T store Payment Open API Java Integration Guide

- 본 문서에 관하여
  T store 의 Payment Open API 를 연동하는 가맹사(Merchant)에 배포되는 문서로 함께 배포되는 Open API Java Library 에 대한 사용 방법을 기술하고 있다.
  가맹사의 개발자가 본 문서의 적정 참고 대상자이며, 본 문서의 참고에 앞서 Open API 의 전체 기능 및 명세와 관련된 부분을 기술하고 있는 'T store Payment Open API Guide' 문서의 우선 확인을 권장한다.

- 사전 확인 사항
  Open API 는 개발환경(Sandbox)과 상용환경(Live)으로 분리되어 운영되어 각 환경별로 아래의 내용의 확인이 선행되어야 한다.
  - 상품 정상 등록 확인
    자동결제 상품에 대해서는 T store 개발자센터가 아닌 운영 담당자를 통한 별도의 등록 절차가 필요
  - 방화벽(ACL) 설정
    가맹사의 서버의 in/out bound 설정 및 Open API 서버의 in/out bound 설정에 대한 유효성 확인
  - OAuth Client Account 확인
    개발 및 상용 환경에 대응되는 client_id 와 secret 코드 확인 및 계정의 활성(Activation) 상태 확인

- Service 요청 Library 초기화
  해당 Library 는 외부 오픈 소스를 참고하고 있으므로, 아래의 라이브러리를 추가를 필요로 한다.

  아래는 해당 Library 설정의 위한 pom.xml 파일 설정이다.

```
<!-- Json Mapper -->
<dependency>
      <groupId>org.codehaus.jackson</groupId>
      <artifactId>jackson-core-asl</artifactId>
      <version>1.9.12</version>
</dependency>
```

```
<dependency>
	<groupId>org.codehaus.jackson</groupId>
	<artifactId>jackson-mapper-asl</artifactId>
	<version>1.9.12</version>
</dependency>

<!-- http client -->
<dependency>
	<groupId>org.apache.httpcomponents</groupId>
	<artifactId>httpclient</artifactId>
	<version>4.3.6</version>
</dependency>

<!-- log4j -->
<dependency>
	<groupId>org.slf4j</groupId>
	<artifactId>slf4j-log4j12</artifactId>
	<version>1.7.2</version>
</dependency>
```

제공되는 Open API 서비스 Library 는 Sandbox 와 Live(상용) 환경 모두에서 사용할 수 있으며, 각 환경별로 아래와 같이 초기화가 필요하다. 또한 Library 자체적으로 Log4j 설정 적용이 가능하므로, 각 시스템 환경에 맞추어 Library 자체의 Log4j 파일 설정이 필요하다.

아래는 Library Instance 를 취득하기 위한 Sample 소스코드 이다.

```
// Get Open API service Instance
AbstractManagerFactory factory =
  ManagerProducer.getFactory(Environment.LIVE, "/temp/log/log4j.prop");
OpenApiManager service = factory.getOpenApiManager();

// to do sth with openApiManager
```

아래는 Library 에 적용되는 log4j property 파일의 Sample 이다.

```
# Log4j Setting file
log4j.rootLogger=INFO, console, file

#Log to file FILE
log4j.appender.file=org.apache.log4j.DailyRollingFileAppender
log4j.appender.file.File=/sample-web/logfile.log
log4j.appender.file.DatePattern='.'yyyy-MM-dd
log4j.appender.file.append=true
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-
3p %c %3x -%m%n
```

```
# Console log
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yyyy-MM-dd
HH:mm:ss} %-3p %l -%m%n
```

아래는 해당 기능과 관련된 연동 개발시의 권고 및 참고 사항이다.

- 연동 환경이 Sandbox(개발)와 Live(상용) 나눠지는 만큼, 연동 코드상에 tomcat configuration 이나 System Property 상에서 환경 정보를 읽어서 적절한 값을 세팅할 수 있도록 개발 및 배포환경의 설정을 권장한다.
- 연동 개발이나 서비스 운용상 문제시 빠른 원인 파악을 위하여 로그 확인이 필요한 만큼, 가급적 위와 같은 로그 설정을 사용하여 Library 로그 파일이 정해진 위치에 생성될 수 있도록 한다.
- 통상적으로 해당 로그는 3 개월 정도의 보관을 권장한다.

