## Document History

<table>
<thead>
<tr>
<th>Doc Version</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.03.2017</td>
<td>• Added isUnattended references to the Silent Installer and Lanes Section.</td>
</tr>
<tr>
<td>08.04.2017</td>
<td>• Added the CXPI driver to the Lanes Section.</td>
</tr>
<tr>
<td>07.11.2017</td>
<td>• Added a note to the Bluetooth Unpairing section that some Bluetooth devices must be on the “Lane Closed” screen for the unpairing process to work successfully.</td>
</tr>
</tbody>
</table>
| 5.26.2017   | • Adding instructions to configure the rolling of log files via the log4net.config.  
|             | • Added isCscSupported to Silent Installer and Configuration sections. |
| 03.06.2017  | • triPOS is no longer listening on the base route, the help documentation has moved to /api/help.  
|             | • Powershell command and example is updated as well. |
| 01.05.2017  | • Since triPOS now supports two Bluetooth devices, update Bluetooth connectivity section to be less specific toicmp. |
| 12.05.2016  | • Updated the Upgrading section with important information about Store and Forward |
| 09.14.2016  | • Added description for the configuration value isCscSupported |
| 09.13.2016  | • Added idleScreen/image to the PIN pad configuration section |
| 08.24.2016  | • Added store/forward configuration values to the transaction configuration section  
|             | • Added store configuration values to the lane configuration section |
| 08.18.2016  | • Added ICMP Connection via Bluetooth information |
| 07.28.2016  | • Added Upgrading section |
| 07.21.2016  | • Added idleScreen/message to the PIN pad configuration section |
| 06.29.2016  | • Updated the Silent Install section with the new INSTALLFOLDER argument. |
| 06.26.2016  | • Updated screenshot of Installation Complete screen  
|             | • Replaced all instances of ExpressPay with Express |
| 06.14.2016  | • Added tp-request-id documentation |
| 02.19.2016  | • Re-branded and reformatted document |
| 11.16.2015  | • Added definition list for all parameters available in Silent Install. |
| 09.04.2015  | • Added isHealthcareSupported to Silent Installer and Configuration sections.  
|             | • Added isEMVSSupported  
|             | • Added description of PIN pad TCP/IP setup |
# Table of Contents

## Introduction ................................................................. 6
  Transaction Flow .......................................................... 6
  Application Deployment .................................................... 8

## Preparing to Install triPOS ............................................... 9
  Create Your Express Test Credentials .................................. 9
  Download triPOS ............................................................ 10

## Installing triPOS .......................................................... 12
  Overview ........................................................................ 12
  Setup Wizard ................................................................... 13
  Silent Install .................................................................... 15
  Preserving triPOS.config Settings ...................................... 16
  Logging ........................................................................... 17
  Upgrading ........................................................................ 17

## PIN pad Setup ................................................................. 18
  PIN Pad Connection via Bluetooth ......................................... 18
  PIN pad FAQ ..................................................................... 19

## PIN pad connectivity ....................................................... 20
  triPOS connectivity monitoring ............................................ 20

## triPOS Configuration ...................................................... 21
  Developer Section .......................................................... 21
  Server Section ................................................................ 21
  Host Section ..................................................................... 21
  Application Section ........................................................ 22
  Transaction Section ........................................................ 22
  Lanes Section (contains PIN pad settings) .......................... 24
  EMV Section .................................................................... 26

## API Overview ................................................................. 27

## HATEOAS API Response Links ......................................... 28

## Required HTTP Request Headers .................................... 30
  Accept header (accept) ..................................................... 30
  Application headers (tp-application-*) .................................. 30
  HMAC authorization header (tp-authorization) ........................ 31
  Recommended HTTP request header (tp-request-id) .................. 32

## Creating an HMAC signature for triPOS ............................ 32

## Create a Self-Signed Certificate and Bind it ....................... 37
  Create the Certificate ..................................................... 37
  Bind the Certificate ........................................................ 39

## Developer Tools ............................................................ 40
  Online API Documentation ................................................ 40
  Demo Applications .......................................................... 44
Introduction

triPOS is a turnkey payment processing application designed to process both magnetic stripe and EMV transactions. triPOS easily interfaces with custom business management software. The application supports processing of domestic debit and credit transactions through the Express platform.

This document is intended for the integrator who wishes to interface to the Express platform via the triPOS payment processing application. It provides a high-level view of the installation process, an overview of the API, how to create a valid request signature and instructions for using the available sample applications.

This document does not cover generic card-present or EMV transaction processing, compliance requirements, hardware interfaces or direct integration to the Express platform. Please refer to the following documentation for further information on these subjects:

Transaction Flow

triPOS accepts requests from business management software for processing end-to-end financial transactions through the Express platform. The diagram below illustrates the interaction among the business management software, triPOS, the PIN pad and Express:

![Figure 1. Interaction among components](image-url)
The graphic below illustrates a typical transaction flow through the system:

<table>
<thead>
<tr>
<th>Business Management Software</th>
<th>Requests Payment</th>
<th>triPOS</th>
<th>Initiates transaction</th>
<th>Submits transaction</th>
<th>Finalizes transaction</th>
<th>Returns response</th>
<th>Generates receipt from response</th>
</tr>
</thead>
<tbody>
<tr>
<td>triPOS</td>
<td></td>
<td>triPOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PINpad</td>
<td>Provides cardholder and transaction info</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Express</td>
<td>Processes transaction</td>
<td></td>
<td></td>
<td></td>
<td>Finalizes transaction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receipt Printer</td>
<td>Sends Response</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Prints Receipt</td>
</tr>
</tbody>
</table>

**Figure 2: triPOS Transaction Flow**

1. The **business management software** sends a payment request to **triPOS**.

2. **triPOS** interfaces with the **PIN pad** to obtain the card information, manages cardholder selection of options and verification, then harvests the EMV tags required to complete the transaction.

3. If applicable, **triPOS** sends the transaction to the **Express** platform to be forwarded to the card issuing bank for an authorization attempt.

4. Upon receiving a response from the **Express** platform, **triPOS** proceeds with final validation of the transaction through the PIN pad.

5. **triPOS** returns a response in the same format as the request (JSON or XML) containing data that the business management software can use to create receipts for the merchant and cardholder.

6. The business management software prints the receipts utilizing the data returned by **triPOS**.
**Application Deployment**

`triPOS` has been validated for the following operating systems:

- Windows 7 (Professional/Enterprise - 32/64bit)
- Windows 8.1 (Professional/Enterprise - 32/64bit)
- Windows 10 (Professional/Enterprise - 32/64bit)

`triPOS` requires that Windows .NET 4.5 Framework is installed on the target machine. If this version of .NET is not present, it will automatically be installed when `triPOS` is installed.

The following diagram illustrates a typical `triPOS` deployment. Both the business management software and `triPOS` reside on the same physical machine to which a PIN pad is attached via a serial or IP connection.

![Figure 3: triPOS deployment diagram](image)
Preparing to Install triPOS

Create Your Express Test Credentials

1. If you do not already have Express test credentials, go to http://www.elementps.com/Create-a-Test-Account and create a test account. You will receive an email containing the credentials needed to install triPOS:

   - AccountID
   - AccountToken
   - ApplicationID
   - AcceptorID

2. Register on the Element site: https://mft.elementps.com/backend/plugin/Registration
3. After clicking Register, you will receive an email with a link to complete your registration:

![Element Payment Services MFT registration](image)

Thank you [name] for registering to Element Payment Services managed file transfer service.

Complete your registration by opening following confirmation link:
https://mft.elementps.com/backends/plugin/Registration/?confirm=khoehn40mercurypay.com&key=550963622b3b185304882

This notification was sent to [email address]

4. Click the link to complete your registration and you will receive a confirmation email:

Download triPOS

1. Login at https://mft.elementps.com with your email address and the password created when you registered (step 2 above).

![Login](image)

2. This will display the Element Product download page. Click on the Element triPOS folder.

![Element triPOS download](image)

3. The download page looks like this:

   ![device drivers]
   ![triPOS SDK zip]

   **Note** Use this same page to download the device drivers.
4. Extract the files to the location of your choice.
5. Follow the instructions under Installing triPOS to complete installation.
Installing triPOS

Overview

triPOS consists of binary files, configuration files and supporting files held in a dedicated directory structure that is created by the triPOS installer. In addition, a directory for logs is created at runtime underneath the installation directory.

