4G MDK　インストールガイド

VeriTrans 4G

MDK for Java Installation Guide

ver.1.0.5（2024 年 3 月～）

目次

1. 導入の前に

1.1 本ガイドの内容

本ガイドは、株式会社 DG フィナンシャルテクノロジーが提供する VeriTrans4G を利用するための専用ソフトウェア MDK（Merchant Development Kit）をインターネット店舗などに導入する開発者向けのガイドです。
MDK ファイル構成や設定方法、サンプルプログラム等について記載しています。

1.2 著作権、および問い合わせ先

［お問い合わせ先］
DG フィナンシャルテクノロジー テクニカルサポート
電子メール：tech-support@veritrans.jp

2. 導入準備

2.1 基本導入手順

VeriTrans4G Java 版 MDK の導入にあたり、以下の作業が必要となります。

1. Java 実行環境の確認・準備
2. MDK・ライブラリ組込
3. MDK 設定

2.2 実行環境の確認と準備

導入環境が Java 8 以上であることをご確認下さい。

3. 設定方法

3.1 MDK・関連ライブラリの組込

ダウンロードした Mdk4G-java8-x.x.x.tar.gz を解凍します。(x.x.x は、MDK のバージョンを示します。)

表 1 Java 版 4G MDK ファイル一覧

ディレクトリ／ファイル名		説明
MdkJava/	README.txt	MDK の導入前にお読みください
	CHANGELOG.txt	改版履歴
	tgMdk/ cg-mdk-java8-x.x.x.jar	決済要求ビジネスロジッククラス群
	tgMdk/ cg-mdk-dto-java8 -x.x.x.jar	決済要求／応答情報クラス群
	tgMdk/ 3GPSMDK.properties	MDK 設定ファイル
	tgMdk/ log4j2.xml	ログ出力用設定ファイル
	lib/ commons-codec-1.16.1.jar	依存ライブラリ
	lib/ json-simple-1.1.1.jar	
	lib/ log4j-api-2.23.0.jar	
	lib/ log4j-core-2.23.0.jar	
	lib/ slf4j-api-2.0.12.jar	
	lib/ log4j-slf4j2-impl-2.23.0.jar	
	resources/ cacerts	CA 証明書ストアファイル

全てのファイルを、利用するアプリケーションから参照するクラスパスに配置して下さい。

なお、lib/ 配下のライブラリを既にご使用の場合、必ずしも入れ替えを行う必要はありませんが、ライブラリのバージョンによっては動作に問題が発生する可能性がありますので、十分な動作確認を行っていただきますようお願いします。

3.1.1. MDK のバージョンアップに関する注意点

(1) 古いバージョンのライブラリの削除について

アプリケーションのクラスパスにある古いバージョンの jar ファイルは必ず削除してください。

MDK のバージョンによってファイル名が異なりますので、以下の表を参考にしてください。

VeriTrans3G MDK	tgMdk-x.x.x.jar, tgMdkDto.jar
VeriTrans4G MDK バージョン 2.0.0 未満	cg-mdk-xxx.jar, cg-mdk-dto-xxx.jar
VeriTrans4G MDK バージョン 2.0.0 以上	cg-mdk-java8-xxx.jar, cg-mdk-dto-java8-xxx.jar

その他の古い依存ライブラリの jar も忘れずに削除してください。

(2) ロギング用ライブラリに関する注意点

MDK のログは、SLF4J (https://www.slf4j.org/) を介して出力されますので、MDK を導入するシステム側で

任意のロガーを選択可能です。

なお、バージョン 2.0.0 未満の MDK では、下位互換性に配慮し log4j のバージョン 1 系のライブラリを同梱していましたが、MDK バージョン 2.0.0 からは log4j バージョン 2 系のライブラリを同梱するように変更しておりますのでご注意ください。

3.2 MDK の設定

VeriTrans4G MDK 設定ファイルは、**3GPSMDK.properties** です。設定ファイルには重要な情報が記載されているため、第三者に見えないようにしてください。

