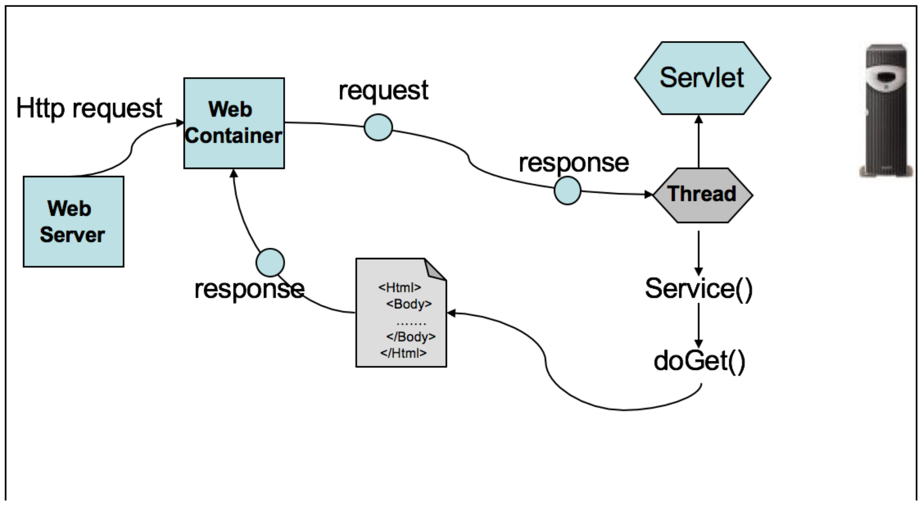
2.2 ViewerServeltRequest

By looking at the code, we find that ViewerservletRequest extends HttpServletRequestWrapper. A "HttpServletRequestWrapper" provides a convenient implementation of the HttpServletRequest interface that can be subclassed by developers wishing to adapt the request to a Servlet. Thus the role of this class is to represent a specific function of HttpServletRequest. More specifically, this function is getParameter();

Before looking into this function, let's recall what is a servlet. A servlet lives in a web container, and it is responsible for generating dynamic web contents. Servlet can be viewed as a special java class without main methods. After a client sends a HTTP request to the web server, the web container is responsible for :

- create an instance of a servlet
- call specific method of a servlet
- destroy a servlet

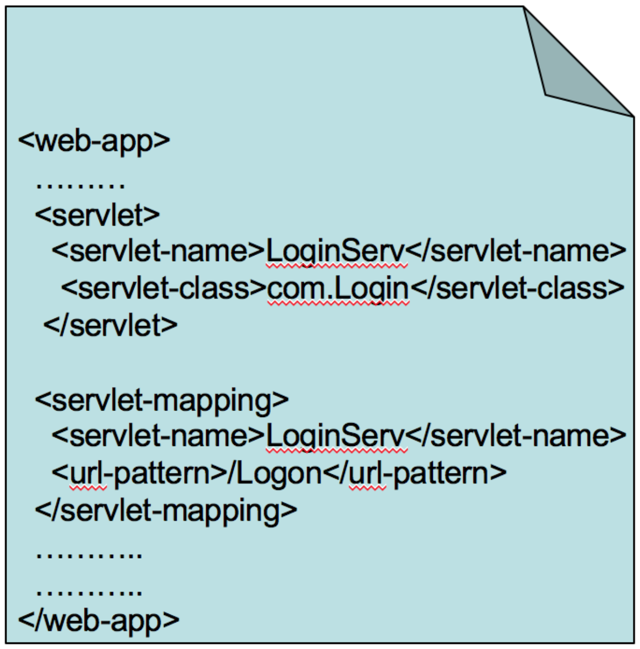




The web container knows which servlet to call because a servlet can have three names :

- Client knows URL name
- Deployer knows servlet secret internal name
- Actual java class name

The XML document is responsible for deployment.



```
<web-app>
  .........
  <servlet>
    <servlet-name>LoginServ</servlet-name>
     <servlet-class>com.Login</servlet-class>
   </servlet>

  <servlet-mapping>
    <servlet-name>LoginServ</servlet-name>
    <url-pattern>/Logon</url-pattern>
  </servlet-mapping>
  ..........
  ..........
</web-app>
```

Web.xml

Now let's look at what function role is this class. **ServletRequest** defines an object to provide client request information to a servlet. The servlet container creates a ServletRequest object and passes it as an argument to the servlet's service method. A **ServletRequest** object provides data including parameter name and values, attributes, and an input stream.
This java class file is used to form the parameter.