The triPOS installer requires that the user performing the installation be part of the local Administrators group. triPOS runs as a Windows Service under the Local System account.

Registry Entries

The triPOS installer writes the installation directory to the HKLM (Local Machine) section of the registry.

<table>
<thead>
<tr>
<th>Locations</th>
<th>Software\Vantiv\triPOS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Software\Wow6432Node\Vantiv\triPOS</td>
</tr>
<tr>
<td>Name</td>
<td>InstallFolder</td>
</tr>
<tr>
<td>Data Description</td>
<td>The full path of the installation directory</td>
</tr>
</tbody>
</table>

To change triPOS.config settings after installation, use the available endpoints documented under http://localhost:8080/api/help.

Installation Options

There are two ways to install triPOS:

- **Setup Wizard**
- **Silent Install**

The Setup Wizard contains configuration screens that ask for initial configuration details. The same information can be passed in as arguments when run silently.

The Silent install allows you to install triPOS through the command-line. This is useful if you want to package the triPOS installation as part of the installation process for your own software. Both methods are described below.
Setup Wizard

1. Double-click on triPOS Setup.exe to launch the setup wizard.

2. The first screen is the TRIPOS® SOFTWARE LICENSE AGREEMENT:

To select a custom installation directory, click **Options** and enter the desired location.

The default installation paths are:

- **64-bit machine**: C:\Program Files (x86)\Vantiv\TriPOS Service
- **32-bit machine**: C:\Program Files \Vantiv\TriPOS Service

3. Click **I agree to the terms and conditions**, then click **Install**.

4. The next screen is **Configuration Information**. Enter your Express credentials here. All fields are required to continue with installation. These credentials can be edited later by manually editing the triPOS.config file in Admin mode.

   **Note** For an explanation of each field, see the table of arguments under **Silent Install**.
5. Click **Next** to reach the **Completed the triPOS Setup Wizard** screen.

6. Click **Finish** to reach the **Setup Successful** screen.
Silent Install

triPOS can also be installed silently through the command-line. Below is a sample command:

```
"triPOS Setup.exe" /q USERCOMPANY="Sample Company"
LISTENINGPORT=8080 ACCEPTORID=1234 ACCOUNTID=1234
ACCOUNTTOKEN=1234 MARKETCODE=Retail TERMINALID=000077 LANEID=1
ISMANUALENTRYALLOWED=true COMPORT=COM1
DEVKEY1=123 DEVSECRET1=456
```

- `/q` – This tells the installer to run silently so that it does not display any UI components.
- `USERCOMPANY` – This is the company name that will be displayed on the PIN pad’s welcome message.
- `INSTALLFOLDER` – This is the path that you would like triPOS to be installed at.

The following triPOS.config values can be set through command-line arguments during the silent install as well. Please refer to the triPOS Configuration section of this document for a description of each field.

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPTORID</td>
<td>HANDSHAKE</td>
</tr>
<tr>
<td>ACCOUNTID</td>
<td>ISCASHBACKALLOWED</td>
</tr>
<tr>
<td>ACCOUNTTOKEN</td>
<td>ISCONTACTLESSMSDENTRYALLOWED</td>
</tr>
<tr>
<td>ALLOWPARTIALAPPROVALS</td>
<td>ISDEBITREFUNDSUPPORTED</td>
</tr>
<tr>
<td>BAUDRATE</td>
<td>ISDEBITSPUPPORTED</td>
</tr>
<tr>
<td>CASHBACKAMOUNTINCREMENT</td>
<td>ISEMVSUPPORTED</td>
</tr>
<tr>
<td>CASHBACKSELECTIONS</td>
<td>ISGIFTSUPPORTED</td>
</tr>
<tr>
<td>CHECKFORDUPLICATETRANSACTION</td>
<td>ISHEALTHCARESUPPORTED</td>
</tr>
<tr>
<td>CLIENTPINPADLISTENINGPORT</td>
<td>ISMANUALENTRYALLOWED</td>
</tr>
<tr>
<td>COMPORT</td>
<td>ISTIPALLOWED</td>
</tr>
<tr>
<td>CONFIRMCONVENIENCEFEEAMOUNT</td>
<td>LANEID</td>
</tr>
<tr>
<td>CONFIRMORIGINALAMOUNT</td>
<td>LISTENINGPORT</td>
</tr>
<tr>
<td>*CORSALLOWEDORIGINS</td>
<td>MARKETCODE</td>
</tr>
<tr>
<td>CREDITAVSENTRYCONDITION</td>
<td>MAXIMUMCASHBACKAMOUNT</td>
</tr>
<tr>
<td>CREDITSALESIGNATURETHRESHOLDAMOUNT</td>
<td>PARITY</td>
</tr>
<tr>
<td>CURRENCYCODE</td>
<td>SIGNATUREFORMAT</td>
</tr>
<tr>
<td>DATABITS</td>
<td>STOPBITS</td>
</tr>
<tr>
<td>DEBITSURCHARGE</td>
<td>TERMINALID</td>
</tr>
<tr>
<td>DEVKEY1</td>
<td>TERMINALTYPE</td>
</tr>
<tr>
<td>DEVSECRET1</td>
<td>TIPSELECTIONS</td>
</tr>
</tbody>
</table>
Preserving triPOS.config Settings

The main configuration file for triPOS is the triPOS.config file. It holds the Express credentials, lane configuration and transaction-related settings. After installing for the first time, you customize this file to configure triPOS.

Consider the scenario where version A is installed and you upgrade to version B.

Until now version B’s installer lacked the ability to preserve triPOS.config settings when creating the new triPOS.config file. During an upgrade, installer A created a backup of triPOS.config in its current install directory, and installer B completely replaced it with a new one. The settings in the backup file needed to be manually copied over to the settings in the new file.

Now, installer B can find and use the most recently created backup to produce a new triPOS.config file that contains new settings along with the preserved settings from version A.

Supported Versions

This functionality is supported when upgrading from version 4.2.207 or higher to version 5.3.0 or higher. Earlier versions are not supported because they are not equipped with three important functions that the new installer uses to preserve the triPOS.config settings: a registry entry with the install location of version A, a backup file created by version A’s installer, and a <configVersion> node in the triPOS.config.

Regardless of the installer’s capabilities, it is always recommended that you create your own backup of the triPOS.config file to save your settings in case of an error.

The triPOS.config Backup

To select the right backup, installer B’s preservation process, which we will refer to here as the preserver, first looks in the registry for the directory of version A. Next, the preserver looks in that directory to find all backup files named “triPOS.config.2*”. If more than one backup file is found, the preserver selects the most recently created backup to use during the upgrade.

If no previous install path is found in the registry, or that directory does not contain any backups, the preserver checks for backups in the current install directory (the path selected for version B). If a backup is found and contains a supported version node, the installer preserves these values before continuing to finalize the new triPOS.config. If a backup is not found, or it does not contain a supported version node, the installer continues to finalize the new triPOS.config.

Finalization

At this stage, the triPOS.config file is either a brand-new default configuration (if no backups were found to preserve), or it contains new settings merged with the preserved settings.
The first stage of finalization processes the command-line arguments that you are able to pass in during an install or upgrade. The command-line arguments take precedence over even the preserved settings, and are mapped directly into the triPOS.config. For example, if your resulting triPOS.config at this stage has an “allowPartialApprovals” value of “true”, and you pass the argument “ALLOWPARTIALAPPROVALS=false” during the upgrade, the final value in the new triPOS.config will be false.

The mapping is straightforward for most of the sections, but the developer and lanes sections are special.

The developer section can contain 1-n sets of developer credentials. The installer only allows one set to be passed as command-line arguments during install or upgrade. For each the developer key and developer secret separately, if the command-line value is NOT null, it replaces the value in the top set of developer credentials.

The lanes section can contain 1-n serial lanes and 1-n IP lanes. The installer only allows one set of serial lane settings to be passed as command-line arguments. If one or more lane setting is NOT null, the installer collects all of the lane argument values into one serial lane, which we will refer to here as the installer lane. If the LANEID of the installer lane is found in the triPOS.config file, the installer replaces the lane in the file with the entire installer lane (if some installer lane values are empty, they are written into the file as empty). If the LANEID of the installer is not found in the file, or is empty, the installer inserts the entire installer lane into the file as a new serial lane.

