SCRUM SERVICES

This class belongs to the Special Purpose stack which comprises several components and applications that extend the base functionalities offered by Apache OFBiz.

In particular it is situated in the scrum package that, as written in the documentation, enables organisations to manage their product backlog and agile development projects offering the following features:

* **Resource assignment;**
* **Product backlog;**
* **Sprint management;**
* **Version management.**

**With agile software development, we refer to a group of software development methodologies based on iterative development in which requirement and solutions evolve through collaboration between self-organizing cross-functional teams.**

Scrum is a subset of Agile that is often used to manage complex software and product development, using iterative and incremental practices. In particular, work is confined to a regular work cycle, known as a sprint or iteration.

In Scrum, each sprint is required to deliver a potentially shippable product increment. This means that at the end of each sprint, the team has produced a coded, tested and usable piece of software and a sprint review meeting is held in which the Scrum team shows what they accomplished during the sprint.

The ScrumServices class is composed by 4 static method that offer important functionalities for managing Scrum. In particular 3 of them are used for managing scrum revision:

* **public** **static** Map<String, Object> viewScrumRevision(DispatchContext ctx, Map<String, ? **extends** Object> context)
* **public** **static** Map<String, Object> retrieveMissingScrumRevision(DispatchContext ctx, Map<String, ? **extends** Object> context)
* **public** **static** Map<String, Object> removeDuplicateScrumRevision(DispatchContext ctx, Map<String, ? **extends** Object> context)

Then there is the last method which manages communication events, relative to specific products.

**public** **static** Map<String, Object> linkToProduct(DispatchContext ctx, Map<String, ? **extends** Object> context)

In the following paragraph these method will be analysed more in detail.

**ViewScrumRevision**

The function of this method is clearly explained in the Javadoc: *Use for view Scrum Revision.*

In particular it takes from the context (in wich there are the input parameter) the revision and repository names as Strings:

String revision = (String) context.get("revision");

String repository = (String) context.get("repository");

Then it creates a process with the command to call the “log” relative to the revision:

String logCommand = "svn log -r" + revision + " " + repository;

Process logProcess = Runtime.getRuntime().exec(logCommand);

And stamp the result in a buffered reader:

BufferedReader logIn = **new** BufferedReader(**new** InputStreamReader(logProcess.getInputStream()));

With a while cycle save all the result of the command in a string:

**while** ((logline = logIn.readLine()) != **null**) {

logMessage.append(logline).append("\n");

The same commands are executed for the extraction of the “diff” relative to the revision. Then the two resulted strings are putted in the result with the revision and the repository reference:

result.put("revision", revision);

result.put("repository", repository);

result.put("logMessage", logMessage.toString());

result.put("diffMessage", diffMessage.toString());

**RetrieveMissingMevision**

The function of this method is clearly explained in the javadoc: *use for retrieve the missing data of the Revision.*

In particular it takes from the context (in which there are the input parameters) the number of revision:

String latestRevision = (String) context.get("latestRevision");

Integer revision = Integer.parseInt(latestRevision.trim());

Then it obtains all the information about the user and the task and so it performs the following queries retrieving the missing data of the revision:

List <GenericValue> workeffContentList = EntityQuery.use(delegator). from("WorkEffortAndContentDataResource").where("contentName",version.trim() ,"drObjectInfo", revisionLink.trim()).queryList();

List<GenericValue> workEffortList = EntityQuery.use(delegator) .from("WorkEffort").where(exprsAnd).queryList();

At the end it calls the runSync method of the dispatcher in order to update the scrum revisions passing as a parameter a Map in which there are all the information necessary.

dispatcher.runSync("updateScrumRevision", inputMap);

**RemoveDuplicateScrumRevision**

The function of this document is immediately understandable by its name and by the javadoc: *use for remove duplicate scrum revision.*

In particular with the following queries it retrieve the revision:

List<GenericValue> workEffortDataResourceList = EntityQuery.use(delegator).from("WorkEffortAndContentDataResource").where(exprsAnd).queryList();

Then the results are distributed over two data structures:

* keys in which there are univocal values;
* exclusion in which there are the duplicates.

**for** (GenericValue workEffort : workEffortDataResourceList)

{

String drObjectInfo =workEffort.getString ("drObjectInfo");

**if** (keys.contains(drObjectInfo)) {

exclusions.add(workEffort);

} **else** {

keys.add(drObjectInfo);

}

}

At the end all the elements in exclusion are removed.

**LinkToProduct**

For this method there is not the Javadoc that explain its function. Reading the code the methods seams to manage the communication events relative to a product. The product is contained in the context as the communication event.

