Contents

[**Recording Introduction** 1](#_Toc516647455)

[**Recording Architecture** 1](#_Toc516647456)

[**AOP Executions** 2](#_Toc516647457)

[Materials 2](#_Toc516647458)

[1. Methods Invoked 2](#_Toc516647459)

[2. Methods Excluded 2](#_Toc516647461)

[**Recording logs** 3](#_Toc516647466)

[1 DoAfterLog 3](#_Toc516647467)

[2 Interceptor 3](#_Toc516647468)

[3 WriteFile 3](#_Toc516647469)

[**Workspace setup** 3](#_Toc516647470)

[**AOP Terminology** 4](#_Toc516647471)

[1. Aspect 4](#_Toc516647472)

[2. Advice 4](#_Toc516647473)

[3. JoinPoint 5](#_Toc516647474)

[4. Pointcut 6](#_Toc516647475)

[7. Aspect 6](#_Toc516647476)

[8. Interceptor 6](#_Toc516647477)

[9. AOP Proxy 6](#_Toc516647478)

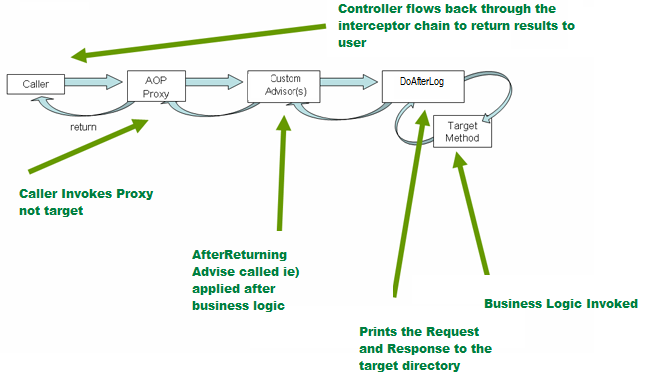
[10. Weaving 6](#_Toc516647479)

## **Recording Introduction**

Recording is used for listening to the incoming request / response for a method in execution and captures / prints the request / response xmls . This is implemented using aspect oriented programming.

  Aspect is a module that encapsulates a crosscutting concern, and it is composed of pointcuts and advice bodies. The interception of an aspect is performed in a join point (a point in the execution flow), and defined inside a pointcut (a set of join points). Whenever the application execution reaches one pointcut, an advice (namely a callback) associated with it is executed and records and prints the request and response logs

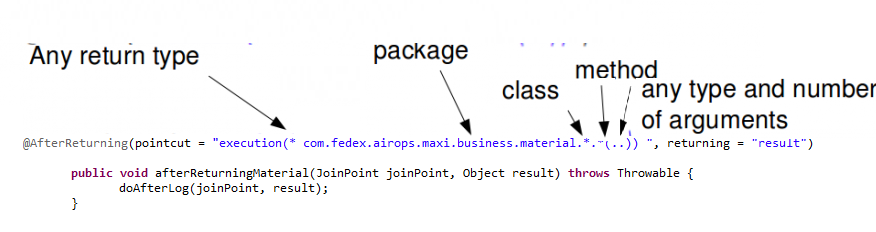
## **2.** **Recording Architecture**



## **3.AOP Executions**

**Materials**

1. **Methods Invoked**



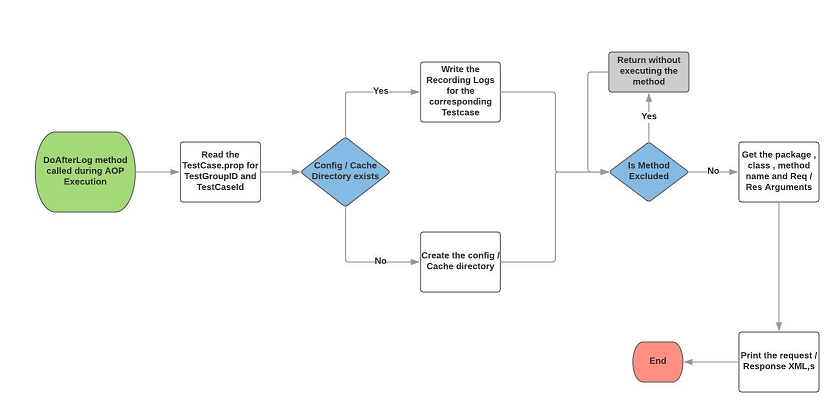
1. **Methods Excluded**

EJB related methods are excluded as the Tomcat server does not support EJB Container

## **Recording logs**

1. **DoAfterLog**

* Read the TestCase.properties from the tmp directory for the TestGroupId and TestCaseId
* Create the config directory and recording.log inside config folder if it doesn’t already exist.
* Create the cache directory under the config folder if it doent not already exist.
* Get the package name, class name, method name and arguments as part of the execution being called.
* Check for the class or methods excluded and exclude from execution.
* Get the request and print / append the request to recording.log
* Get the response and print / append the response object to recording.log
* For cache related request / response , print / append the file to cache.log



1. **Interceptor**

Converts the object to ByteArrayOutputStream

1. **WriteFile**

Writes the ByteArrayOutputStream to file.