The installer does not perform any validation on the command-line arguments that are passed during install or upgrade. If the command-line arguments contain invalid values or a misconfigured lane, triPOS may not be able to start successfully, and the upgrade or install will fail. Make sure to fully test your command-line arguments in a test environment to verify that they are valid.

**Logging**

It is also recommended to run the installer from the command-line (either silent or manual) to take advantage of the logging that occurs throughout any installer process. To capture the logs, run the following command to launch the installer:

```
“triPOS Setup.exe” -l log.txt
```

The -l (lowercase L) argument specifies that the logs are to be written to the file specified as the second argument, log.txt. To install silently, add the /q argument as illustrated below:

```
“triPOS Setup.exe” /q -l log.txt
```

In either case, this command will produce two logs files. One is called log.txt (or the name you’ve provided in the command) which holds the logs of the executable Bootstrapper that wraps the .msi.

The second log is called log_000_triPOS.txt which contains the bulk of the installer’s logs. Here, you can follow the process and custom actions that execute throughout the installation or upgrade.

**Upgrading**

To upgrade from version 5.9 or older, it is required to delete your triPOS.db file before you will be able to store transactions using the Store and Forward feature. **IMPORTANT: Make sure to forward all**
stored transactions before deleting the triPOS.db file. All unprocessed transactions will be lost when deleting the triPOS.db file.

**PIN pad Setup**

1. Download the appropriate device driver from the same site you downloaded the triPOS SDK.
2. Install the device driver.
3. Update the triPOS.config file with the appropriate settings if they were different from the default settings.
4. Stop and restart the service so the changes go into effect.

Please see the [Configuration - Lanes Section](#) in this document for detailed information.

⚠️ **Note** When installing the device driver, be sure to note the COM port the PIN pad will be using. You must edit the triPOS.config file for that COM port. If you do not know the COM port, you can find it by going into the Windows Control Panel > Hardware and Sound > View Devices and Printers.

**Example Lane Configuration**

```xml
<serialLane laneId="1" description="MX915 on COM1">
  <pinpad>
    <terminalType>1</terminalType>
    <driver>VeriFoneFormAgent</driver>
    <comPort>COM9</comPort>
    <dataBits>8</dataBits>
    <parity>None</parity>
    <stopBits>One</stopBits>
    <handshake>None</handshake>
    <baudRate>115200</baudRate>
    <isManualEntryAllowed>true</isManualEntryAllowed>
    <idleScreen>
      <message>This is No.1 lane</message>
      <image>logo.png</image>
    </idleScreen>
  </pinpad>
  <transaction/>
  <host>
    <terminalId>0000000001</terminalId>
  </host>
  <store>
    <transactionAmountLimit>10</transactionAmountLimit>
  </store>
</serialLane>
```

**PIN Pad Connection via Bluetooth**

**Standard Bluetooth Pairing**

1. Ensure the PIN pad is powered on and the Standard Bluetooth device has Bluetooth connectivity enabled.
2. The PIN pad will boot to a pairing screen. If the terminal is not at this screen, then the terminal must be unpaired, see section [Bluetooth Unpairing](#).
3. To begin the pairing process, select the “Standard” key (F2) on the PIN pad.
4. Select the pair option on the Bluetooth device.
5. The PIN pad will display an eight(8) digit randomly generated pairing PIN:

![BT Name: ICMP-12345678
BT Pairing...
PIN: 12345678](image)

6. On the standard Bluetooth device, search for the terminal’s logical Bluetooth name that is on the screen of the terminal and select it to pair.
7. When the standard Bluetooth device prompts for a PIN, enter the PIN that is displayed on the terminal screen.

**Bluetooth Unpairing**
1. To unpair the PIN pad from the host simply press the ‘Function’ key four (4) times in less than two (2) seconds.
   
   **Note:** Some Bluetooth devices can only be un-paired when the device is on the offline screen. When the triPOS service is stopped and it shuts down the lanes, it may appear the device is on the offline screen since it displays “Lane Closed”, but that is not always the case. Always first stop triPOS, then reboot the device to reach the offline screen before attempting to un-pair.
2. The terminal will beep and then go to the BT Pairing Required screen (or some terminals may have a specific “Unpair” button).

**PIN pad FAQ**

**How do lanes work with triPOS? Do we need to create a new lane for each terminal every time the batch closes?** No, you do not need to create a new lane for each terminal every time the batch closes. A lane is not related to an batching process at all. You create a lane for each PIN pad. For example, if you have five PC’s each with its own PIN pad, then create five separate lanes.

**Do I need to create a lane for each terminal at the start of each business day?** No, this is a onetime process. Lanes are defined in the *triPOS.config* and can be configured by either editing the XML or using the *triPOS* configuration API.
Do I need to check the status of the lane prior to every transaction to verify the lane has already been created? No. Lanes only need to be configured one time prior to running any transaction. You can verify that a lane exists but you should not have to check the status of a lane prior to every transaction. If a lane is currently in use when you try to send a request using that lane, triPOS will indicate that the lane is in use.

I am using a VeriFone MX915 USB PIN pad and it was accidentally disconnected. Now it isn’t communicating with triPOS. How do I correct this? Turn off the PIN pad, re-boot the computer, then turn on the PIN pad.

I am using an iCMP or other Bluetooth PIN pad, what type of lane do I create? When the PIN pad is paired it will add 2 COM ports in your Device Manager. It is not immediately apparent which COM port should be used. Set up the PIN pad as a serial lane trying each COM port individually to determine which is the appropriate connection to triPOS.

**PIN pad connectivity**

**triPOS connectivity monitoring**

triPOS will monitor PIN pad connectivity every 60 seconds in perpetuity. All lanes present in the configuration file will be monitored during each sweep. If any of the lanes have disconnected PIN pads a reconnection attempt will be made during the next sweep. When a PIN pad reconnects it will also be configured. Lanes updated via the API will trigger an immediate sweep but please allow up to 60 seconds for changes to take effect. All connections to triPOS are assumed to be open and valid until an API request fails. Once a request fails due to a PIN pad disconnect, the lane is flagged for reconnection.
triPOS Installation and Configuration

The triPOS configuration is stored in an XML file named triPOS.config located in the root of the installation directory. Some configuration values are set during installation. Once triPOS is installed, additional configuration may be required before using it for the first time.

In addition to the description below, details about each section can be found within the configuration file itself.

**Note** To change triPOS.config settings after installation, use the available endpoints documented under [http://localhost:8080/api/help](http://localhost:8080/api/help) (See Swagger Documentation – View API later in this document for more information).

### Developer Section

**developer**

A developer entry, which is comprised of a developer key and a developer secret, is used to authenticate the requests you send to triPOS (see the Headers section for more details). There can be multiple developer entries, but the installer creates two entries for you. You can set the developerKey and developerSecret to any value as long as the developerKeys are unique within this file. These identities can be used across multiple installations of triPOS on different machines.

- **developerKey**: This is the key that is transmitted in the clear as part of the header of each request to triPOS.
- **developerSecret**: This is the secret that is used to sign each request to triPOS.

A GUID is a good value to use for these entries.

### Server Section

**listeningPort**

This is the port the triPOS service will listen on. Setting it to **8080**, for example, will allow you to send requests to triPOS through the URL: [http://localhost:8080/api/...](http://localhost:8080/api/...)

**useHttps**

A boolean value that determines whether or not triPOS will accept https REST requests.

**clientPinPadListeningPort**

This is the listening port that triPOS will use to accept incoming tcp/ip connections from client PIN pads.

### Host Section

**driver**

Defines what host processor to communicate with (currently, only Express is supported). Use **Null** to test without a connection to a host processor.

**acceptorId**

Your Express acceptor ID which is also your Merchant ID.

**accountId**

Your Express account ID.