（1） 接続先 URL の設定

接続先サーバ（決済サーバー）の URL を設定します。

※デフォルトでの設定のままご利用下さい。

HOST_URL = https://xxx.xxx.xxx.xxx:xxx

（2）接続制御用の設定

（A）接続タイムアウト値(秒)の設定

決済サーバーへの接続が確立できない場合に、接続試行を中断するまでの接続試行開始からの秒数を指定します。

CONNECTION_TIMEOUT = XXX

（B）読み取りタイムアウト値(秒)の設定

決済サーバーからのレスポンスがない場合に、接続を切断するまでのリクエスト開始からの秒数を指定します。

READ_TIMEOUT = XXX

（C）ダミーモードの指定

テスト用にダミー決済を発生させる場合に指定します。

DUMMY_REQUEST = x

「0」：ダミーモード OFF

「1」：ダミーモード ON

（D）MDK 固有エラーモード

VeriTrans4G に無関係な、MDK 固有のエラーテストを実施する場合に指定します。

MDK_ERROR_MODE = x

「0」：MDK エラーモード OFF

「1」：MDK エラーモード ON

※エラーモード ON の場合には vResultCode に ”MA99” が返戻されます。

(3)プロキシサーバの設定

(A)プロキシサーバ URL

プロキシサーバを利用する場合、サーバホスト名／IP アドレス、ポート番号を設定して下さい。

```
PROXY_URL = http://xxxx.xxx.xxx.xxx:xxxx
```

(B)プロキシサーバ接続認証用ユーザ ID

プロキシサーバを利用し、かつ接続に認証が必要な場合、プロキシサーバ接続用ユーザ名を設定して下さい。

```
PROXY_USER_ID = xxxxxxx
```

(C)プロキシサーバ接続認証用パスワード

プロキシサーバを利用し、かつ接続に認証が必要な場合、プロキシサーバ接続用パスワードを設定して下さい。

```
PROXY_USER_PW = xxxxxxx
```

(4)マーチャント認証用の設定

(A)マーチャント CCID の設定

ベリトランスより通知の、店舗サイト用 CCID 文字列を指定します。

```
MERCHANT_CCID = xxxxxxxxxxxx
```

(B)マーチャント認証鍵の設定

ベリトランスより通知の、店舗サイト用認証鍵文字列を指定します。

```
MERCHANT_SECRET_KEY = xxxxxxxxxxxxxxx
```

(5)SSL 通信用の設定(オプション設定)

通常は設定せずにご利用ください(デフォルトでは未設定)。決済サーバーとの SSL 暗号化通信でエラーが発生する場合、OS や Java を安全なバージョンに更新したり、本設定で CA 証明書ファイルのパスを明示的に指定することで解決する場合があります。

※ver1.9.0 以前の MDK をご利用の場合は設定が必須になります。

(A)CA(Certificate Authority)証明書の設定

MDK に同梱の CA 証明書ファイルの絶対パスを指定します。

```
SSL_TRUSTSTORE_FILE =
```

(B)CA 証明書アクセスパスワードの設定

CA 証明書のアクセスパスワードを指定します。

（A）にてMDKに同梱の CA 証明書ファイルを指定した際は”changeit”を指定してください

SSL_TRUSTSTORE_PASSWORD =

（6）SSL プロトコルの設定

ベリトランスと SSL 暗号通信を行う際の SSL プロトコルを指定します。

※デフォルトでの設定のままご利用下さい。

SSL_PROTOCOL = xxxxxxx

[ログ設定ファイル]

ログ設定ファイルはご利用のロギングライブラリによって異なりますが、ここでは Log4j を例とします。

Log4j の設定ファイルは **log4j2.xml** です。以下に最小限の指定項目を記載します。

その他の設定については、「log4j (http://logging.apache.org/log4j/)」の設定方法を参照し設定することが可能です。

（1）ログ出力レベルの指定

MDK のログ出力レベルを指定します。

指定可能値：「OFF／FATAL／ERROR／WARN／INFO／DEBUG（※）」

※より多くの情報を出力したい場合は DEBUG を設定して下さい。

（2）ログ出力ファイルの指定

ログ出力ファイルのパスを指定します。

アプリケーションからファイル出力可能なディレクトリを指定して下さい。

4. サンプルプログラムの利用

4.1 サンプルプログラム(コマンドラインプログラム)設定・動作確認

✓ 動作確認を行うためには Ant(Apache Ant)をインストールする必要があります。

(1)Mdk4G-Sample-java-x.x.x.tar.gz を解凍

Mdk4G-Sample-java-x.x.x.tar.gz を解凍します。(x.x.x は、サンプルプログラムのバージョンを示します。)

コマンドラインから動作確認が可能なサンプルプログラムは MdkSample-Java/command ディレクトリにあります。

(2)4G MDK 本体の jar ファイル群の配置

command/lib ディレクトリに、MDK のアーカイブに含まれる MDK 本体の jar ファイルおよび依存ライブラリの jar ファイルを配置します。

MDK アーカイブに含まれるファイルの詳細については「表 1 Java 版 4G MDK ファイル一覧」をご参照下さい。

(3)環境設定

command/src ディレクトリに、MDK のアーカイブに含まれる設定ファイルを配置します。「3.2 MDK の設定」を参照し、店舗様のサーバ環境に合わせた設定値を設定して下さい。