- OAuth AccessToken 발행하기
  Open API 에서 제공하는 모든 API 를 호출하기 위해서는 가맹사 서버가 API 의 호출시 유효한 AccessToken 을 보유하고 있는 상태여야 한다.

  아래는 accessToken 을 신규로 발급받기 위한 Sample 소스코드 이다.

```
// Sample Credentials for Sandbox Environment
String clientId =
  "dCzM2NuHr1dOBQVqSpJO2VKzqguuuwgrddcMrXgFEyod0wDBNfm1YfCRKRsZ5CQ3";
String clientSecret =
  "6XV9AWDxsarQZpF8cF+425zQg96NYhziAB7pdsD7/5E=";

// Get New AccessToken
OAuthClientInfo oauthClientInfo =
  new OAuthClientInfo(clientId, clientSecret, "client_credentials");

OAuthManager oAuthManager = factory.getOAuthManager(oauthClientInfo);

OAuthAccessToken accessTokenObj = oAuthManager.createAccessToken();
String accessToken = accessTokenObj.getAccessToken();

System.out.println("AccessToken > " + accessToken);
```

[실행 결과]

```
AccessToken > ec13e388f5236cf8cd7cd7604a3665ec
```

아래는 해당 기능과 관련된 연동 개발시의 권고 및 참고 사항이다.

- AccessToken 과 Expired (해당 토큰 만료시간)값은 가맹사의 서버에서 저장 관리되어야 하는 값으로, API 호출전 Expired 유무를 판단하여 신규로 AccessToken 을 발행 여부를 결정하는 로직이 구현되어야 한다
- AccessToken 과 Expired 값은 저장시 암호화하여 저장을 권장한다.
- 서로 다른 유효 AccessToken 의 계속적인 발행은 가능하나 가급적 만료시 신규 AccessToken 의 발행을 권장한다.

- 대량 결제 요청

대량 결제 API 를 호출하기 위해서는 우선 유효한 AccessToken 을 발급받은 상태여야 하며, 대량 파일 결제 요청후 발급된 대량 결제 Job ID 로 향후 결제 결과 파일을 조회할 수 있다.

아래는 대량 결제 요청을 위한 Sample 소스코드이다.

```
// Create File Payment
OpenApiManager service = factory.getOpenApiManager();

FilePaymentHeader filePaymentHeader = new FilePaymentHeader();
filePaymentHeader.setVerBulkPay("1");
filePaymentHeader.setBid("skplanet");
filePaymentHeader.setNotiUrl("https://yourdomain.com/noti_listener");
filePaymentHeader.setCntTotalTrans(1);
filePaymentHeader.setPriority("Instant");

File file = new File("res/sample.pay");

FilePaymentResult result =
     service.createFilePayment(filePaymentHeader, file, accessToken);

System.out.println(result.getJobId() + "|" + result.getResultCode() +
          "|" + result.getResultMsg() + "|" + result.getWaitingJobs());
```

[실행 결과]

```
777|0000|Success|0
```

아래는 해당 기능과 관련된 연동 개발시의 권고 및 참고 사항이다.

- 성공시 발생되는 JobID 는 향후 결제 결과 조회를 위하여, 가맹사의 서버에서 반드시 저장관리 되어야 하는 값이며, 향후 운영을 위하여 요청시간, 결과, 요청의 header 정보 등의 정보도 추가 저장 관리를 권장한다.
- 일반적으로 가맹사의 서버의 Batch 모듈에서 해당 모듈이 매일 특정 시간이 호출되는 형태로 운영되며, 설정상의 실수로 같은 요청이 중복으로 발생할 수 있도록 각별한 유의가 필요하다.
- 요청 파일의 Size 나 내부 요청 항목의 수의 제한은 없으나, 결제 요청의 성격이나 사용자 그룹에 따라서 요청 파일을 분할하여 요청/처리할 수 있다.
- NotiUrl 은 null 로 세팅되어도 결제는 진행되나 서비스 배포시 반드시 NotiUrl 설정이 필요하다.
- FilePaymentHeader 설정시 CntTotalTrans 값은 대량 결제요청을 하는 파일의 결제 건수와 반드시 일치하여야 한다. 이를 테면 결제 요청 파일에 10 개의 결제 건이 있다면 CntTotalTrans 도 10 으로 설정해야 한다. 만약 FilePaymentHeader 값과 실제 요청하는 파일의 요청 개수가 맞지 않는 경우 결제가 누락되거나 응답을 받지 못할 수 있으니 반드시 확인한다.