**accountToken**

Your Express account token.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoReversalRetryLimit</td>
<td>If a communication error occurs among triPOS, the processor and the host, triPOS will try to reverse the transaction to allow you to resend it while avoiding duplicates. This value limits how many times triPOS tries to reverse the transaction before returning to the client application.</td>
</tr>
<tr>
<td><strong>Application Section</strong></td>
<td></td>
</tr>
<tr>
<td>testMode</td>
<td>Set to <strong>true</strong> to be in Test/Certification mode or <strong>false</strong> for Production mode. In test mode, the Express processor endpoint will go to the Test/Certification endpoints. Also, the HTTP authorization header requirements are relaxed in test mode. For more information, see the Required HTTP authentication headers section.</td>
</tr>
<tr>
<td>pinPadIdleMessage</td>
<td>The message that the PIN pad will display during idle.</td>
</tr>
<tr>
<td>corsAllowedOrigins</td>
<td>Origins triPOS will accept requests from: Disabled by default, use ‘*’ to accept from any origin</td>
</tr>
<tr>
<td><strong>Transaction Section</strong></td>
<td></td>
</tr>
<tr>
<td>allowPartialApprovals</td>
<td>If set to <strong>true</strong>, the business application is equipped to handle partial approvals. This value <strong>can be overridden on a per-request basis</strong> by setting the allowPartialApprovals value in the configuration section of applicable requests.</td>
</tr>
<tr>
<td>confirmOriginalAmount</td>
<td>If set to <strong>true</strong>, the PIN pad will prompt the cardholder to confirm the original amount transaction after it obtains the card information.</td>
</tr>
<tr>
<td>creditAvsEntryCondition</td>
<td>This value determines how often, if ever, the PIN pad should prompt the user to enter the billing zip code associated with the card. <strong>Values</strong>: Always, Never or Keyed. <strong>Keyed</strong> indicates that manually-entered card data should be prompted for AVS.</td>
</tr>
<tr>
<td>checkForDuplicateTransactions</td>
<td>If set to <strong>true</strong>, enables duplicate checking for all transactions. This <strong>can be overridden on a per request basis</strong>.</td>
</tr>
<tr>
<td>currencyCode</td>
<td>The currency code that will be used for all transactions. <strong>Values</strong>: <strong>Usd</strong> (US dollars), <strong>Cad</strong> (Canadian dollars) or <strong>Eur</strong> (Euro). Currently, only <strong>Usd</strong> is supported. This value can be <strong>overridden on a per-request basis</strong> by setting the currencyCode value in the configuration section of applicable requests.</td>
</tr>
<tr>
<td>isCashbackAllowed</td>
<td>Set to <strong>true</strong> to allow cashback.</td>
</tr>
<tr>
<td>isDebitSupported</td>
<td>Set to <strong>true</strong> to allow debit-only cards to be used. The credit functionality of check cards will still work if this is set to <strong>false</strong>.</td>
</tr>
<tr>
<td>isGiftSupported</td>
<td>Set to <strong>true</strong> to allow gift-only cards to be used.</td>
</tr>
<tr>
<td>isEmvSupported</td>
<td><strong>Values</strong>: true or false</td>
</tr>
</tbody>
</table>
|                             | The default value for this field is false. This field can be passed in
during silent install to be set to true. If set to true, EMV is enabled and your application must support accepting EMV transactions. To not allow EMV transactions, set this value to false.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values: true or false</th>
</tr>
</thead>
<tbody>
<tr>
<td>isHealthcareSupported</td>
<td></td>
</tr>
<tr>
<td>confirmConvenienceFeeAmount</td>
<td>Set to true to require the PIN pad to prompt the cardholder to accept the convenienceFeeAmount if the amount is non-zero.</td>
</tr>
<tr>
<td>isTipAllowed</td>
<td>Set to true to allow tips to be added.</td>
</tr>
<tr>
<td>cashbackSelections</td>
<td>Enter whole numbers separated by commas for the cashback amounts to select from. Use other to show other on the PIN pad.</td>
</tr>
<tr>
<td>cashbackAmountIncrement</td>
<td>Increments you want customers to be able to get back. Values: common values include, 1, 5, 10 and 20.</td>
</tr>
<tr>
<td>marketCode</td>
<td>The market code specific to the types of transactions that will be submitted. This value can be overridden on a per-request basis by setting the marketCode value in the configuration section of applicable requests. Values: AutoRental, DirectMarketing, Ecommerce, FoodRestaurant, HotelLodging, Petroleum, Retail, QSR.</td>
</tr>
<tr>
<td>maximumCashbackAmount</td>
<td>The maximum cashback amount allowed for cardholders to get back.</td>
</tr>
<tr>
<td>tipSelections</td>
<td>• Enter decimal numbers for tip amounts. • Add percent sign after the number to display calculated percentages (i.e. 10%, 15%, 20%). • Add none to show None on the PIN pad, otherwise Other will always be shown on the PIN pad.</td>
</tr>
<tr>
<td>debitSurcharge</td>
<td>Enter a decimal amount to add to debit transactions Note Please read the credit / debit card surcharge statutes pertaining to your state of business: <a href="http://www.ncsl.org/research/financial-services-and-commerce/credit-or-debit-card-surcharges-statutes.aspx">http://www.ncsl.org/research/financial-services-and-commerce/credit-or-debit-card-surcharges-statutes.aspx</a>.</td>
</tr>
<tr>
<td>creditSaleSignatureThresholdAmount</td>
<td>Any transaction under this amount will not require a signature. Leave as 0 or empty to always require a signature.</td>
</tr>
<tr>
<td>signatureFormat</td>
<td>triPOS will return the signature data in the format you choose here Values: PointsLittleEndian, PointsBigEndian, Ascii3Byte Note Ascii3Byte format has been around for over 20 years and is supported by most signature-capable PIN pads. Big and little Endian point formats are also included for those who prefer not to decode 3-byte ASCII.</td>
</tr>
<tr>
<td>emvFallbackAllowed</td>
<td>EMV fallback occurs when a the chip of a chip card cannot be used, and the system allows the cardholder to swipe the card instead. If</td>
</tr>
</tbody>
</table>
this configuration value is set to **NeverAllow**, then the system will not allow the cardholder to swipe the chip card at all. If this value is set to **AllowAfterChipError**, the system will only allow a swipe fallback if there is a chip or chip reader error.

<table>
<thead>
<tr>
<th><strong>isDebitRefundSupported</strong></th>
<th>Set to true to allow refunds onto a debit card. Not all processors support this.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>isCscSupported</strong></td>
<td>Set this value to true to prompt the cardholder to enter the card security code when the card is manually keyed into the PIN pad.</td>
</tr>
</tbody>
</table>

### store

<table>
<thead>
<tr>
<th><strong>storeMode</strong></th>
<th>Allow triPOS to store transactions automatically upon losing connectivity, or merchant chooses to store on per request basis. Values: automatic, perRequest.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>unprocessedTotalAmountLimit</strong></td>
<td>When storing, triPOS will decline any transaction that causes the sum of all unprocessed transactions to exceed this maximum amount. Values: Integer greater than 0.</td>
</tr>
</tbody>
</table>

### forward

<table>
<thead>
<tr>
<th><strong>forwardMode</strong></th>
<th>Allow triPOS to forward transactions automatically upon gaining connectivity, or merchant chooses to forward on per request basis. Values: automatic, perRequest.</th>
</tr>
</thead>
</table>

### Lanes Section (contains PIN pad settings)

### lane

<table>
<thead>
<tr>
<th><strong>laneld</strong></th>
<th>The ID of the lane to be used in the REST requests to designate which Lane (PIN pad) to use. To test without a physical PIN pad, set laneld to 9999 to use the built-in test PIN pad that simulates a credit sale.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>description</strong></td>
<td>A general description for your own internal use to help keep track of multiple lanes</td>
</tr>
<tr>
<td><strong>initializeOnStartup</strong></td>
<td>Set to <strong>false</strong> if you do not want this lane initialized each time triPOS starts up. In this case, the lane will initialize upon the first request sent to it. This may be useful when multiple lanes have been defined for test purposes. Set to <strong>true</strong> (default) or leave the attribute out of the element if you want the lane initialized each time the service starts.</td>
</tr>
</tbody>
</table>

