

MercuryPay™ 2021

Platform Integration Specifications and Reference Guide

April 2021

MercuryPay Platform

- ✓ Datacap DSI Direct
- ✓ NETePay™/dsiEMVUS®/dsiPDCX®
- ✓ TranCloud®
- ✓ WebServices/JSON Direct
- ✓ HostedCheckout Direct
- ✓ StoreCard™/Gift

**Datacap Systems, Inc. and MercuryPay™ Platform Integration Guide Disclaimer,
Terms of Use and Compliance Representations**

Disclaimer

This **Integration Guide** and all specifications and documentation contained herein or provided to you hereunder (the “Specifications”) are licensed by Datacap Systems, Inc. (“Datacap”) and Worldpay from FIS (individually and collectively, “Licensors”), as applicable to each of the Licensors’ interest in such Specifications, on an “AS IS” basis. For purpose of clarity, Datacap’s Specifications are owned and controlled by Datacap and Worldpay from FIS’s Specifications are owned and controlled by Worldpay from FIS. No representations or warranties are expressed or implied, including, but not limited to, warranties of suitability, quality, merchantability, or fitness for a particular purpose (irrespective of any course of dealing, custom or usage of trade), and all such warranties are expressly and specifically disclaimed. Licensors shall have no liability or responsibility to you or any other person or entity with respect to any liability, loss, or damage, including lost profits whether foreseeable or not, or other obligation for any cause whatsoever, caused or alleged to be caused directly or indirectly by the Specifications. Use of the Specifications signifies agreement with the disclaimer set forth in this paragraph and the below license and restricted use terms and conditions.

Ownership and Restricted Terms of Use

Ownership of all Specifications, related documentation and all intellectual property rights therein and thereto shall remain at all times with Licensors. All rights not expressly granted to you herein are reserved to Licensors. Licensors grant you the right to use the Specification for the sole purpose of transmitting transactions directly to WIP from a merchant transaction originating device. Under no circumstances may you reverse engineer, translate, re-direct, emulate, disseminate to other entities, decompile, adapt, or disassemble the information contained in the Specifications nor shall you attempt to create source code/object code to emulate the MercuryPay platform. You agree that the Specifications and the printed materials and documentation that accompany these Specifications are the confidential information of Licensors and may not be used except as otherwise expressly permitted herein.

Compliance Representations

You represent, warrant and agree:

- To comply with the card brand operating regulations and all applicable PCI Data Security Standards (“PCI DSS”) and Payment Application Data Security Standards (“PA-DSS”) and all requirements applicable to the distribution and use of these Specifications, Worldpay IP products, and your payment application.
- To comply with all applicable federal, state and local laws, rules and regulations, including those related or pertaining to privacy and truncating and masking cardholder account information and data.
- To follow and abide by this integration guide and any modifications made to same from time to time by Licensors. The most recent version of this integration guide can be obtained from a WIP Developer Integrations Analyst by calling 800-846-4472, Ext 1808.
- To distribute and otherwise make available integration upgrades or modifications made by WIP to your reseller and merchant base utilizing any WIP product integrated by you or your payment application.

Any use of your point of sale system or environment to store, process or transmit cardholder data, places that point of sale system and environment within full scope of the PA-DSS. You agree to comply with PCI DSS and, if your point of sale system or environment engages in any of the above activities, to ensure that the point of sale system and environment meets all PA-DSS requirements.

By your use of these Specifications, you warrant, represent and certify your agreement to the above terms and conditions and that your payment application is compliant and adheres to the above requirements and that future versions of your payment application will continue to comply and adhere with these requirements.

MercuryPay StoreCard™ Integration Guide Disclaimer and Terms of Use**Disclaimer**

This **MercuryPay StoreCard™ Integration Guide** and all specifications and documentation contained herein or provided to you hereunder (the "Specifications") are licensed by Worldpay Integrated Payments, LLC on an "AS IS" basis. No representations or warranties are expressed or implied, including, but not limited to, warranties of suitability, quality, merchantability, or fitness for a particular purpose (irrespective of any course of dealing, custom or usage of trade), and all such warranties are expressly and specifically disclaimed. Worldpay Integrated Payments shall have no liability or responsibility to you or any other person or entity with respect to any liability, loss, or damage, including lost profits whether foreseeable or not, or other obligation for any cause whatsoever, caused or alleged to be caused directly or indirectly by the Specifications. Use of the Specifications signifies agreement with the disclaimer set forth in this paragraph and the below license and restricted use terms and conditions.