3 Check List

3.1 ProductionRun.java

3.1.1 Naming Conventions

1. All class names, interface names, method names, class variables, method variable, and constant used are meaningful.

2. The only one one-character variable in this ProductionRun.java file is
   GenericEntityException e
   And it is used as a parameter for the exception, for catching the

statement. Therefore it is used in the loop, and it is temporary ”throwaway” variable.

3. Class name of this ProductionRun.java file is:
   ProductionRun
   Therefore, it is with the first letter of each word in capitalized.

4. There is no ant defined interface within the ProductionRun.java file.

5. The methods of this file are as follow:
   exist()
   getGenericValue()
   store()
   getProductionProduced()
   getQuantity()
   setQuantity()
   getEstimatedStartDate()
   setEstimatedStartDate()
   getEstimatedCompletionDate()
   setEstimatedCompletionDate()
   recalculateEstimatedCompletionDate()
   getProductionRunName()
   setProductionRunName()
   getDescription()
   setDescription()
   getCurrentStatus()
   getProductionRunComponents()
   getProductionRunRoutingTasks()
   getLastProductionRunRoutingTask()
   clearRoutingTasksList()
   getEstimatedTaskTime()
   isUpdateCompletionDate()
   All the names of methods are verbs, and with the first letter of each addition work capitalized.

6. All of the class variables are as follow:
   productionRun
   productionRunProduct
   productionProduced
   quantity
   estimatedStartDate
   estimatedCompletionDate

productionRunName
description
currentStatus
productionRunRoutingTasks
dispatcher
There is no class variable with underscore, but all of the variables are lowercase first letter, and others with first letter capitalized.

7. There are two constant:
   But ”module” and ”resource” should be modified to ”MODULE”

```
public static final String module = ProductionRun.class.getName();
```

```
public static final String resource = ”ManufacturingUiLabels”;
```

   and ”RESOURCE”.

3.1.2 Indention

For all indention, in the file ProductionRun.java, it adopts the convention of four space and done so consistently.
For indention, there is no tab used.

3.1.3 Braces

Bracing style adopted for entire class is Kernighan and Ritchie style.
For all body of all if-else,while,do-while,try-catch and for, the curly braces are used also for only one statement.

3.1.4 File Organization

1. In this file, for all of the sections, there is a blank line to separate from each others.

2. In this file, there is few line exceed 80 characters.

3. In this file, there is no line exceed 120 characters.

3.1.5 Wrapping Line

Every expressions in the ProducationRun.java file fit on a single line, so the convention is valid.

3.1.6 Comments

In this file, all of the comments are used to adequately explain what the class, interface, methods, and blocks of code are doing. What is more, there are also some comments in the method, in order to explain the detail.

3.1.7 Java Source Files

1. In this file, contains only one single public class.

2. In this file, this public class is this first class in the file.

3. There is no external program interface.

4. The javadoc is completed.

3.1.8 Package and Import Statements

The package statements are in the first non-comment statement.
And the Import statements follow with the package statements.

3.1.9 Class and Interface Declarations

The class declarations are in the correct order. And there is no interface. Which is:

1. class documentation comment.
2. class statement.
3. class (static) variable.
4. instance variable.
5. constructors.
6. methods.

The methods are grouped by the functionality.

1. All variable and class members are declared with correct type and the right visibility.
2. All variables are being used only in the scope where they are declared.
3. We do not have a constructor with an empty parameter, so when declaring there is no default constructor called. But when the class ProductionRun is initialized, there is a constructor that can be called.
4. All object references are initialized before used.
5. Several variable attributes have not been initialized explicitly. They may assume a standard value in phase of computation.
6. Almost all declarations appear at the beginning of block, except some are declared after some instructions.

3.1.10 Method Calls

All parameters are presented in the correct order.
We have found two pairs of method that have same name: getEstimatedTaskTime and recalculateEstimatedCompletionDate, but each of them refers to the same functionality.
All method return type is correct.

3.1.11 Array

There are no problem with off-by-one error or out-of-bounds, we manager the array using iterator instead of index.