### pinpad (serial/USB)

<table>
<thead>
<tr>
<th><strong>terminalType</strong></th>
<th>Indicates what function / type of PIN pad is being used</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Values</strong></td>
<td>Unknown, PointOfSale, Ecommerce, Moto, FuelPump, Atm.</td>
</tr>
<tr>
<td><strong>driver</strong></td>
<td>Indicates which PIN pad driver to use.</td>
</tr>
<tr>
<td><strong>Values:</strong> Null, IngenicoRba, VeriFoneXpi, VeriFoneCXpi, VeriFoneFormAgentXpi</td>
<td></td>
</tr>
</tbody>
</table>

| **comPort** | If the PIN pad is not connected via IP, name of serial device to which the PIN pad is attached (for example, COM1, /dev/tty1, etc.). |
| **dataBits** | If the PIN pad is connected via serial device, the data bit setting. |
| **Values:** typical values are either 7 or 8 |

| **parity** | If the PIN pad is connected via serial device, the parity setting. |
| **Values:** None, Even, Odd. |

| **stopBits** | If the PIN pad is connected via serial device, the stop bits setting. |
| **Values:** None, One, OnePointFive, Two |

| **handshake** | If the PIN pad is connected via serial device, the handshake setting. |
| **Values:** None, RequestToSend, XonXoff |

| **baudRate** | Baud rate of serial communications |
| **Values:** |
| - For the VX series: baud rate is typically 9600 |
| - For the MX series: |
| - Serial connection: 57600 |
| - USB connection: 115200 |

| **isManualEntryAllowed** | Set to true to allow manual entry of card account numbers on the PIN pad. |

| **isContactlessMsdEntryAllowed** | Set to true to allow contactless entry. |

| **idleScreen/message** | The message that the PIN pad will display during idle. |

| **idleScreen/image** | The image that the PIN pad will display during idle. |

| **isUnattended** | Boolean value to indicate that the PIN pad is an unattended device. |

**pinpad (TCP/IP)**

| **terminalType** | Indicates what function / type of PIN pad is being used |
| **Values:** Unknown, PointOfSale, Ecommerce, Moto, FuelPump, Atm. |

| **driver** | Indicates which PIN pad driver to use. |
| **Values:** Null, IngenicoRba, VeriFoneCXpi, VeriFoneFormAgentXpi |

| **ipAddress** | The IPv4 address of the PIN pad on the network (for example, 192.168.1.2). Depending on the setup, this value is either configured statically on the PIN pad or assigned statically by the DHCP server. |

| **ipPort** | The 16-bit TCP port number to which the PIN pad is listening for a connection. This value is configured on the PIN pad. |

| **isManualEntryAllowed** | Set to true to allow manual entry of card account numbers on the PIN pad. |
isContactlessMsdEntryAllowed | Set to true to allow contactless entry.
idleScreen/message | The message that the PIN pad will display during idle.
idleScreen/image | The image that the PIN pad will display during idle.
isUnattended | Boolean value to indicate that the PIN pad is an unattended device.

**host**

terminalId | The unique identifier of the terminal / PIN pad for Express

**store**

transactionAmountLimit | When storing, triPOS will decline any transaction greater than this amount. 
Values: Integer greater than 0.

**EMV Section**

The EMV configuration is stored in an XML file named `triPOS.EMV.config` that is also located in the root of the installation directory. Below is a brief explanation of some important default EMV values. We recommend that you do not change any of the values in the EMV section of the `triPOS.config` without consulting the Certification Team.

**terminalCapability**

The default value for this field enables the following Cardholder Verification Methods (CVMs): online PIN, offline plaintext PIN, offline enciphered PIN and none.

**ridConfiguration/aidConfiguration**

These sections hold the necessary configuration to define which brands are supported. The default configuration supports the following brands: Visa, Visa Electron, Visa Interlink, Visa US debit, MasterCard, Maestro, MasterCard/Maestro US debit, Discover, Discover US debit and American Express.
API Overview

API REST Documentation
Documentation and descriptions of the REST API for triPOS are located on the installed triPOS service. Browse to http://localhost:8080/api/help for up-to-date documentation.

Message Transport
The business management software communicates with triPOS over a local TCP/IP socket connection. To simplify integration of existing solutions, requests and responses are transported via HTTP in the same fashion as the business management software would when interfacing directly to the Express platform (though the Express platform itself supports HTTPS only).

Message Format
triPOS accepts either XML or JSON formatted request messages and returns responses in the same format as the request.

Each request is identified by a transaction type and is accompanied by data elements pertaining to the request. However, a typical triPOS request is simpler than an Express request as card information is not included. Card information is obtained downstream via direct interactions between triPOS and the PIN pad.

Each request requires a header with specific fields. If the request is a POST or PUT request it requires parameters to be sent in the request body. For GET and DELETE, any parameters will be sent up in the URL’s query string. For any type of request, some values such as PaymentType may be sent in the URL. See the API documentation at http://localhost:<port>/api/help (once triPOS is installed) for more detailed API documentation.

Receipts
Receipts can be generated using the returned data from each transaction. If a signature was collected on the PIN pad, the Signature field of the response will contain a byte array of the signature data along with the format of the signature data. You can look at our sample DisplaySignature application for source code that shows how to convert this byte array into a BMP.
HATEOAS API Response Links

triPOS implements HATEOAS (Hypertext as the Engine of Application State). HATEOAS allows a business application to interact with the triPOS REST service through dynamic hyperlinks returned in the response. Using these response links is a best practice for decoupling the business application from the triPOS REST API. It is also the best way to future-proof an application for additional triPOS functionality as it allows you to avoid hard-coding endpoint URLs in your application.

Each return link contains three values: href, method, and relation. For example, for a default installation, sending a GET request to the root triPOS service URL at http://localhost:8080/api/v1/ will return a list of the root triPOS hyperlinks.

Sample Response from GET Request to api/v1

```
{
"_errors": [],
"_hasErrors":false,
"_links": [
{
"href":"http://localhost:8080/api/v1/sale",
"method":"POST",
"rel":"sale"
},
...
...
{
"href":"http://localhost:8080/api/v1/authorization",
"method":"POST",
"rel":"authorization"
},
...
...
...
...
"_logs": [],
"_type":"getServicesResponse",
"_warnings":[]
}
```

From this response you know:

- **To transact a sale,** POST to the href value of the link containing “sale” as the relation and
- **To transact an authorization,** POST to the href value of the link containing “authorization” as the relation

Furthermore, a POST request sent to the sale endpoint might generate this response:
Sample Response from POST Request to api/v1/sale

```
{
    "approvedAmount":25.00,
    "paymentType":"Credit",
    "transactionId":"2004010421",
    ...
    ...
    "_errors":[]
    "_hasErrors":false,
    "_links":
    [
      {
        "href":"http://localhost:8080/api/v1/return/2004010421/credit",
        "method":"POST",
        "rel":"return"
      }
    ],
    "_logs":[]
    "_type":"saleResponse",
    "_warnings":[]
}
```

From this response, the **href** indicates that the endpoint hyperlink for returning a sale is in the format `/api/v1/return/{transactionId}/{paymentType}`. Note that `{transactionID}` and `{paymentType}` are variables.

- For this particular credit card sale request to localhost, the **transactionId** of **2004010421** was returned, therefore the return endpoint URL is **http://localhost:8080/api/v1/return/2004010421/credit**.

**Note** The **triPOS** development team **strongly recommends** business applications use the HATEOAS response link with the appropriate relation to call any service endpoint (for example, always use the **return** relation with method **POST** to POST a **Return** to the **API**).
Required HTTP Request Headers

To verify that each request is coming from an authorized source, triPOS checks specific values that the client must send up in the header with each request in a value called tp-authorization. Along with the tp-authorization header, the client must always send up application-related fields in the header. triPOS can be switched into test mode by editing the application section in the triPOS.config file and setting testMode to true. The authorization header should look like this in production mode:

Sample Production HTTP Header

```
<tp-authorization: Version=1.0, Algorithm=TP-HMAC-SHA1, Credential=045c58b9-7441-495f-849a-5a6de45fbe31, SignedHeaders=, Nonce=b1c0c4bd-546f-44e1-8187-4249bf725d36, RequestDate=2014-11-26T19:57:54.4548775Z, Signature=668b83860dac84db30b3557b14d87b8c29f27ad4
tp-application-id: 12345
tp-application-name: MyApplicatio
tp-application-version: 1.0.0
tp-request-id: ee25beb5-99f6-463a-bb51-66f69379cf0c
tp-return-logs: false
accept: application/xml
```

In test mode, the authorization header can look like this, although including all elements of the authorization header will make triPOS try to validate the header so you can verify that your application is computing the signature correctly:

Sample Test Mode HTTP Header

```
<tp-authorization: Version=1.0, Credential=045c58b9-7441-495f-849a-5a6de45fbe31
tp-application-id: 12345
tp-application-name: MyApplicatio
tp-application-version: 1.0.0
tp-request-id: ee25beb5-99f6-463a-bb51-66f69379cf0c
tp-return-logs: false
accept: application/json
```

⚠️ Note The credential value must match a developer key in the triPOS.config.