Ownership and Restricted Terms of Use

Ownership of all Specifications, related documentation and all intellectual property rights therein and thereto shall remain at all times with Worldpay Integrated Payments. All rights not expressly granted to you herein are reserved to Worldpay Integrated Payments. Worldpay Integrated Payments grants you the right to use the Specification for the sole purpose of transmitting transactions directly to Worldpay Integrated Payments from a merchant transaction originating device. Under no circumstances may you reverse engineer, translate, re-direct, emulate, disseminate to other entities, decompile, adapt, or disassemble the information contained in the Specifications nor shall you attempt to create source code/object code to emulate the MercuryPay platform. You agree that the Specifications and the printed materials and documentation that accompany these Specifications are the confidential information of Worldpay Integrated Payments and may not be used except as otherwise expressly permitted herein.

© 2020 - Datacap Systems, Inc. and Worldpay from FIS | Integrated Payments - all rights reserved.

Document Version History

Date	Description
April 2021	<ul style="list-style-type: none"> Removed no longer supported StoreCard Rewards, VoidReward transactions Cleaned up older AcqRefData style and replaced with b [TransactionID]#20Space and e [Response Code]
December 2020 May 2020	<ul style="list-style-type: none"> Corrected Missing COF Recurring image Updated HostedCheckout Transaction Web Services Gift; Truncated test card data Updated reference on <TransactionID> tag with ACH to <AcqRefData>
March 2020	<ul style="list-style-type: none"> Corrected typo on HC mobile src setting; added note on HC ADA compliance
January 2020	<ul style="list-style-type: none"> Updated note on Store and Forward batch size to 9999 per item; Added clarification on HC lowercase password in token requests Updated with Worldpay from FIS Branding Major document consolidation and reorganization ("Encyclopedia Mercurypedia"). Expanded this guide to include broader MercuryPay interface platforms —Datacap Direct, dsiEMVUS, dsiPDCX, TranCloud, WebServices and Gift/StoreCard Now incorporates supplemental guides including Incremental Auth, Pay at the Table, Store and Forward, Payment Account Updater and Check/ACH. Removed no longer supported platform features (i.e. Stand -In).
June 2019	<ul style="list-style-type: none"> Updated FSA healthcare eligible MCC
July 2018	<ul style="list-style-type: none"> Updated with Worldpay rebranding
Sept 2016	<ul style="list-style-type: none"> Removed reference to IDTech VivoPay contactless device (not supported by MercuryPay)
August 2016	<ul style="list-style-type: none"> Clarified definition of <u>VoidSale and Reversal</u>, including updated XML examples
June 2016	<ul style="list-style-type: none"> Rebranded to Vantiv Integrated Payments Added: MToken Error Responses to Appendix 7, Information on dsiEMVUS throughout the document, Information on PCI Qualified Integrators and Resellers (QIR), Clarification on using ZeroAuth with American Express, Device Distributor websites for production equipment Updated: Links to Datacap resources, Mercury Test network information, Fraud Prevention using Card Security Codes and Address Verification, Required fields for BatchClose, Batch settlement times, Appendix 2: List of supported E2E devices, Terms and Conditions page Removed: Check Authorization, Section on Mercury Product Suite
April 2015	<ul style="list-style-type: none"> Added note MercuryPay does NOT support partial reversals. Corrected syntax description for LaneID
Jan 2015	<ul style="list-style-type: none"> Re-organized and re-formatted document for improved clarity. Added: Quick Reference, ZeroAuth, Note on Corporate Cards and Recurring billing Updated: MercuryPay Product Suite, Overview of the Integration Process, Contactless Payments, RefNo description, Corrected typo in PreAuth Reversal Required XML Example, Address Verification Service (AVS), Appendix 1: Payment Industry Rules and regulations, Appendix 8: Summary of Links in this document, Appendix 9: Acronym list Removed: Sections on ActiveX and SOAP integration methods – these are now supplemental guides, Gift details – this is available in a separate guide, Appendix 8: AVS Response codes – this was redundant to the AVS section of the guide, Appendix 9: Certification Network Testing Information (available as a separate document)
March 2014	<ul style="list-style-type: none"> Added reference to MercuryPay's Developer Network Portal. Updated MercuryPay Platform Products and Programs Added information on the LaneID tag throughout all XML examples. Changed test MID's to G2 certification platform in all XML examples Added important notes for recurring tokens Updated Sections: Approved Text Responses, Preventing Fraud by using Card Security Codes and Address Verification, Downgrades: Swiped vs. Manual, Appendix 2: Platform Security Features Using End-to-End Encryption, Appendix 8: Certification Network Testing Corrected comment syntax in XML examples Removed Appendix on Datacap Systems MercuryIP Bridge
January 2014	<ul style="list-style-type: none"> Added information on Gift reloaded per day limit.