- 3GPSMDK.properties とログ出力設定ファイル(Log4j の場合は log4j2.xml)が設定対象です。

MDK アーカイブに含まれるファイルの詳細については「表 1 Java 版 4G MDK ファイル一覧」をご参照下さい。

(4)Java ソースコードの確認、リクエストパラメータの修正

Java ソースコードを確認し、必要に応じてリクエストパラメータを修正します。

クレジットカード決済（与信）のサンプルプログラムを例とします。
（ソースファイル：src/jp/veritrans/tercerog/sample/command/card/CardAuthorizeCommand.java）

以下に、VeriTrans4G へ送信するパラメータの設定箇所を抜粋します。
開発ガイドをご参照のうえ、必要なパラメータを設定してください。

```
//----------------------------------
// テスト用リクエスト電文項目設定
//----------------------------------
// 取引 ID
String orderID = Custom.getOrderID();
// 与信方法(同時売上実施の有無)
String withCapture = "false";
// 支払金額
String amount = Custom.getAmount();
```

```
// 支払方法
String jpo = "10";
// トークン
String token= "0a812412-682c-4dad-8a5d-720caf23bca0";

以下省略 ……
```

(5)プログラムのテスト

command ディレクトリに戻り、プログラムのテストを行います。

① command/build-command.xml をエディタで開き、command ディレクトリのパスを設定します。

```
<property name="command.home" value="command ディレクトリのフルパス" />
```

② 実行環境に Ant のパスを設定し、以下のコマンドを実行すると、各サービスの Ant ターゲット名の一覧が出力されます。

```
C:¥> ant -f build-command.xml help
Buildfile: build-command.xml

help:
     [echo] Usage:
     [echo] ---------------------------------------------------------------------
     [echo]   ant compile                -  to Compile all programs.
     [echo]   ant bank-authorize         -  to Execute Bank Authorize.
     [echo]   ant card-authorize         -  to Execute Card Authorize.
     [echo]   ant card-reauthorize       -  to Execute Card Re-authorize.
     [echo]   ant card-cancel            -  to Execute Card Cancel.
     [echo]   ant card-capture           -  to Execute Card Capture
     [echo]   ant card-retry             -  to Execute Card Retry
     [echo]   ant cvs-authorize          -  to Execute Cvs Authorize.
     [echo]   ant cvs-cancel             -  to Execute Cvs Cancel.
     [echo]   ant em-authorize           -  to Execute e-Money Authorize.
     [echo]   ant em-cancel              -  to Execute e-Money Cancel.
     [echo]   ant em-refund              -  to Execute e-Money Refund.
     [echo]   ant mpi-authorize          -  to Execute Mpi Authorize.
     [echo]   ant mpi-reauthorize        -  to Execute Mpi Re-authorize.
     [echo]   ant saison-cancel          -  to Execute Saison Cancel.
     [echo]   ant search                 -  to Search.
     [echo] ---------------------------------------------------------------------

BUILD SUCCESSFUL
Total time: 1 second
```

③ サンプルの動作を確認します。

build-command.xml に定義されている Ant ターゲットを実行すると、対応するサンプルプログラムがコンパイル後に実行されます。以下は、クレジットカード決済(与信)のサンプルプログラムの実行例です。

```
C:¥> ant -f build-command.xml card-authorize
Buildfile: build-command.xml

compile:

card-authorize:
    [java] *- Card(Authorize) -*
    [java] << REQUEST >>
    [java] [Order ID]: 1269950814479
    [java] [WithCapture]: false
    [java] [Amount]: 100
    [java] [JPO]: 10
    [java] [Token]: 0a812412-682c-4dad-8a5d-720caf23bca0
    [java] << RESPONSE >>
    [java] [Status]: success
    [java] [Message]: 処理が成功しました。
    [java] [Result Code]: A001H00100000000
    [java] [Auth Code]: 1234567
    [java] [Reference Number]: 012345678901

BUILD SUCCESSFUL
Total time: 6 seconds
```

- ✓ コンパイルエラーが発生する場合は、build-command.xml に設定したパスが正しいこと、MDK および依存ライブラリが適切なディレクトリにコピーされていることをご確認ください。
- ✓ プログラムの実行時エラーが発生する場合は、classes ディレクトリにコピーされた 3GPSMDK.properties とログ出力設定ファイルの設定の内容が適切であることをご確認下さい。
- ✓ 実行時のコンソールには、決済サーバーからのレスポンス値が出力されます。通信内容の詳細についてはログファイルをご確認ください。