Accept header (accept)

The Accept request header field is used to specify the media type (JSON or XML) that is acceptable for the response. All responses from triPOS back to the business application will be in the format specified in the accept header. Acceptable values for triPOS are:

- accept: application/json
- accept: application/xml

⚠️ Note It is recommended to set the accept header on every request.

Application headers (tp-application-*

Three headers are required to identify which application is calling triPOS. These values will be passed on to the host and reflected in the Express dashboard:

- tp-application-id: The ID of your application
- tp-application-name: The name of your application
- tp-application-version: The version of your application
**HMAC authorization header (tp-authorization)**

The HMAC authorization header serves multiple purposes for triPOS:

- Ensures that messages being sent are from an authorized user of triPOS
- Ensures that messages are not altered in transit from the business management software to triPOS
- Ensures that messages are not replayed before they are set to expire

The triPOS HMAC authorization is based on a modified version of the main authentication method used by Amazon Web services. It is made up of seven parts: **Version, Algorithm, Credential, SignedHeaders, Nonce, RequestDate** and **Signature**. See [Creating an HMAC signature for triPOS](#) for more information.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version</strong></td>
<td>The version specifies which version of the header structure and signature computation process is being used. If this request validation methodology changes in the future, the version will increment.</td>
</tr>
<tr>
<td><strong>Algorithm</strong></td>
<td>The algorithm tells triPOS how the message was encrypted. The following values are valid:</td>
</tr>
<tr>
<td></td>
<td>- tp-hmac-md5</td>
</tr>
<tr>
<td></td>
<td>- tp-hmac-sha1</td>
</tr>
<tr>
<td></td>
<td>- tp-hmac-sha256</td>
</tr>
<tr>
<td></td>
<td>- tp-hmac-384</td>
</tr>
<tr>
<td></td>
<td>- tp-hmac-512</td>
</tr>
<tr>
<td></td>
<td>When creating the signature, the encryption algorithm used to create hashes must match the algorithm defined in the authorization header</td>
</tr>
<tr>
<td><strong>Credential</strong></td>
<td>The credential must match one of the developer keys found in the triPOS.config file. Each developer key is paired with a developer secret, both of which will be used when calculating the signature</td>
</tr>
<tr>
<td><strong>SignedHeaders</strong></td>
<td>This field holds the list of headers which you have chosen to sign as part of the HMAC signature. You may choose to sign no headers or you can sign all of them, except for the tp-authorization field</td>
</tr>
<tr>
<td><strong>Nonce</strong></td>
<td>A nonce is a random string uniquely generated by the client, which allows the server to verify that a request has never been made before. This helps to prevent replay attacks when requests are made over a non-secure channel. The nonce value MUST be unique across all requests with the same timestamp, client credentials, and token combinations. Use a unique identifier such as a GUID for this value.. The triPOS server tracks all nonces used in the last five minutes</td>
</tr>
<tr>
<td><strong>Request Date</strong></td>
<td>The request date is the timestamp of when you sent the request to triPOS. The triPOS server will validate that the request was made within plus or minus five minutes of the current server time. For this reason, systems connecting to triPOS should sync up against a global time server to make sure the requests are processed. This date is passed as a string and should be in ISO 8601 format</td>
</tr>
<tr>
<td><strong>Signature</strong></td>
<td>The signature is calculated by following a set algorithm and is derived from a combination of the request method, URL, headers, nonce, request date, request body, developer secret and credential (developer key). The server reconstructs the HMAC signature for each request and verifies that it was able to make an identical signature</td>
</tr>
</tbody>
</table>
Recommended HTTP request header (tp-request-id)

While the **tp-request-id** is optional for legacy purposes, it is recommended that every request made to triPOS contain a **tp-request-id**, which acts as an identifier for the request. See the [Production and Test Mode header samples](#) for an example. Here are some of the benefits:

- The request ID allows an integrator to precisely match each request and response.
- When utilizing triPOS’ manual store and forward capability, the request ID is used to identify the stored transaction to be forwarded.

The following guidelines apply:

- Sent to triPOS in a header named **tp-request-id** on the HTTP request.
- A valid UUID or GUID.
- Unique for each and every request.

Creating an HMAC signature for **triPOS**

The following describes the steps to create an HMAC signature for **triPOS**.

1. **Select an HMAC algorithm**
   Decide which HMAC algorithm to use for the entire request signing process. Preface this with **tp-** and this will be your algorithm to pass into the header:

   **Algorithm**: tp-hmac-sha256

2. **Collect the request method and URL**
   To calculate a signature, start by determining the HTTP method and the URL of the request.

   **e.g. HTTP METHOD**: POST

   The base URL is everything after and including the “/api” in the URL. For example, `http://localhost:8080/api/v1/example?id=1` would end up being: **URL**: /api/v1/example?id=1

3. **Calculate the request body hash**
   If you request contains a body (POST or PUT), then create a hash of the body using the following process:

   1. **UTF-8** encode the body
   2. Compute the hash of the UTF-8 encoded body using the defined algorithm without a key
   3. Hex encode the hash in all lower case

   **Example**
   ```plaintext
   Input: { “Some”:”JSON” }
   Output: 30e7e41efab31a8beaa512ea48aa4d99070496095423d4d653137a7f511eb877
   ```

4. **Generate the canonical signed headers**
   The signed HTTP headers that are sent to **triPOS** should be gathered in alphabetical order:

   1. Iterate through the list of HTTP headers, excluding all the ones starting with **tp-**.
   2. Sort the header keys in alphabetical order
   3. Concatenate all the header keys with a semi-colon between each key, remove any trailing semi-colons
This is the **CanonicalSignedHeaders**.

**Example**

```
accept;accept-encoding;accept-language;connection;cookie;dnt;host;origin;user-agent
```

5. **Generate the canonical headers**

Generate canonical headers by iterating through all of the headers found in the **CanonicalSignedHeaders**:

1. Iterate through the HTTP headers in the same order as the **CanonicalSignedHeaders**.
2. For each key/value in the header, create a new line-delimited string. For multiple entries of the same key, each value should be comma-delimited with a space between each value. For example:
   
   TestHeaderKey:Value1, TestHeaderKey:Value2
   
   should become:
   
   TestHeaderKey:Value1, Value2

**Example**

```
accept:*/*
accept-encoding:gzip, deflate
accept-language:en-US, en; q=0.8
connection:keep-alive
cookie:_test=1
dnt:1
host:localhost:11171
origin:chrome-extension://hgmloofddfndnphfgcellkdbfbjeloo
user-agent:Mozilla/5.0, (Windows NT 6.1; WOW64), AppleWebKit/537.36, (KHTML, like Gecko),
Chrome/37.0.2062.124, Safari/537.36
```

6. **Generate the canonical URI**

The canonical URI starts at “/api” and continues through the start of the query string.

**Example**

```
Input: http://localhost:11171/api/v1/example?exampleArg1=6&exampleArg2=3&exampleArg2=x
Output: /api/v1/example
```

7. **Generate the canonical query string**

The canonical query string starts after the “?” denoting the beginning of the query string and continues through the end of the query string.

**Example**

```
Input: http://localhost:11171/api/v1/example?exampleA=6&exampleC=3&exampleB=x
Output: exampleA=6&exampleB=x&exampleC=3
```

8. **Generate the canonical request**

Generate the canonical request by line-delimiting the HTTP method, the canonical URI, the canonical query string, the canonical headers, the canonical signed headers and the request body hash.

**Example**

```
POST
/api/v1/example
a=3&id=6&y=test&y=sa&Z=x
accept:*/*
accept-encoding:gzip, deflate
```
9. Generate the canonical request hash

Using the same algorithm as before, hex encode a hash of the canonical request:

1. UTF-8 encode the canonical request
2. Compute the hash of the UTF-8 encoded canonical request using the defined algorithm without a key
3. Hex encode the hash in all lower case.

Example

8f6acef4d2fc6ed86eb84c224da441fbd6d39c5ff04f8360ad2b700ff1456544

10. Generate a key signature hash

Using the same algorithm as before, create a hex encoded hash that uses the `nonce` and the `developer secret` as the data and the request date in ISO 8601 format as the key for the hashing algorithm:

1. UTF-8 encode a concatenated value of the `nonce` and the `developer secret`. This is the data.
2. UTF-8 encode the request date in ISO 8601 format. This is the key.
3. Compute the hash of the data using the defined algorithm using the request date key.
4. Hex encode the hash in all lower case.