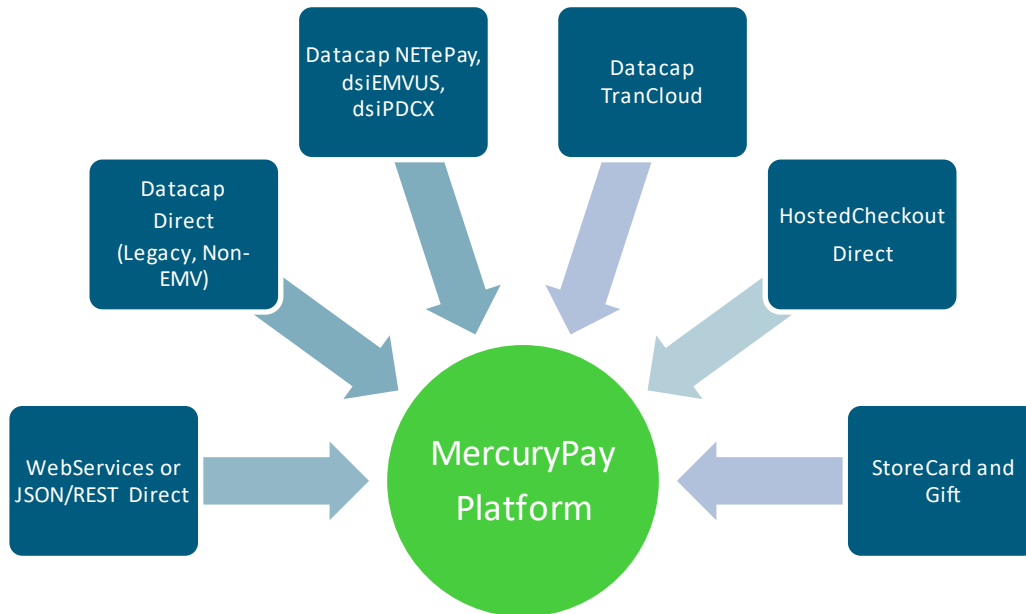
July 2013	<ul style="list-style-type: none">• Re-organized the content• Added clarification regarding Web Services <i>CBatch</i> and <i>CAI/Detail</i> nuances Revised recommendation for PreAuth/PreAuthCapture and Sale/Adjust Requests• Revised paragraph on Partial Authorization• Revised section on Comparing ActiveX and Web Services• Added clarification on FSA Certification and SIGIS registration• Updated the MercuryStand-In™ section• Added <i>Void/Issue</i> to table of supported transactions• Added section on Data Masking in E2E Devices
May 2013	<ul style="list-style-type: none">• Fixed error in code example.
January 2013	<ul style="list-style-type: none">• Re-designed format; reorganized, updated and expanded content:• Updated sections: Overview of Payment Industry Rules and Regulations and Appendix 6: Datacap Systems MercuryIPBridge for Dial Back Up• Expanded sections: Failover between primary and secondary WS servers, US PIN Debit, CardLookUp, Stand-In Authorization• Added sections: Table of Contents, Table of Figures, Store and Forward functionality, Electronic Signature Capture, Device Encryption Keys, Appendix 10: Summary of Links referenced in this document, Appendix 11: List of Acronyms