First extract from the context the communication event ID, and so the communication event itself with a query:

String communicationEventId = (String) context.get("communicationEventId");

GenericValue communicationEvent = EntityQuery.use(delegator).from("CommunicationEvent").where( "communicationEventId", communicationEventId).queryOne();

Then extract the subject of the communication event, and from this the ID of the product at issue. The id is localized and extracted like a substring starting from after the “PD#” characters until the last digit:

String subject = communicationEvent.getString("subject");

**int** pdLocation = subject.indexOf("PD#");

**if** (pdLocation > 0) {

**int** nonDigitLocation = pdLocation + 3;

**while** (nonDigitLocation < subject.length() && Character.isDigit(subject.charAt(nonDigitLocation))) {

nonDigitLocation++;

}

String productId = subject.substring(pdLocation + 3, nonDigitLocation);

}

Then, after has extracted the product, create the CommunicationEventProduct mapping with the delegator:

GenericValue communicationEventProduct = delegator.makeValue( "CommunicationEventProduct", UtilMisc.toMap("productId", productId, "communicationEventId", communicationEventId));

communicationEventProduct.create();

Then extract the Owner of the product and the User Login:

GenericValue productRoleMap = EntityQuery.use(delegator).from("ProductRole").where("productId",productId, "partyId", communicationEvent.getString("partyIdFrom"), "roleTypeId","PRODUCT\_OWNER").queryFirst();

GenericValue userLogin = (GenericValue) context.get("userLogin");

And uses these to commit the communication:

dispatcher.runSync("setCommunicationEventStatus", UtilMisc.<String, Object>toMap("communicationEventId", communicationEvent.getString("communicationEventId"), "statusId", "COM\_COMPLETE", "userLogin", userLogin));

So the communication events are registered.

*List of issues*

**Class ScrumServices**

[6] su module and resource?

[25] The class or interface declarations shall be in the following order:

(a) class/interface documentation comment;

(b) class or interface statement;

(c) class/interface implementation comment, if necessary;

(d) class (static) variables;

i. first public class variables;

ii. next protected class variables;

iii. next package level (no access modifier);

iv. last private class variables.

(e) instance variables;

i. first public instance variables;

ii. next protected instance variables;

iii. next package level (no access modifier);

iv. last private instance variables.

(f) constructors;

(g) methods.

There is no class declaration.

**viewScrumRevision**

[10] Consistent bracing style is used, either the preferred “Allman" style (brace goes underneath the opening block) or the “Kernighan and Ritchie" style (brace is on the same line of the instruction that opens the new block).

Ii is used the “Kernighan and Ritchie" style.

[12] Blank lines and optional comments are used to separate sections.

No blank lines nor optional comments are used. They can be used in the lines 129, 138 and 145.

[13] Where practical, line length does not exceed 80 characters.

Line length exceed 80 character in lines 130, 134, 139 and 141. Line 130 cannot be changed, instead line 134, 139 and 141 can be divided in 2 or more expressions.

[18] Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

The comment that explain the class is not exhaustive. It is general with the specification of the result. There are no comments for the blocks of code.

[33] Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces ‘{‘ and ‘}’). The exception is a variable can be declared in a for loop.

The lines from 138 to 144 form a conceptual block. They are not surrounded by braces but it is the same a block, and has the declarations at the beginning.

[40] !=null

[44] isNotEmpty?

[51] check that the code id free of any implicit type conversion.

General value

(attribute non usati?)

**LinkToProduct**

[10] Consistent bracing style is used, either the preferred “Allman" style (brace goes underneath the opening block) or the “Kernighan and Ritchie" style (brace is on the same line of the instruction that opens the new block).

Ii is used the “Kernighan and Ritchie" style.

[13] Where practical, line length does not exceed 80 characters.

Many lines exceed the length of 80: 62, 70, 75, 77, 79, 83, 87, 91, 94, 101 and 107. Many of these lines can be divided in two or more parts.

[14] When line length must exceed 80 characters, it does NOT exceed 120 characters.

Many lines exceed the length of 120: 62 with 151 characters, 77 with 192 characters, 79 with 176 characters, 83 with 205 characters, 87 with 212 characters, 91 with 136 and 101 with 121 characters.

[18] Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

The method function is not explained at all. The comments at the block are not exhaustive to understand the function of the block.

[33] Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces ‘{‘ and ‘}’). The exception is a variable can be declared in a for loop.

Many declarations appear in the code, not only at the beginning of the blocks, as at line 73.

[40] !=null

[44] isNotEmpty?

[51] Check that the code is free of any implicit type conversions.

Often use the class “Object” and the class “GenericValue” for save some result, without an explicit casting.

Profondità di parentesi?