4.2 サンプルプログラム（Web アプリケーション）設定・動作確認

- ✓ 動作確認を行うためには Ant（Apache Ant）と tomcat 等の WEB アプリケーションサーバ（サーブレットコンテナ）をインストールする必要があります。

（1）Mdk4G-Sample-java-x.x.x.tar.gz を解凍

ダウンロードした Mdk4G-Sample-java-x.x.x.tar.gz を解凍します。

WEB アプリケーションとして実装されているサンプルプログラムは MdkSample-Java/web ディレクトリにあります。

（2）サンプル Web アプリケーションの設定

web/WEB-INF/src ディレクトリに、MDK のアーカイブに含まれている設定ファイルを配置します。「3.2　MDK の設定」を参照し、店舗様のサーバ環境に合わせた設定値を設定して下さい。

- ➢ 3GPSMDK.properties と log4j2.xml が設定対象です。

MDK アーカイブに含まれるファイルの詳細については「表　1　Java 版 4G MDK ファイル一覧」をご参照下さい。

次に、各決済サービスの動作確認を行うために必要な properties ファイルの設定を行います。

■mpi.properties

・本人認証・決済後の戻り URL を指定します。

termUrl=http://貴社ドメイン/web/mpi/SearchExec

■em.properties

・nanaco 決済　決済完了後の戻り URL を指定します。

tcc.completeNoticeUrl=http://貴社ドメイン/web/em/Complete

・nanaco 決済　復旧処理後の戻り URL を指定します。

tcc.reAuthorizeRedirectionUrl=http://貴社ドメイン/web/em/Complete

■upop.properties

・決済後の戻り URL を指定します。（加盟店側の UPOP ゲートウェイからの結果を受け取る URL を設定）

termUrl=http://貴社ドメイン/web/upop/AuthorizeResult

■paypal.properties

・ユーザ認証後の戻り URL を指定します。

return.url=http://貴社ドメイン/web/paypal/AuthorizeConfirm

・ユーザ認証キャンセル時の戻り URL を指定します。

cancel.url=http://貴社ドメイン/web/PaymentMethodSelect

■push.properties

・結果通知データ出力ディレクトリを指定します。

outputDirectory=/tmp/

■saison.properties

・PC、スマートフォン版のマーチャントのリダイレクト先 URL を指定します。

merchant_redirection_uri.PC=http://貴社ドメイン/web/saison/Capture

・フィーチャーフォン版のマーチャントのリダイレクト先 URL を指定します。

merchant_redirection_uri.MB=http://貴社ドメイン/web/saison/mb/Capture

・決済方式を指定します。

PC_OR_SP_settlement_method=PC

■alipay.properties

・決済成功後の戻り URL と決済失敗後の戻り URL を指定します。

（店舗側の Alipay ゲートウェイからの結果を受け取る URL を設定）

successUrl=http://貴社ドメイン/web/alipay/AuthorizeResult

errorUrl=http://貴社ドメイン/web/alipay/AuthorizeResult

■carrier.properties

・決済成功時、決済エラー時、および決済キャンセル時に店舗側サイトに画面遷移を戻すための URL を指定します。

successUrl=http://貴社ドメイン/web/carrier/Result?result=SUCCESS

cancelUrl=http://貴社ドメイン/web/carrier/Result?result=CANCEL

errorUrl=http://貴社ドメイン/web/carrier/Result?result=ERROR

・決済成功時、決済エラー時、および決済キャンセル時に店舗側サイトに画面遷移を戻すための URL を指定します。（フィーチャーフォン版）

successUrl.mb=http://貴社ドメイン/web/carrier/mb/Result?result=SUCCESS

cancelUrl.mb=http://貴社ドメイン/web/carrier/mb/Result?result=CANCEL

errorUrl.mb=http://貴社ドメイン/web/carrier/mb/Result?result=ERROR

・「ダミー取引」時のプッシュ URL を指定します。

pushUrl=http://貴社ドメイン/web/push/carrierPush

・「ダミー取引」時のプッシュ URL を指定します。（フィーチャーフォン版）

pushUrl.mb=http://貴社ドメイン/web/push/carrierPush

■oricosc.properties

・決済完了後、消費者ブラウザに店舗の申込完了画面を表示するための URL を指定します。

（PC 版、スマートフォン版の店舗のリダイレクト先）

merchant_redirection_url.PC　= http://貴社ドメイン/web/oricosc/AuthorizeComplete

・決済完了後、消費者ブラウザに店舗の申込完了画面を表示するための URL を指定します。

（フィーチャーフォン版の店舗のリダイレクト先）