Table of Contents – Document Navigation

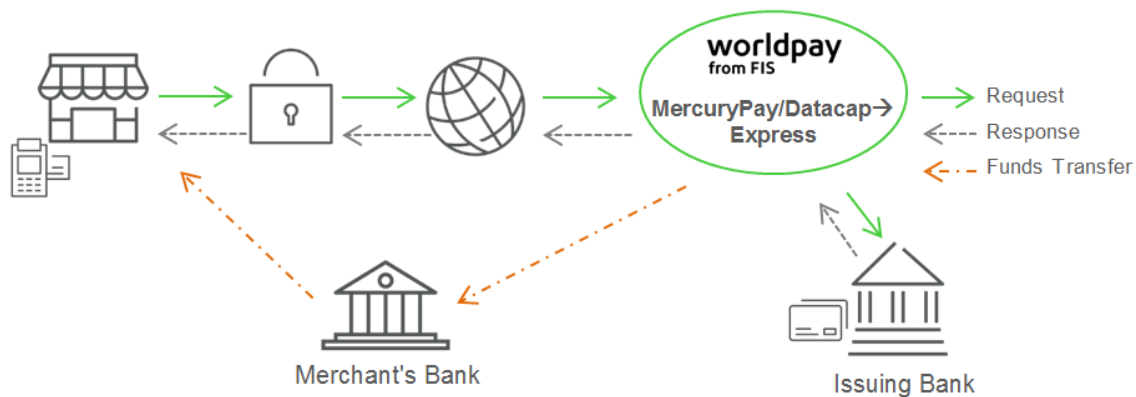
Document Version History	4
Integration Interface Methods to MercuryPay™	8
The MercuryPay™ U.S. Payments Platform	8
Overview	8
MercuryPay Platform and Datacap Systems Features	9
Phases of the Integration Process	11
MercuryPay Integration Methods	13
1. Datacap Direct Interface (DSIClientX).....	13
Check/ACH and Check Commerce	13
ACH Integration Details.....	15
ACH XML Examples.....	17
2. Datacap NETePay and dsiEMVUS/dsiPDCX.....	23
Datacap dsiEMVUS and dsiPDCX—a Quick API Comparison.....	23
NETePay™ 5.07.33 and dsiEMVUS® 1.53/dsiPDCX® 1.73.....	24
Datacap's EMV Testing Application.....	28
EMV Devices, Flow and Functionality with Datacap's dsiEMVUS.....	29
EMV Data Flow and Terminology.....	31
EMV Impacts on Transaction Processing.....	32
Gratuities, ZeroAuth, IncrementalAuth, Cashback and Fallback.....	32
EMV Commands, Functionality, TranCodes and Secure Devices.....	41
Using Datacap's dsiPDCX® as a Complement to dsiEMVUS	43
MercuryPay's Tokenization with dsiEMVUS	44
Details of a dsiEMVUS XML Request and Response.....	44
EMV Receipt Requirements.....	51
Additional MercuryPay/Datacap Features.....	53
Store and Forward Functionality	53
Datacap's Pay-At-The-Table.....	57
Payment Account Updater Management Tool.....	61
Standard Datacap EMV XML Request/Response Examples	71
EMV Credit	71
3. Datacap Systems TranCloud™	89
Overview	89
Supported Devices.....	90
TranCloud Transactions	93
4. Web Services Direct: SOAP or REST/JSON for non EMV.....	94
Overview	94
Web Services SOAP Integration Method Details	95
PaymentsAPI: JSON Remote Procedure Call	106
5. HostedCheckout Direct.....	134
Overview of Worldpay's Integrated Hosted Platforms	134
MercuryPay HostedCheckout.....	134
Cascading Style Sheets (CSS) Functionality	143
Regulatory Compliance and Recommended Best Practices for eCommerce Solutions	162
Additional HostedCheckout/eCommerce Details	165
HostedCheckout Card-on-File/Credentials-on-File (COF)	174
Recurring, Installment, Unscheduled and Cardholder Initiated	175
6. Gift and StoreCard.....	184
Overview	184
StoreCard	184