- 대량 파일 결제 결과 파일 다운로드
  대량 결제 처리가 완료되면 Open API PNS(Payment Notification System)로 대량 결제 처리 완료 Notification 메시지가 가맹사의 서버로 전달되고, 해당 메시지의 수신 이후 시점부터 처리가 결제 처리가 결과를 확인 할 수 있는 파일을 다운로드 받을 수 있다.

  아래는 대량 결제 요청의 처리결과 파일을 취득하기 위한 Sample 소스코드이다.

```
// Get File Payment Job Result
File resFile = new File("result/resFileinMerchant.txt");

service.getFilePaymentJobStatus(jobId, resFile, accessToken);
System.out.println("Path >>> " + resFile.getAbsolutePath());
```

  [실행 결과]

```
Path >>> /sample-web/result/resFileinMerchant.txt
```

  아래는 해당 기능과 관련된 연동 개발시의 권고 및 참고 사항이다.

- 결과 파일은 대량의 결제건에 대한 정보가 저장되어 있으므로, 반드시 라인 단위로 read 하여 처리를 권장한다. (대량의 파일 로딩시 JVM heap memory 부족으로 인한 오류 가능성에 대한 대응)
- 결제 결과를 가맹사 서버(DB 등)에 이력을 반영하기전, 실제 요청 파일에 대해서 결과상 누락된 부분이 없는지, 요청에 대한 유효한 결과를 포함하고 있는지 확인하는 로직의 적용을 권고한다.
- 만약 처리완료 Notification 메시지가 서버로 전달되기 전에 조회를 하는 경우 올바른 결과값을 받을 수 없다. 따라서 반드시 결제 처리 완료 Notification 메시지를 받은 후 처리 결과를 요청하여 이용한다.

- 개별 결제건 상세 조회
  대량 결제 이후 필요에 따라서 개별 결제 Transaction 건에 세부 내역의 조회가 가능하다.

  아래는 개별 결제 내역 상세 조회를 위한 Sample 소스코드이다.

```
// Get Payment Transaction Details
TransactionDetail txDetail =

service.getPaymentTransactionDetail("TS003_20151108152453940395925465529 ", accessToken);
System.out.println(txDetail.getResultCode() + "|" +
   txDetail.getResultMsg() + "|" + txDetail.getPayer().getAuthKey());
```

  [실행 결과]

```
0000|성공|9843FA6B77E5C8F8DFCD17B2C356373CA2FCF0531D9967E8B61836F364996505
```

  아래는 해당 기능과 관련된 연동 개발시의 권고 및 참고 사항이다.

  - 본 API 는 결제에 대한 검증(Payment Verification), 결제건에 대한 취소시 상태 확인, 가맹사 Admin 에서 직업 조회가 필요한 경우를 제외한 용도로의 사용은 권장하지 않는다.
  - 결제된 아이템이 사용되는 시점마다 해당 API 를 호출하는 용도 등으로 과다한 요청을 발생시킬 경우 OAuth 에 의해서 사용이 제한될 수 있다.

- 개별 결제건 취소
  개별 결제 Transaction 에 대한 취소가 가능하다.