merchant_redirection_url.MB = http://貴社ドメイン/web/oricosc/mb/AuthorizeComplete

■rakuten.properties

・決済成功時および決済エラー時に店舗側サイトに画面遷移を戻すための URL を指定します。

successUrl=http://貴社ドメイン/web/rakuten/Result?result=SUCCESS

errorUrl=http://貴社ドメイン/web/rakuten/Result?result=ERROR

・「ダミー取引」時のプッシュ URL を指定します。

pushUrl=http://貴社ドメイン/web/push/rakutenPush

■recruit.properties

・決済成功時および決済エラー時に店舗側サイトに画面遷移を戻すための URL を指定します。

successUrl=http://貴社ドメイン/web/recruit/Result?result=SUCCESS

errorUrl=http://貴社ドメイン/web/recruit/Result?result=ERROR

・「ダミー取引」時のプッシュ URL を指定します。

pushUrl=http://貴社ドメイン/web/push/recruitPush

■linepay.properties

・決済成功時、決済エラー時、および決済キャンセル時に店舗側サイトに画面遷移を戻すための URL を指定します。

successUrl=http://貴社ドメイン/web/linepay/Result?result=SUCCESS

cancelUrl=http://貴社ドメイン/web/linepay/Result?result=CANCEL

errorUrl=http://貴社ドメイン/web/linepay/Result?result=ERROR

・「ダミー取引」時のプッシュ URL を指定します。

pushUrl=http://貴社ドメイン/web/push/linepayPush

■paynowid.properties

・クレジットカード決済(3D-Secure 認証付き)の本人認証・決済後、戻り URL の設定

（※PayNowID カード情報を利用した場合）

termUrl=http://貴社ドメイン/web/mpi/SearchExec

■bank.properties

・銀行決済完了後の戻り URL を指定します。

termUrl=http://貴社ドメイン/web/bank/Thanks

■token.properties

・クレジットカードを利用した決済時に、店舗サイト毎に発行されたトークン取得用のキーを指定します。

tokenApiKey=店舗サイト毎に発行されたトークン取得用のキー

・クレジットカードを利用した決済時に、トークン取得用の URL を指定します。

※デフォルトでの設定のままご利用下さい。

tokenApiUrl=https://xxx.xxx.xxx.xxx/4gtoken

■paypay.properties

・決済成功時、決済エラー時、および決済キャンセル時に店舗側サイトに画面遷移を戻すための URL を指定します。

successUrl=http://貴社ドメイン/web/paypay/Result?result=SUCCESS

cancelUrl=http://貴社ドメイン/web/paypay/Result?result=CANCEL

errorUrl=http://貴社ドメイン/web/paypay/Result?result=ERROR

・「ダミー取引」時のプッシュ URL を指定します。

pushUrl=http://貴社ドメイン/web/push/paypayPush

■amazonpay.properties

・決済成功時、決済エラー時、および決済キャンセル時に店舗側サイトに画面遷移を戻すための URL を指定します。

successUrl=http://貴社ドメイン/web/amazonpay/Result?result=SUCCESS

cancelUrl=http://貴社ドメイン/web/PaymentMethodSelect

errorUrl=http://貴社ドメイン/web/amazonpay/Result?result=ERROR

(3)サンプル WEB アプリケーションのコンパイル

build-web.xml の以下の行を修正します。

<property name="application.home" value="店舗様アプリの web コンテナのパスを設定"/>

<property name="servlet.api" value="Servlet API を含んだ jar へのフルパスを設定"/>

※ tomcat の場合、ServletAPI の jar は[インストールディレクトリ]/lib/servlet-api.jar にあります。

以下のコマンドでコンパイルを行います。

ant -f build-web.xml

(4)サンプル Web アプリケーションのデプロイ

設定済みの web ディレクトリを Web アプリケーションサーバにデプロイします。

※ tomcat の場合、デフォルトの設定では[インストールディレクトリ]/webapps にコピーすると自動デプロイされます。

(5)サンプルの動作確認

WEB アプリケーションの URL にブラウザからアクセスすると、サンプルのトップページが表示されます。

URL の例:

http://貴社ドメイン:8080/web/PaymentMethodSelect.jsp

トップページから、各決済サービスの画面に遷移し、動作をご確認ください。

4.3 サンプルプログラム（結果通知受信用プログラム）設定・動作確認

結果通知受信用プログラムは、決済サーバーから店舗様へ消費者アクションの結果、または決済サーバーにおける取引のステータスの変化の通知データを受信するサーブレットのサンプルプログラムです。