APPENDICES	233
Appendix 1 – MercuryPay XML Basics	233
Transaction Quick References	233
Appendix 2 – XML Details by TranType	237
Credit	237
FSA	238
DEBIT (U.S. PIN Debit)	238
U.S. EMV	239
Prepaid/Gift and StoreCard	239
EBT	239
ADMIN	240
Appendix 3 – XML Requests and Responses	242
Structure of an XML Request	242
Structure of the Transaction Response	246
Appendix 4 – Industry, TranTypes and TranCodes	251
Overview	251
Credit Transactions and TranCodes	252
U.S. PIN Debit	260
Appendix 5 – Card Branded Prepaid Debit Cards	264
Overview	264
<PartialAuth>	264
Appendix 6 – Merchant Settlement	268
Batch Closing Batch Capture	268
Appendix 7 – MercuryPay Security Features	273
Point-to-Point Encryption and Tokenization	273
MercuryPay Tokenization – MToken	277
Appendix 8 – Duplicate Transactions and TimeOuts	288
Preventing Unintended Duplicate Transactions and TimeOuts	288
Appendix 9 – Fraud Protection	290
Card Security Codes and Address Verification	290
Appendix 10 – Contactless Payments	294
Appendix 11 – Corporate Card Support on MercuryPay	296
Level II data – Tax and CustomerCode	296
Appendix 12 – EBT and FSA on MercuryPay	297
Appendix 13 – Card Security, PCI and Regulations Overview	305
United States Federal and State Laws	306
General Card Data Compliance Summary	309
Appendix 14 – eCommerce Environments	311
Ecommerce Solutions on MercuryPay	311
Appendix 15 – Electronic Signature Capture	315
Overview	315
Appendix 16 – Supported Transactions	317
APPENDIX 17 – MercuryPay Platform, Datacap Errors	318
Error, Exception, and Status Response Messaging	318
Appendix 18—HostedCheckout CSS Functionality	320
Cascading Style Sheets	320
Appendix 19 – List of Acronyms	323

Integration Interface Methods to MercuryPay™



MercuryPay Processes through Express and is a part of the Worldpay from FIS Integrated Payments Network.



The MercuryPay™ U.S. Payments Platform

Overview

This guide provides developers with information on how to integrate to the **MercuryPay™** processing platform. MercuryPay, a part of the Worldpay from FIS Integrated Payments (Worldpay IP) division of FIS, is a long-standing partner with Datacap Systems, Inc. MercuryPay and Datacap share a unique partner-centric focus on integration requirements, transaction types, and features, all grounded in Datacap's specifications.

- Where MercuryPay is the front-door to the broader Worldpay processing back end networks, issuers, payment brands and settlement systems, Datacap's specifications offer the structure and language to integrate to MercuryPay.
- Integrating to **MercuryPay** is driven by your preferred integration method supported by the platform—be that using Datacap's locally installed NETePay™ and client suite (dsiClientX®, dsiEMVUS®, and

dsiPDCX®), Datacap's EMV-ready Payment hub TranCloud™ or using one of the platforms direct integration methods such as Web Services or Hosted Checkout.

- **MercuryPay** receives transaction requests from the POS over the internet, routes these through a "transaction broker" to our Express Gateway and on to the back-end networks, obtains the authorization response, and passes the response data back to the POS. At the heart of **MercuryPay** is Datacap Systems, Inc. "ePay" software: **NETePay** is used to process credit/debit payments, and **GIFTePay™** is used to support services from *MercuryGift* and *MercuryPay StoreCard*.

This guide contains transactions payload details, information on platform integration methods, features and processing requirements. Your business model and target market will drive the features and functionality of your integrated solution.

MercuryPay offers multiple integration methods for credit and gift integrations:

- **Datacap NETePay and Client Suite**; locally installed distributed software components for secure EMV processing and multiple EMV device support enablement.
- **Datacap TranCloud®** locally installed hardware application "hub" for mobile and browser-based EMV processing.
- **Web Services SOAP or REST/JSON Remote Procedure Calls** for non-EMV to process via our Web Services and PaymentsAPI servers directly.
- **HostedCheckout** iFrame or redirect hosted solution for eCommerce/card not present transactions
- **Datacap Direct**—using Datacap's legacy, in-scope, locally installed dsiClientX for non-EMV processing on Windows based applications to hit our servers directly. Check Commerce Check/ACH is also supported.