아래는 개별 결제 내역 상세 조회를 위한 Sample 소스코드이다.

```
// Cancel Payment Transaction
CancelRequest cancelRequest = new CancelRequest();
RefundTransactionRequest refundTxReq = new RefundTransactionRequest();
refundTxReq.setAmount(1000);
refundTxReq.setTid("tx_my00001111");
cancelRequest.setRefundTransactionRequest(refundTxReq);

CancelResponse cancelRes =
   service.cancelPaymentTransaction(cancelRequest, accessToken);

System.out.println(">>>" + cancelRes.getResultCode() + "|" +
   cancelRes.getResultMsg());
```

[실행 결과]

```
0000|Success
```

아래는 해당 기능과 관련된 연동 개발시의 권고 및 참고 사항이다.

- 결제 취소는 요청시 Parameter 값이 정확히 일치해야만 정상적으로 처리된다. 반드시 해당 호출전 가맹사 서버의 Transaction 내용과 개별 결제 조회를 통하여 Parameter 값의 정확성 여부의 확인이 필요하다.
- 결제 취소가 일시적으로 대량으로 발생시 해당기능이 제한될 수 있으며, 운영상의 문제로 대량의 취소가 필요한 경우 운영팀으로 사전 문의가 필요하다.

- 대량 결제 Job Notification 메시지 수신처리
  Open API 시스템에서는 대량 결제 Job 이 완료된 직후 가맹사의 서버로 완료 Notification 메시지가 전달된다. 메시지의 Handling 을 위해서 가맹사의 서버는 별도의 Notification Listen API 의 개발이 필요하다.

아래는 해당 API 를 Spring 기반으로 개발하였을때의 Sample Controller 의 소스코드이다.

```
@RequestMapping(value = "/noti_listener", method = RequestMethod.POST)
@ResponseBody
public String notiListener(@RequestBody NotiReceive notificationResult)
{
      // Store Notification Data to DB
      tempDao.insertNotificationMessage(notificationResult);
```

```
        NotiVerifyResult notiResult = new NotiVerifyResult();
        notiResult.setResultCode("0000");
        notiResult.setResultMsg("SUCCESS");

        return convert(notiResult);;
}
```

아래는 해당 기능과 관련된 연동 개발시의 권고 사항이다.

- Open API 의 PNS 서버(pns.payplanet.co.kr)는 Notification Message 를 전송후 10 초간 가맹사 서버로 부터 응답(read) 대기를 하게된다. 10 초 내에 정상적인 응답을 read 하지 못한 경우 timedout 처리하게 되며, Notification Message 는 일정 시간 경과후 재전송하도록 되어있다.
- [필수사항] 메시지 수신후 응답 사이 Biz 로직 처리에 대량의 시간이 소비되거나, 트래픽 및 서버 상태에 따라서 처리 시간이 가변적인(대량으로 늘어 날 수 있는) 로직이 수행되어서는 안된다.

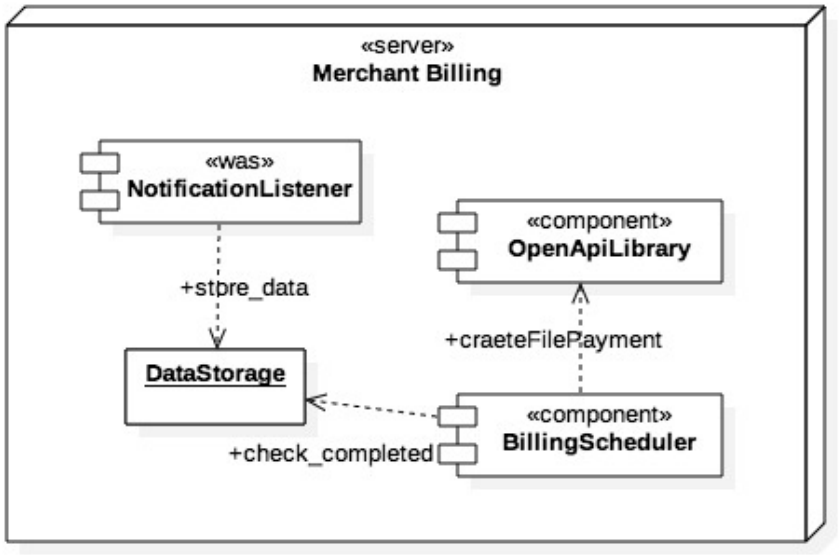
- 대량 결제 Job Notification 메시지 검증 처리 (message session close)
  Open API 의 Notification 전송은 송수신측 모두 동일 메시지의 전달을 보장하기 위하여, 송수신 메시지의 검증을 수행하도록 되어 있다. 검증의 의미는 Open API(PNS)에서 전송한 메시지가 맞는지, 수신측(가맹사)으로 정상적인 메시지가 전달되었는지를 확인하기 위한 절차이다.
  Open API(PNS)에서 메시지를 전송하였으나 일정시간(30 분) 이내에 검증 요청이 없을 경우 가변적인 시간 주기로 동일 메시지를 최대 48 동안 재전송(retry)하게 된다.