通知される内容（以後、結果通知）の詳細については、開発ガイドをご参照ください。

（1）Mdk4G-Sample-java-x.x.x.tar.gz を解凍

ダウンロードした Mdk4G-Sample-java-x.x.x.tar.gz を解凍します。

WEB アプリケーションとして実装されているサンプルプログラムは MdkSample-Java/web ディレクトリにあります。

結果通コンパイルおよびデプロイの手順については、4.2 をご参照ください。

（2）結果通知受信用 Java ソースコードを確認

結果通知受信用 Java ソースコードは web/WEB-INF/src/jp/veritrans/tercerog/sample/push に入っています。

サンプルプログラムのため、結果通知データを受信し、ファイル出力を行う単純な実装としています。

（3）結果通知データの出力先ディレクトリの設定

web/WEB-INF/src/push.properties に、結果通知データの出力先ディレクトリを設定してください。

```
outputDirectory=/tmp/          <- 店舗様環境に合わせて設定
```

（4）コンパイルとデプロイを実施

4.2 を参考に、コンパイルとデプロイを行います。

(5)結果通知データ受信用 URL のマッピングルール確認

web/WEB-INF/web.xml に、結果通知データ受信用 URL のマッピングルールが定義されています。店舗様の環境に合わせてご調整ください。

```
<servlet>
    <servlet-name>BankPushServlet</servlet-name>
    <servlet-class>jp.veritrans.tercerog.sample.push.BankPushServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>CarrierPushServlet</servlet-name>
    <servlet-class>jp.veritrans.tercerog.sample.push.CarrierPushServlet</servlet-class>
</servlet>
                              ～ 省略 ～
<servlet>
    <servlet-name>PayNowIdCardCleaningPushServlet</servlet-name>
    <servlet-class>jp.veritrans.tercerog.sample.push.PayNowIdCardCleaningPushServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>PayNowIdRecurringPushServlet</servlet-name>
    <servlet-class>jp.veritrans.tercerog.sample.push.PayNowIdRecurringPushServlet</servlet-class>
</servlet>

                              ～ 省略 ～

<servlet-mapping>
    <servlet-name>BankPushServlet</servlet-name>
    <url-pattern>/push/bankPush</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>CarrierPushServlet</servlet-name>
    <url-pattern>/push/carrierPush</url-pattern>
</servlet-mapping>
                              ～ 省略 ～
<servlet-mapping>
    <servlet-name>PayNowIdCardCleaningPushServlet</servlet-name>
    <url-pattern>/push/paynowidCardCleaningPush</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>PayNowIdRecurringPushServlet</servlet-name>
    <url-pattern>/push/paynowidRecurringPush</url-pattern>
</servlet-mapping>
```

(6)結果通知受信用サンプルプログラムの動作確認

結果通知受信用サンプルプログラムを動作させます。

web ディレクトリ自体は、標準 WEB アプリケーションを想定して作られているため、一般的なアプリケーションサーバの WEB コンテナにデプロイすれば基本的には動作します。

例えば tomcat の場合は、webapps に web ディレクトリをコピーすれば、下記 URL で結果通知受信用サーブレットが動作します。

MPI 用	http://貴社ドメイン/web/push/mpiPush
銀行用:	http://貴社ドメイン/web/push/bankPush
コンビニ用:	http://貴社ドメイン/web/push/cvsPush
電子マネー用:	http://貴社ドメイン/web/push/emPush
PayPal 用:	http://貴社ドメイン/web/push/paypalPush
銀聯用:	http://貴社ドメイン/web/push/upopPush
Alipay 用:	http://貴社ドメイン/web/push/alipayPush
キャリア決済用:	http://貴社ドメイン/web/push/carrierPush
ショッピングクレジット決済用:	http://貴社ドメイン/web/push/oricoscPush
楽天 ID 決済用:	http://貴社ドメイン/web/push/rakutenPush
リクルートかんたん支払い用:	http://貴社ドメイン/web/push/recruitPush
LINE Pay 用:	http://貴社ドメイン/web/push/linepayPush
PayPay 用:	http://貴社ドメイン/web/push/paypayPush
AmazonPay 用:	http://貴社ドメイン/web/push/amazonpayPush
ワンクリック継続課金サービス（洗替機能）用:	http://貴社ドメイン/web/push/paynowidCardCleaningPush
ワンクリック継続課金サービス（継続課金機能）用:	http://貴社ドメイン/web/push/paynowidRecurringPush

※https://も利用可能です。

結果通知データを受信すると、push.properties に設定した場所に CSV 形式のファイルが作成されます。
（結果通知項目の詳細は、VeriTrans4G 開発ガイドを参照して下さい）