MercuryPay and Datacap Systems

IMPORTANT: Integrators are encouraged to be familiar with the Datacap implementation and specifications as a basis for any MercuryPay integration work. For a successful integration, we recommend creating an account on Datacap Systems Developer Portal at <https://www.dsidevportal.com/> and using these APIs, sample XML, quick starts and device details in your development work. Your Worldpay IP Implementations Consultant will assist with additional information on testing and certifying your integration.

MercuryPay Platform and Datacap Systems Features

- Comprehensive transaction support for US EMV, magstripe and contactless Credit, US Debit, Canadian EMV, Gift, EBT and FSA. Support includes both card present and card not present environments using **HostedCheckout** or **Web Services**.
- Server redundancy in multiple geographic locations
- Partnership with Datacap Systems, Inc. and their suite of ePay products and client controls:
 - **DSIClientX®** supports non-EMV legacy direct to MercuryPay transaction routing. This legacy client control can also be used to implement new MercuryPay features for cards on file/Account Updater and Check Commerce Check/ACH.
 - **dsiEMVUS®** supports processing EMV transactions in the United States and includes such attractive features as Store and Forward, Pay At The Table, Incremental Authorizations.
 - **dsiPDCX®** for processing Gift/StoreCard, FSA, EBT.
 - **TranCloud®** is a web-based service "hub" using Datacap's Mobile and browser-based architecture.
 - **GIFTePay™** supports our **Gift** and **StoreCard** solutions
 - **DSIEMVClientX®** supports processing EMV transactions in Canada

- **NETePay™** "one-to-many" integration solution; PA-DSS Validated; includes a locally distributed "In Store" version and a server-side/hosted processing environment supported by MercuryPay.
- **Point-to-point-Encryption (P2PE)** is used to encrypt card data at the point of entry. This encrypted data can only be decrypted by MercuryPay's secure servers. Worldpay from FIS is a level one service provider and as such, maintains secure encryption environments to industry standards. Our partnerships with device manufacturers, Key Injection Facilities (KIF) and industry-leading distributors allow us to maintain security compliance and quality throughout the entire supply chain of device procurement. The use of MercuryPay's point-to-point encryption is non-validated and is PCI scope-reducing.
- **MToken™**, MercuryPay's tokenization technology works hand-in-hand with encryption, returning a token to the local POS in place of sensitive card data. A new token is generated with every transaction. Tokens are perfect for follow up transactions – captures, reversals, adjusts, tip modification; tokens may be stored and used for recurring billing and card-on-file. Tokens usage can be "grouped" across merchants that share common underwriting attributes like business owner, legal name and Tax ID. MToken, combined with encryption are preferred for scope reduction.
- Developers can choose from multiple supported integration interface methods to send their transaction payload to MercuryPay for card-present and card-not-present environments—including distributed Windows-based technologies from Datacap Systems, Datacap Client direct, web-based methods direct using Web Services **SOAP** or **REST/JSON**, **HostedCheckout** and using the mobile/browser-based architecture of Datacap's TranCloud.

PCI/PA-DSS and Card Data Security

MercuryPay/Worldpay from FIS is a long-standing, PCI validated, Level One Service Provider, listed on Visa's Global Registry of validated Service Providers and MasterCard's SDP Program (Site Data Protection) of compliant Service providers. MercuryPay is audited quarterly to maintain the highest level of platform security and compliance.

- All payment applications processing with MercuryPay are required to adhere to PA-DSS/PCI DSS compliance standards.
- Worldpay's compliance team offers assistance in understanding PCI DSS and all merchants are required to maintain compliance using Worldpay's OmniShield Assure program.
- Integrating to PA-DSS/PCI Security Standards from the onset assures a preferred combination of compliance integrity and market advantage.

It has become increasingly important for all participants in the payments industry to become vigilant in understanding and complying with all card brands, State and Federal industry regulations, and Payment Application Card Data Security (PA-DSS) requirements. Before a POS developer embarks on the process of building a POS, it is **absolutely critical** to have a solid working knowledge of why these regulations are in place and how complying with these regulations helps to mitigate potential payment processing risks and vulnerabilities.