## **5. Class Diagram**

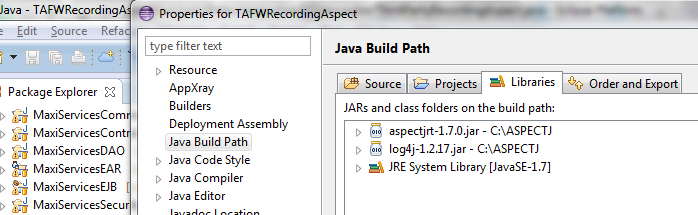
[..\..\..\..\..\Workspace\recording\TAFWRecordingAspect\src\com\fedex\airops\mach\interceptor\MateriaLRecording.jpg](../../../../../Workspace/recording/TAFWRecordingAspect/src/com/fedex/airops/mach/interceptor/MateriaLRecording.jpg)

## **5.** **Workspace setup**

**Repository Location:**

<https://wtc-5271240-w24.corp.ds.fedex.com/svn/tcoe/Onsite/MaterialsRecordingAndExecution/TAFWRecordingAspect>

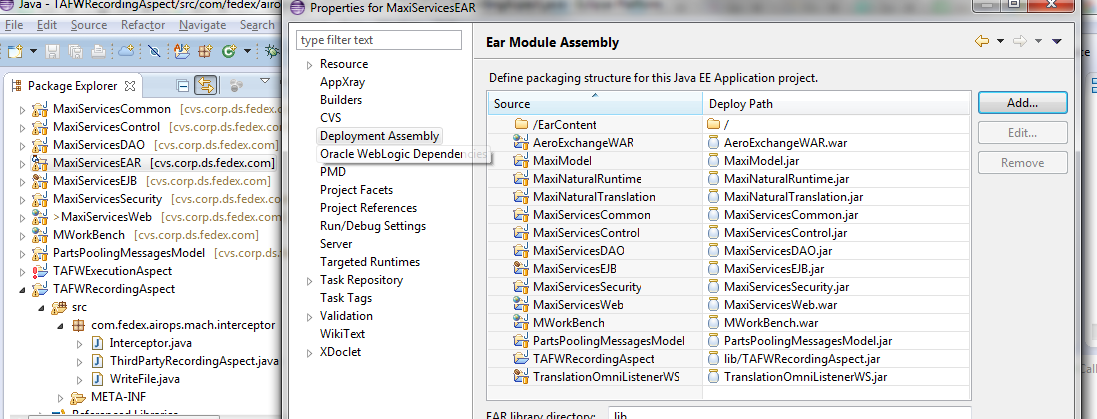
Apectj related jars need to be part of the build path.



**Java Version:** 1.7.0\_95

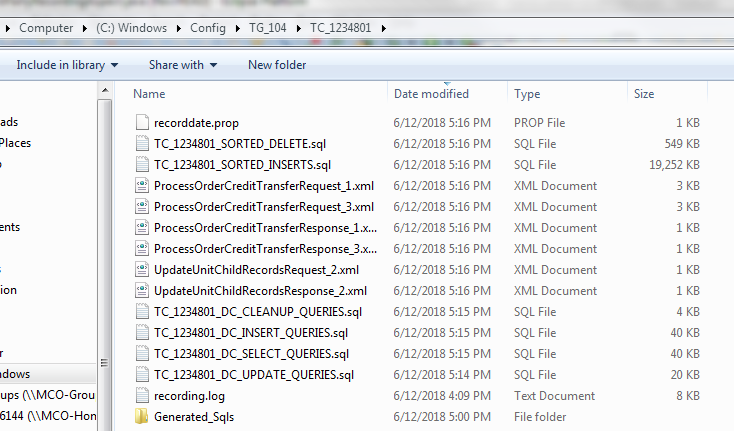
**Deployment:**

Add the TAFWRecordingAspect as part of the deployment assembly of MaxioServicesEAR.



## **6. Testing the Recording**

The output will be generated in the config directory of the corresponding test case.



## **7. AOP Terminology**

1. **Aspect**
2. **Advice**
3. **JoinPoint**
4. **Pointcut**
5. **Introduction**
6. **Target**
7. **Proxy**
8. **Weaving**
9. **Adviser**

### 1. Aspect

* An aspect represents the cross-cutting functionality name, just name only.
* Aspect denotes only the cross-cutting functionality name not its implementation

### 2. Advice

* Advice is the implementation of Aspect.
* An Advice provides the code for implementation of the service.

#### 1.Before Advice

* services will be applied before business logic
* MethodBeforeAdviceinterface extends the BeforeAdvice interface.
* If we implement MethodBeforeAdvice interface, we need to override before()
* before() method are executed at before business logic

#### 2.After Advice

* services will be applied After business logic
* AfterReturningAdviceinterface extends the AfterAdvice interface.
* we need to override afterReturning() method

#### 3.Around Advice

* It is the combination of both Before and After Advice.
* MethodInterceptorinterface extends the Interceptor interface.
* In Around Advice, we implement Before and After Advice in a single method called invoke(), in order to separate Before an After services to execute business logic,

#### 4.Throws Advice

* Services will be applied when business logic methods throws an exception.
* ThrowsAdviceinterface also extends the AfterAdvice
* we should implement afterThrowing() method

### 3. JoinPoint

While creating the business logic of the method the additional services are needed to be injected at different places or points, we call such points as joinpoints.  At a joinpoint a new service will be added into the normal flow of a business method.

While executing the business method, the services are required at the following 3 places, we call them as JoinPoints.

* Before business logic of the method starts
* After business logic of the method got completed
* If business logic throws an exception at run time

### 4. Pointcut

A pointcut defines what advices are required at what join points. In above diagram Authentication Advice, Logging Advice, Transaction Advice are required after withdraw logic & after balance logic. So this point is known as PointCut.

### 7. Aspect

It is a class that contains advices, joinpoints etc.

### 8. Interceptor

It is an aspect that contains only one advice.

### 9. AOP Proxy

It is used to implement aspect contracts, created by AOP framework. It will be a JDK dynamic proxy or CGLIB proxy in spring framework.

### 10. Weaving

It is the process of linking aspect with other application types or objects to create an advised object. Weaving can be done at compile time, load time or runtime. Spring AOP performs weaving at runtime.