11. Generate the un-hashed signature

Create a line delimited string using the algorithm, request date, developer key and canonical request hash.

Example

TP-HMAC-SHA256
2014-10-07T23:21:57.2531100Z
5c4ea9fe-9a9b-43d8-8a18-fbody4fac02
8f6acef4d2fc6ed86eb84c224da441fbd6d39c5ff04f8360ad2b700ff1456544

12. Generate the signature

Using the same algorithm as before, create a hex encoded hash that uses the un-hashed signature and the key signature hash:

1. UTF-8 encode the un-hashed signature. This is the data.
2. UTF-8 encode the key signature hash. This is the key.
3. Compute the hash of the data using the defined algorithm using the key.
4. Hex encode the hash in all lower case.

13. Generate the authorization signature header

Create an HTTP header with the key name `tp-authorization`. The value is generated by comma delimiting the following value:

1. Version: The version of the header you are using.
2. Algorithm: The algorithm used to create the hash values. This should be prefaced with `TP-`
3. Credential: This should match your developer key
4. SignedHeaders: The signed headers from Generate canonical signed headers.
5. Nonce: A GUID or unique value that is generated by the client on each request
6. RequestDate: The current time in ISO 8601 format and matches the same time used in generating the un-hashed signature.
7. Signature: The output from Generate the signature.
Example

tp-authorization: Algorithm=TP-HMAC-SHA256, Credential=5c4ea9fe-9a9b-43d8-8a18-fbb3e4fac02, SignedHeaders=accept;accept-encoding;accept-language;connection;cookie;dnt;host;origin;user-agent, Nonce=d4c5f730-9404-4b74-96e2-f4e2a8540954, RequestDate=2014-10-07T23:21:57.2531100Z, Signature=2a5eb585d11d370dbb7229fd477898c41088ef71be5016515da360654e46c9e6
Create a Self-Signed Certificate and Bind it

triPOS can accept HTTPS requests when the useHttps flag is set to true, however the host computer must have a certificate bound the the listening port.

Create the Certificate

Windows 7

IIS must be enabled. If IIS is not installed, please follow these instructions:

1. Click the **Start Button**.
2. Click **Control Panel**.
3. Click Programs, then **click Turn Windows features on or off** (you may be prompted for administrator credentials, provide them and continue).
4. In the list of windows features, select **Internet Information Services**, and then click **OK**.

Once IIS is enabled you can begin creating a self-signed certificate.

1. Click the **Start Button**.
2. Type IIS Manager in the search box.
3. Click on the **Internet Information Services (IIS) Manager** program.
4. On the features page under security double click **Server Certificates**.
5. On the right pane under **Actions** click **Create Self-Signed Certificate**.
6. Create Self-Signed Certificate Window will open, type a name in the text box and select **OK**.
7. Double-Click on the new certificate in the **Server Certificates Window**.
8. The **Certificate Window** will open, click on the **Details** tab.
9. Scroll to the field **Thumbprint**.
10. The **Thumbprint** is displayed in the window in hexadecimal form, this value is used to bind the port later, take note.
Windows 8, 10

1. Open up a Powershell prompt with administrator privileges. Go to the windows search box and type in Powershell.

2. Right-Click on Windows Powershell and select Run as administrator.

The following command will create a SHA-2 certificate on the triPOS host machine.

```
New-SelfSignedCertificate -certstorelocation cert:[cert_location] -dnsname [domain_name]
```

- `[cert_location]`: the location where you want to store this cert (localmachine\my).
- `[domain_name]`: the domain name of the computer where triPOS is installed.

The output will show the Certificate’s Thumbprint, this value is used to bind the certificate, take note.

```
Thumbprint -----------------------
2BB2F1A89DB69FB5FB5D221AE52A943699AFC63B
Subject -----------------------
CN=localhost2
```

Some useful commands for Powershell in regards to certificates:

To view the certificate in the store:
```
Get-ChildItem –Path cert:[cert_location]
```

To remove the certificate type:
```
Remove-Item –Path cert:[cert_location]\[certificate_thumbprint]
```
**Bind the Certificate**

Open a **PowerShell** prompt with **administrator** privileges (if you don’t know how to do this, please review the create certificate for Windows 8, 10 section.)

**Ensure the listening port is not bound.**

Enter the Network Shell:

```
netsh
```

Enter the http shell inside netsh, this shell will be used for the rest of the commands in this section.

```
http
```

Show the URL Action Control List:

```
show urlacl
```

Example Output:

```
Reserved URL   : http://+:8080/api/
  User: EPS\madachi
  Listen: Yes
  Delegate: No
  SDDL: D:(A;;GX;;S-1-5-21-2695982286-1527443976-3846036151-12021)
```

If you see any urls bound to triPOS listening port unbind them with the following step:

```
delete urlacl url=
```

Example Output:

```
PS C:\WINDOWS\system32> netsh http delete urlacl url=http://+:8080/api/
URL reservation successfully deleted
```

Next bind the certificate to the listening port with the following command:

```
add sslcert ipport=0.0.0.0:[listening_port] certhash=[certificate_thumbprint] appid={{[app_id]}}
```

- `[listening_port]`: the listening port for triPOS.
- `[certificate_thumbprint]`: the certificate thumbprint from above.
- `[app_id]`: the application id for triPOS (f6161905-2d97-4ad8-8a25-4ff9ed98e926), be sure to enter it enclosed with curly braces {}.

**Verify Configuration:**

Confirm SSL certificate binding:

```
show sslcert
```

Confirm configuration of URL Action Control List

```
show urlacl
```

If you don’t see the url in the list add it.

```
Add urlacl url=https://*:0.0.0.0:[listening_port]/api/
```
Developer Tools

*triPOS* comes with a number of tools to assist you with integrating to *triPOS*. These include:

- **Online API Documentation**
- **Demo applications**
- **Log Files**

Each of these are discussed below.

**Online API Documentation**

*triPOS* comes with a help file located at: [http://localhost:<port>/api/help](http://localhost:<port>/api/help)

---

**Swagger Documentation** This documentation provides details of each endpoint such as the parameters (header and body) and the response. More details are provided on the following pages.

**What is triPOS®?** contains two links:

- **View details** opens the *triPOS* page on [www.elementps.com](http://www.elementps.com)
- **View Release Notes** opens a page summarizing the additions and changes to the current and past releases of *triPOS*. 

---
**Metadata** This section provides metadata for each web service along with examples in XML, JSON and CVV. Scroll to the bottom of this page to view the XSDS, WSDLs and Plugin Links.

**Swagger Documentation – View API**

Upon clicking the **View API documentation** button, the below screen displays:

Here you will find a list of all available triPOS endpoints. Click on each one to view the details shown below.
There are three tabs available for the request parameters and response values:

Each of these tabs contains essentially the same information, presented in different ways:

- **DTO** provides the syntax for the body of the resource.
- **Model** provides the specifications for each element associated with that resource.
- **Schema** provides a combination of syntax and specifications for each element within the body of the resource.