** 현재 Library 상 공식 미지원으로 개발사에 직접 개발 필요 **

기타 연동 참고

- 가맹사(Merchant) 시스템 권장 구조
  - Notification Message 를 수신하기 위해서는 별도의 HTTP Server 모듈이 필요하다. Tomcat 과 같은 WAS 도 문제없으나 가벼운 데이터 처리에 해당되므로 경량 웹서버로 구성하여도 무관하나 반드시 HTTPS 통신을 지원해야 된다.

- 앞서 언급한 바와 같이 Notification Listening 로직은 최대한 간단한 로직으로 구성되어야 한다.
- 동일한 방식으로 Notification 은 향후 개별 결제 성공/실패, 결제 취소 등으로 확대될 수 있다.
- File Payment 요청시 요청에 대한 이력을 반드시 DB 나 File System 등에 가급적 모든 정보(Context)를 저장하고 Job 에 대한 완료 및 Notification Message 검증 여부를 별도의 State Column 을 두어 관리를 권장한다. 해당 Column 에는 아래와 같은 상태값들이 사용될 수 있다.
    - Created (Job 이 최초 생성된 상태)
    - Processing
    - Completed
    - VerifiedCompled (Notification 메시지에 대한 검증까지 완료되어, 파일을 다운로드 할 수 있는 상태)
    - Failed
- 가맹사에서 월 정기결제와 같은 형태로 Open API 를 연동 중이라면, 별도의 Batch 모듈이 하루중 특정 시간에 구동되는 형태라면 다음과 같은 형태로 서버 모듈이 구성을 고려해 볼 수 있다.



    - Notification 메시지(Json/Http)를 수신하는 [NotificationListener]는 무중단 서비스 정책이라면 이중화를 고려해 볼 수 있으나, Notification 메시지는 전송 실패시 최대 48 시간 재전송이 이루어 지기 때문에 서비스 중단으로 인한 메시지 유실은 없다.
    - Notification Message 는 DB 나 File System 등 별도의 저장 공간[DataStorage]에 Context 가 저장되고, Job Flow 를 관리하는 [BillingScheduler] 모듈에서 일정 시간 주기로 Context 의 상태를 Polling 하거나, 혹은 File System 의 변화를

Event 로 감지하여 Job 의 상태정보를 취득하여 메시지 검증 및 파일 다운로드 로직을 권장한다.

- 어떠한 경우라도 Notification Message 수신에 대한 응답이 즉각적으로 이루어져야하며, 즉각적인 반응이 필요한 경우 [NotificationListener]와 [BillingScheduler]간 실시간 메시지 전달을 위한 Queue 나 Interface 를 고려할 수 있다.
- 파일 다운 로드후, 가맹사 서버의 별도 File System 에 해당 파일을 저장 후, 해당 CSV 파일을 라인단위로 읽어들여 사용기간 연장, 상품 Activation, 아이템 배송 등의 자체 처리 로직이 동작하는 식의 방식을 권장한다.
- [BillingSceduler] 와 같이 Notification 검증과 결과 파일을 전담하는 로직은 이중화가 필요할 경우 프로세스간 동일 데이터 접근에 따른 별도의 동기화 로직이 필요하다.

- 위의 구조는 권고안으로 개발사의 시스템 구조 및 Business Model 에 따라 다양하게 구성될 수 있다.