Merchants are vulnerable to fraud, malware and being hacked. ***It is the responsibility of the POS integrator to understand and comply with all industry laws, regulations and card data security standards.*** The entire industry is working on tightening unwarranted and criminal access to unencrypted card data, clear text files, identity information, open WIFI networks, un-segmented internet networks, and all manner of stored data files. Payment applications that expose any of these vulnerabilities are unacceptable by the current industry standards.

Developers who wish to board merchants and process with MercuryPay are required to either:

- Confirm validation with a certified Payment Application Qualified Security Assessor (PA-QSA),
- Use an existing reduced scope, compliant application or
- Demonstrate a reduction-of-scope compliance path by implementing a combination of Point-to-Point Encryption (P2PE/E2E) and Tokenization.

There are multiple governing bodies overseeing the payments industry regulations:

- PCI Security Standards Council (PCI SSC),
- State and Federal legislation, and
- The card brands (Visa, MasterCard, Discover and American Express).

Important: It is an industry standard to review, understand and comply with these regulations as they apply to your payment application and as your POS system is implemented with your merchant base.

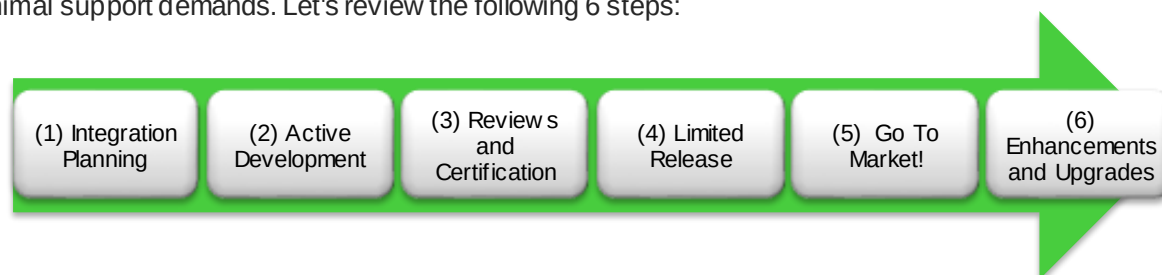
Integration Services

Worldpay and Datacap will work with you during your integration and provide a number of resources to help make the process as smooth as possible:

- **Dedicated Implementations Consultant** Our Consultants are experts in payment industry nuances and will provide you with industry knowledge, technical guidance and best practices.
- **Worldpay's Developer Engine.** Designed for software engineers, <https://developerengine.fisglobal.com/> and <https://developerengine.fisglobal.com/additional-products> for MercuryPay
- **Sample Projects** Datacap maintains sample projects on their developer portal under the Quick Start tab at <https://www.dsidevportal.com/quick-start/>.
- **Datacap Developer Portal** maintained by Datacap Systems, Inc. and contains extensive resources to jump start your integration work. To create an account and log in at <https://www.dsidevportal.com/>
- **Test and Certification Services using MercuryCert.Net:** A testing environment that emulates the Worldpay from FIS/MercuryPay/Express platforms; includes access to test devices and test cards. Testing credentials are available from your Integration Consultant.

Phases of the Integration Process

Although integrations are unique, the overall process falls into phases proven to bring about an accelerated time to market and return on investment while delivering a card brand industry compliant POS application with minimal support demands. Let's review the following 6 steps:



1. Integration Planning

You will be assigned a Developer Implementations Consultant to assist with your integration. As a first step, Worldpay IP will arrange a kickoff discussion call to help introduce the planning process. A Scope of Work (SOW) document will be drafted to guide the integration work. The SOW will include:

- Deciding which features and functionalities you wish to implement. This may drive the programming language or integration method best suited for these features.
- Understanding the specific requirements for your target industry/business.
- Deciding which integration method best fits your native POS architecture.
- Reviewing MercuryPay's integration specifications, the XML request/response format, and error response handling.
- Reviewing the payment industry rules and regulations, and consulting with a qualified security assessor (QSA) on Card Data and Payment Application Security Standards.

After the kickoff call, your Implementations Consultant will provide you a set of materials to help you get started.

- A copy of this integration guide

- Supplements to this integration guide that are specific to your chosen integration method
- Testing Guide - contains information on connecting to our certification network
- Transaction Review Test script: provides a variety of processing scenarios to ensure your POS functions as planned.