- ✓ このサンプルプログラムを利用して、決済サーバーからの通知を受信するテストを実施する場合は、MAP（マーチャント管理ポータル）の「各種設定変更」より通知 URL の設定を行う必要があります。設定方法につきましては、MAP のご利用ガイドをご参照ください。
  - ➢ 一部のサービスについては、MDK のリクエストパラメータに通知 URL を設定することができます。この場合は、MAP から通知 URL の設定を行わなくても通知の受信テストが可能です。テスト対象のサービスが通知 URL をリクエストパラメータで設定できるかどうかは、VeriTrans4G 開発ガイド（サービス毎のインターフェース詳細）にて、リクエストパラメータの記載内容をご確認ください。

5. 付録

5.1 MDK メソッドリスト

MDK のクラス、メソッドの一覧を Javadoc 形式で出力して公開しています。

ダウンロードサイトより「MDK_MethodList_Java」を入手し、「index.html」、「index-all.html」よりご参照ください。

5.2 MDK で複数のマーチャント ID を使い分ける方法

(1)決済要求毎にマーチャント ID を切り替える

MDK は通常、設定ファイル(3GPSMDK.properties)に記述されたマーチャント CCID と SECRET_KEY(以下、認証情報)を利用して署名を算出し、決済要求時に付与して送信します。署名の算出は、処理要求単位(トランザクション単位)で行います。

マーチャント ID の切り替えを行うためには、決済要求の実行前に、MDK が保持しているメモリ上の認証情報を上書きする必要があります。

※MDK は、初回の呼び出し時に設定ファイルから読み込んだ認証情報を、スタティックな情報として保持しています。そのため、設定ファイル内の認証情報を書き換えても、アプリケーションの再起動を行うまで認証情報は反映されません。

認証情報を上書き設定する手順を以下に示します。

1. 利用する認証情報を取得する
2. メモリ上の認証情報を、取得した情報で上書き設定する(*1)
3. トランザクションオブジェクトを生成する
4. トランザクションを実行する

以下の点に注意して実装してください。

- ✓ MDK の仕様上、設定ファイル(3GPSMDK.properties)内のマーチャント CCID と SECRET_KEY には、適当な値(半角英数字)を設定してください。未設定の場合、MDK の初期化時にエラーとなります。
- ✓ 認証情報の上書き設定は、必ずトランザクションオブジェクトを生成する「前に」行ってください。
- ✓ MDK の設定はメモリ上に保持されますので、毎回認証情報を設定してください。スレッドが再利用されるケースでは、前回上書きした値が保持されますので、意図しないマーチャント ID で決済が行われる可能性があります。
- ✓ 本機能をご利用の際は、マーチャント ID が正しく切り替わっているかどうか、十分にテストを実施してください。

(*1)

スレッドスタティックな形で保持している情報が上書きされますが、決済要求の都度、使用する情報を設定する形で実装してください。例えば、設定ファイルには認証情報 A が記述されているので、認証情報 B を使いたいときだけ上書きを行う、という実装では正しく切替ができません。

【サンプルコード】

ここでは、認証情報がプロパティファイル(merchant_info.properties)に保存されているものとします。

merchant_info.properties

```
# properties ファイル読み込みサンプル_抜粋.txt に対応する properties ファイルのサンプルです。

# "merchant_" + merchantUniqueValue + "_ccid"でそのマーチャントの CCID を。
# "merchant_" + merchantUniqueValue + "_secretKey"でそのマーチャントの SecretKey を設定します。
# merchantUniqueValue の値は、CCID と SecretKey でペアになるように記述してください。

# マーチャント１の設定
merchant_1_ccid=A100000000000000000001CC
merchant_1_secretKey=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

# マーチャント２の設定
merchant_2_ccid=A100000000000000000002CC
merchant_2_secretKey=YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY

# マーチャント３の設定
merchant_3_ccid=A100000000000000000003CC
merchant_3_secretKey=ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
```

認証情報の上書き処理の実装例を以下に示します。

```
// マーチャント毎の ccid と secretKey を管理しているプロパティファイルを読み込みます。
ResourceBundle bundle = ResourceBundle.getBundle("merchant_info");

// マーチャントを識別するための文字列を取得します。(サンプルとして、ユニーク値を 1 としています。)
String merchantUniqueValue = "1";

// プロパティファイルから、ccid と secretKey を取得します。
String ccid = bundle.getString("merchant_" + merchantUniqueValue + "_ccid");
String secretKey = bundle.getString("merchant_" + merchantUniqueValue + "_secretKey");

MerchantSettingContext.Data data = MerchantSettingContext.getContext();
//マーチャント CCID を設定します
data.setMerchantCcid(ccid);
//マーチャント鍵を設定します
data.setMerchantSecretKey(secretKey);

//RequestDto にリクエストを設定する処理を行ってください。

//この getInstance()のタイミングで merchant_ccid, secretKey が設定されます。
tran = TransactionFactory.getInstance(reqDto)
```