The following are examples of each of these views for the first few elements in the POST /api/V1/AuthorizationRequest:

**DTO**
Model

Request Class

<table>
<thead>
<tr>
<th>Field</th>
<th>Required</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>laneId</td>
<td>true</td>
<td>int</td>
<td>Specifies which lane to use for the card sale.</td>
</tr>
<tr>
<td>transactionAmount</td>
<td>true</td>
<td>double</td>
<td>The total transaction amount. This is the amount of funds to move on the card</td>
</tr>
<tr>
<td>address</td>
<td>false</td>
<td>POST_Address/api/v1/authorization</td>
<td>The cardholder address information for the transaction.</td>
</tr>
</tbody>
</table>

Scroll down to see more details for fields that contain one or more sub-fields. The figure below shows some of the address sub-fields:

Address

<table>
<thead>
<tr>
<th>Field</th>
<th>Required</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>billingAddress1</td>
<td>false</td>
<td>string</td>
<td>The street address used for billing purposes.</td>
</tr>
<tr>
<td>billingAddress2</td>
<td>false</td>
<td>string</td>
<td>The street address used for billing purposes.</td>
</tr>
<tr>
<td>billingCity</td>
<td>false</td>
<td>string</td>
<td>The name of the city used for billing purposes.</td>
</tr>
</tbody>
</table>

Schema

Request Class

```
module: {  
  "description": "Specifies which lane to use for the card sale.",  
  "type": "int",  
  "required": true
},  
"transactionAmount": {  
  "description": "The total transaction amount. This is the amount of funds to move on the card",  
  "type": "double",  
  "required": true
},  
"address": {  
  "description": "The cardholder address information for the transaction.",  
  "type": "POST_Address/api/v1/authorization",  
  "required": false
},
```

Here is how the address sub-fields are displayed:

Address

```
module: {  
  "billingAddress1": {  
    "description": "The street address used for billing purposes.",  
    "type": "string",  
    "required": false
  },  
  "billingAddress2": {  
    "description": "The street address used for billing purposes.",  
    "type": "string",  
    "required": false
  },  
  "billingCity": {  
    "description": "The name of the city used for billing purposes.",  
    "type": "string",  
    "required": false
  }
},
```
Demo Applications

triPOS is packaged with a set of demo applications. These applications provide sample code for several functionalities that you may need to perform when interacting with triPOS:

1. **Demo Application 1 - DisplaySignature**
   - Displaying or saving a signature image from the given signature byte data

2. **Demo Application 2 - SendRequestAndViewReceipt**
   - Parsing a Card Sale response to obtain and display receipt information
   - Creating a valid, signed header that is required for all requests when not in test mode

The source code is provided for these demo apps, as well as compiled versions you can run immediately.

**Demo Application 1: DisplaySignature**

DisplaySignature is a Windows Forms Application that provides sample code for turning signature byte data into a signature image. The application allows you to select the sample signature byte data format and enter sample signature byte data to view or save the corresponding signature image.

You can choose to test with the sample signature byte data included with the application or enter your own data.

**Procedure**

1. Run the application.
2. Select the appropriate data format and enter the byte data, or select **Populate Sample Data**.
3. Click **Display Signature** to display the signature.
4. Enter a file path and click **Save Signature** to save a bitmap of the signature.

**Errors**

If any exceptions or errors occur while following this procedure, an error message will display in the area below the **Signature Image** box:

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid input Data!</td>
<td>The incorrect format is selected for the given byte data</td>
</tr>
<tr>
<td>or No error message, but a jumbled inaccurate signature image</td>
<td></td>
</tr>
<tr>
<td>Format and Byte Data cannot be empty</td>
<td>A format is not selected, or the <strong>Byte Data</strong> field is empty when <strong>Display Signature</strong> or <strong>Save Signature</strong> is clicked</td>
</tr>
<tr>
<td>The path is not of a legal form</td>
<td>The file path is incorrect or does not exist when trying to save valid signature data</td>
</tr>
<tr>
<td>or A generic error occurred in GDI+</td>
<td></td>
</tr>
</tbody>
</table>
Demo Application 2: SendRequestAndViewReceipt

SendRequestAndViewReceipt is a Windows Forms Application that provides two sets of sample code:

- The first sample demonstrates the process for creating a **valid, signed header** that is required for each request.
- The second sample demonstrates one method for **pulling receipt data** from the triPOS response and displaying a sample receipt.

**Notes**

- This demo application only performs a sample **sale** request. You can choose to test with a sample request in either JSON or XML that is included in the demo application, or you can enter your own sale request.
- The application pulls the **developer key** and **developer secret** from the triPOS.config file of your installed service. These two values are used when calculating the signed header. The path to the triPOS.config file is hardcoded in the sample application to use the default installation path selected by the triPOS installer. If you installed triPOS in a different location, or moved the triPOS.config file, the sample application will not work.
- The application allows you to use a port other than the default 8080, but the port must match the port configured in the triPOS.config file.

**Procedure**

1. Run the application.
2. Verify that your **triPOS.config** file is in the standard location determined by the triPOS installer.
3. Verify that the port number in the path matches the port number in the **triPOS.config** file.
4. Click one of the Sample data buttons to populate the POST body with either an XML or JSON sample sale request.

   **Note** The sample data included with the application is a sale request that uses the PIN pad simulator by default (lane 9999). When this lane number is used, the sample transaction will complete without being connected to a physical PIN pad. If you have an actual PIN pad you would like to use instead, change the lane number in the textbox.

5. Click **Send Request**. The application will:
   - Populate the **Request Header Sent** field with the valid, signed request header that was generated by the application.
   - Populate the **Response** field with the triPOS response.
   - Populate the sample **receipt fields** with values from the triPOS response.
   - Display the **signature bitmap image** that was generated from the response signature byte data if applicable (see the **Signature Values** section below for a list of signature cases and corresponding application behavior.)
**Signature Values**

The application will display different information on the receipt, depending on the card sale flow and how the lane is configured.

The application obtains the *SignatureStatus*Code from the response and follows the logic below:

<table>
<thead>
<tr>
<th>Description</th>
<th>Application Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown</td>
<td>Does not display signature or signature line on the receipt—it is blank</td>
</tr>
<tr>
<td>Signature required</td>
<td>Checks for signature data in the response:</td>
</tr>
<tr>
<td></td>
<td>• If there <strong>IS</strong> signature data, the signature is displayed on receipt</td>
</tr>
<tr>
<td></td>
<td>• If there <strong>NO</strong> signature data, a blank signature line is displayed on the receipt</td>
</tr>
<tr>
<td>Signature present</td>
<td>Displays the signature on receipt</td>
</tr>
<tr>
<td>Signature required, but cancelled by</td>
<td>Displays a blank signature line on receipt</td>
</tr>
<tr>
<td>cardholder on PIN pad</td>
<td></td>
</tr>
<tr>
<td>Signature required, but not supported by</td>
<td>Displays a blank signature line on receipt</td>
</tr>
<tr>
<td>PIN pad</td>
<td></td>
</tr>
<tr>
<td>Signature required, but an error occurred in the</td>
<td>Displays a blank signature line on receipt</td>
</tr>
<tr>
<td>PIN pad</td>
<td></td>
</tr>
<tr>
<td>Signature not required, signature threshold is</td>
<td>Blank - does not display a signature or signature line on the receipt</td>
</tr>
<tr>
<td>above transaction total</td>
<td></td>
</tr>
<tr>
<td>Signature not required by payment type</td>
<td>Blank - does not display signature or signature line on the receipt</td>
</tr>
<tr>
<td>Signature not required by transaction type</td>
<td>Blank - does not display signature or signature line on the receipt</td>
</tr>
</tbody>
</table>

*Note* If lane 9999 is used (the built-in PIN pad simulator) the signature will display as **EPS NULL**. This is a valid signature for the PIN pad simulator and is not an error.
Errors
If any exceptions or errors occur during the above procedure, the error message will be displayed in the area below the Response field. An error will occur in the following cases:

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verify that the port number above matches the port number in triPOS.config</td>
<td>Occurs if the application is unable to reach the triPOS service, either because it is not running or the port number is incorrect</td>
</tr>
<tr>
<td>POST Body must be valid XML or JSON</td>
<td>Occurs if the POST Body is missing, or there is invalid formatting</td>
</tr>
</tbody>
</table>

Log Files
triPOS uses the .Net library log4net to generate log files. These files are useful when debugging your code. triPOS generates the following log files:
- debug
- error
- fatal
- info
- trace
- verbose
- warn

These files are configured to a default maximum file size of 5 MB before they are rolled to a new file. Rolled files will have the naming structure of <logname>.log.1, <logname>.log.2, etc. Files will be retained up to the default maximum of 5 with the exception of the Verbose.log which has a maximum of 20 retained files. Files older than the maximum of 5 or 20 (verbose.log) will automatically be deleted.

The behavior of these files can be configured to meet your needs.

- log4net.config is located in C:\Program Files (x86)\Vantiv\TriPOS Service
- the log files are located in: C:\Program Files (x86)\Vantiv\triPOS Service\Logs

Note: See http://logging.apache.org/log4net/ to learn more about editing log4net.config

To change the number of retained files and/or the file size:
1. Open the log4net.config file.
2. Locate the log
3. Update the following values to fit your business needs:

   <maxSizeRollBackups value="5" />
   <maximumFileSize value="5MB" />
4. Save the changes.
5. Restart triPOS.