3.1.12 Object Comparison

In class is always used == to compare a object with NULL, and just one use equals (line 85).

3.1.13 Output Format

The class return always the desired output.
The error message is managed in the classes of exception, so from this class we can not argue on the comprehensiveness.

3.1.14 Computation, Comparisons and Assignments

In this java class we do not have long and complex arithmetic expressions. And there is not special arithmetic expression to be taken with particular attention(like division), therefore, there are no operator precedence problems.
Other operators are also in correct form.
The code does not contain any explicit and implicit type conversions.

3.1.15 Exceptions

For every try statement there are at least one catch statement that take care of exceptions.

3.1.16 Flow of Control

In the class there are not any switch statement. And for loops, they are correct.

3.1.17 Files

This class does not manage the files.

3.2 ViewerServletRequest

3.2.1 Naming Conventions

1. The name for class ViewerServletRequest.java, and all name of its attributes, method and constant are meaningful.

2. There is only one one-character variable, and it is used as parameter of catch statement, therefore it is "throwaway" variable.

3. The class name is composed with three nouns, initial letter of each word is capitalized.

4. We have not defined any interface within the class.

5. There is only one method in class: getParameter(), it contains a verb, also, every addition word begin with capitalized letter.

6. We do not have attributes beginning with an underscore, whatever, the initial word is lowercase, and first letter of each others is capitalized.

7. In class has a constant is written in lowercase:

```
public final static String module = ViewerServletRequest.class.getName();
```

   Where 'module' should be written using all uppercase.

3.2.2 Indention

For all indentation, we adopt the convention of four space. There are not tab used to indent.

3.2.3 Braces

Bracing style adopted for entire class is "Kernighan and Ritchie" style. For all body of all if-else,while,do-while,try-catch and for, the curly braces are used also for only one statement.

3.2.4 File Organization

1. For each section there is a blank line to separate from others.

2. There only few line exceed 80 character (38,48,54 and 59). Neither of them exceed 120 characters.

3.2.5 Wrapping Lines

Every expression in the class fit on a single line, so the convention is valid.

3.2.6 Comments

The class is completely lack of comment.

3.2.7 Java Source Files

There is external program interface, HTTPServletRequest.

3.2.8 Package and Import Statements

The package statements are in the first non-comment statement. And the Import statements follow with the package statements.

3.2.9 Class and Interface Declarations

No problem.

3.2.10 Initialization and Declaration

1. All variables and class members are declared with correct type and the right visibility.
2. All variables are declared in the proper scope.
3. There is no call for the constructor.
4. There is no object references which is used.
5. All variables are initialized where they are declared.
6. All declarations appear at the beginning of block, some exceptions are declared after some instructions.

3.2.11 Method Calls

1. All of the parameters are presented in the correct order.
2. There is only one method: getParameter(String name), and it is been call correctly.
3. All method return type is correct

3.2.12 Arrays

There is no array used in this file.

3.2.13 Object Comparison

There are only two comparison in this ViewerServletRequest.java file, and they are used correctly.

3.2.14 Output Format

The class return always the desired output. The error message is managed in the classes of exception, so from this class we can not argue on the comprehensiveness.

3.2.15 Computation, Comparisons and Assignments

1. In this ViewerServletRequest.java file, we do not have long and complex arithmetic expressions.
2. All of the comparison and Boolean operators are correct.
3. For the throw-catch expression, the error condition is actually legitimate
4. The code does not contain any explicit and implicit type conversions.

3.2.16 Exceptions

1. The relevant exception is caught.

```
try {
reportFileUrl = FlexibleLocation.resolveLocation(reportParam, loader);
} catch (MalformedURLException e) {
Debug.logError(e, module);
}
if (reportFileUrl == null) {
throw new IllegalArgumentException("Could not resolve location to URL: "
+ reportParam);
}
```

The function in this code block tries to locate the file url. In case of wrong url and no file found, an exception is raised.

3.2.17 Flow of Control

No **switch** and **loop** in this file.

3.2.18 Files

This java class does not deal with file operations.

4 Work Time

- 21/01/2017 ZHOU YINAN 1h document structure
- 02/02/2017 ZHOU YINAN 3h function role and checklist
- 02/02/2017 ZHAN YUAN 4.5h checklist
- 04/02/2017 ZHAO KAIXIN 6h checklist