（2） 結果通知における content-hmac の検証

VeriTrans4G より通知される結果通知の POST リクエストには、content-hmac というヘッダが設定されています。この値はリクエストボディから算出されます。

この content-hmac のチェックを行うことで、リクエストが改竄されていないかどうかをチェックすることが可能ですが、算出に SECRET_KEY の値を利用するため、content-hmac のチェック時には注意が必要となります。具体的なチェック方法につきましては、以下のサンプルをご参照ください。

【サンプルコード】

content-hmac のチェックには、MdkMerchantUtility クラスの checkMessage メソッドを利用します。

デフォルトのサンプルでは、String 型の引数を 2 つ指定するメソッドで実装されていますが、任意のマーチャント認証鍵を指定する場合は、以下のように、マーチャント認証鍵の値を第一引数に指定する checkMessage のオーバーロードメソッドを利用します。

```
import jp.veritrans.tercerog.mdk.util.MdkMerchantUtility;
・・・
String contentHmac = request.getHeader("content-hmac");
String body = ・・・ // リクエストボディの値
・・・
boolean isOK = MdkMerchantUtility.checkMessage("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX", body, contentHmac);
```

第二、第三引数に指定する値は、マーチャント認証鍵を指定しない場合のサンプルにおける body(リクエストボディの内容）、contentHmac(content-hmac ヘッダの値)の値と同じです。

補足：Java 版 MDK における認証情報のスコープ

以下のとおり、MerchantSettingContext.Data は Singleton で実装されています。

MerchantSettingContext.Data のオブジェクト（インスタンス）は、ThreadLocal に保存されているため、Java VM で 1 つではなく、該当スレッドで 1 つ保持される形となっています。

```
MerchantSettingContext.Data data = MerchantSettingContext.getContext();
//マーチャント CCID を設定します
data.setMerchantCcid(ccid);
//マーチャント鍵を設定します
data.setMerchantSecretKey(secretKey);
```

6. 改訂履歴

2017/04 ：Ver1.0.0 リリース

※ 以下、「3G MDK インストールガイド」 Ver 3.0.2 からの更新分を記載します。

「3.1 MDK・関連ライブラリの組込」の「表 1 Java 版 4G MDK ファイル一覧」について、

・tgMdk/tgMdk/を MdkJava/tgMdk/に修正

・tgMdk-3.x.x.jar を cg-mdk-x.x.x.jar に修正

・tgMdkDto.jar を cg-mdk-dto-x.x.x.jar に修正

「3.1.1 VeriTrans3G からのバージョンアップに関する注意点」を追加

「4 サンプルプログラム利用」各項のサンプルプログラムのアーカイブ名、ディレクトリ名を修正

「4.2 サンプルプログラム（Web アプリケーション）設定・動作確認」にトークンの設定を追加

2018/02 ：Ver1.0.1 リリース

「4.2 サンプルプログラム（Web アプリケーション）設定・動作確認」の本人認証（mpi.properties）の設定項目から決済種別（payment.mode）を削除

2020/07 ：Ver1.0.2 リリース

「3.1 MDK・関連ライブラリの組込」に新規ライブラリを追加

「4. サンプルプログラムの利用」に AmazonPay と PayPay に関する情報を追加

2023/01 ：Ver1.0.3 リリース

「4.2 サンプルプログラム（Web アプリケーション）設定・動作確認」から masterpass の記載を削除

「5.1 MDK メソッドリスト」からダウンロードサイトの URL の説明を削除

「5.2 MDK で複数のマーチャント ID を使い分ける方法」を追加

2023/12 ：Ver1.0.4 リリース

「3.2 MDK の設定」の SSL 通信用の設定をオプション設定に変更

2024/03 ：Ver1.0.5 リリース

動作する Java のバージョンを Java 8 に変更

log4j のバージョンを 1 系から 2 系に変更

「3.1 MDK・関連ライブラリの組込」の「表 1 Java 版 4G MDK ファイル一覧」

・改訂履歴を追加

・cg-mdk-dto-x.x.x.jar を cg-mdk-dto-java8 -x.x.x.jar に修正

・cg-mdk-x.x.x.jar を cg-mdk-java8-x.x.x.jar に修正

・log4j.propertie を log4j2.xml に修正

・依存ライブラリのバージョンを変更

以下の章タイトルを変更し、内容を修正

変更前：「3.1.1. VeriTrans3G からのバージョンアップに関する注意点」

変更後：「3.1.1. MDK のバージョンアップに関する注意点」

「4. サンプルプログラムの利用」

ログ出力用設定ファイル log4j.propertie を log4j2.xml に修正