Regular check-ins with your Consultant will be arranged to stay on track throughout the integration process.

2. Active Development and Testing

Combine your software code with the tools and sample code from our integration specifications. Test your solution against the MercuryCert.Net certification environment.

- Follow the XML formatting guidelines from this document and the **Datacap Developer Portal** available at <http://www.dsidevportal.com>.
- Program and test your solution using the MercuryPay certification network.

3. Reviews and Certification

Once your integration work is complete, we will perform a series of transaction reviews to make sure your POS is functioning properly, that any new functionality is understood and your POS is ready to go to market.

- **Transaction Review:** Complete and submit your Transaction Review Test script to your Implementations Consultant. Your consultant will review the test results and provide a quick turnaround, identifying any issues that need to be addressed before we can issue the Transaction Review Certification letter.

4. Limited Release

Once certification is completed, your POS system may "soft launch" up to five merchants into live production. Your Implementations consultant will work with you to ensure that your merchant installations run smoothly. We will review your system topology for a typical merchant location including both physical hardware and software components, and provide feedback on your implementation/setup documentation. Upon completion of this phase your product will be ready for release into full production.

5. Go-To-Market

After you release your product into full production, Worldpay from FIS provides you with ongoing resources and support, including:

- An internal sales force, which can help you grow your channel
- A 24-7 customer support team to assist your merchants with common questions and processing concerns
- Knowledgeable technical support teams to help troubleshoot and resolve more complex issues.

6. Enhancements and Upgrades

Releasing your POS into production introduces you to a profitable product life cycle and a broad range of future possibilities. It is the beginning, not the end of your relationship with Worldpay from FIS. As new industry standards, new technologies and card brand mandates enter the payment space, the Worldpay from FIS platforms must evolve to support them—and so must the integrated POS application. Enhancements may include:

- EMV adoption or next adding the generation of PIN Pads with enhanced features.
- Prepaid Gift and StoreCard™, which drives merchant loyalty and increases sales.
- Extend your POS appeal and increase merchant profits by implementing a new form factor such as "line-busters," unattended Kiosks, Pay at the Table.
- Adding any of the latest Datacap/MercuryPay features such as Store and Forward, IncrementalAuth, Check/ACH, Account Updater, Credentials on File/Card on File.

Your integration consultant can help you with all of these things, and assist you with their expertise of the platform, transaction review, certification and go to market readiness.

MercuryPay Integration Methods

The remainder of this guide and a rich appendix of supporting documentation will review the following integration methods to the MercuryPay platform:

- Datacap Direct Interface (DSIClientX®)
- Datacap NETePay and dsiEMVUS/dsiPDCX
- Datacap TranCloud
- Web Services Direct SOAP or JSON Remote Procedure Calls for non EMV
- HostedCheckout Direct
- Gift and StoreCard utilizing MercuryPay hosted GIFTePay™

Note: Datacap Canadian Integrations and eWIC are detailed in supplemental specifications.

1. Datacap Direct Interface (DSIClientX)

The Datacap Direct Interface option using a locally installed Windows Active-X control, the DSIClientX 3.86, dates from 2001 and is Datacap's longstanding in-scope interface for NETePay and GIFTePay software. This interface is available for developers who wish to manage, maintain and validate their own cardholder data environment directly including any direct device interaction.

Historically, integration to the DSIClientX was the developer's portion of Datacap System's client-server, one-to-many architecture that was eventually adopted by multiple processors and evolved into the MercuryPay platform—and the "Pay" in MercuryPay is an honorable nod to Datacap's hosted NETePay server. Building upon the easily accessible XML formatted requests and responses, hosting the NETePay and expanding upon its strengths evolved into the broader MercuryPay multi-merchant NETePay infrastructure.

Because the DSIClientX was historically fundamental to understanding the XML client-server exchange of transaction requests and responses, it is where integrators learned to understand the basics of payment processing with Datacap on MercuryPay. Since the DSIClientX was developed prior to EMV and the industry's current reliance on peripheral devices to control the encryption, chip interaction and security compliance of the transaction request, its adoption has been more limited; it is nonetheless still made available as a supplemental control to the dsiEMVUS and dsiPDCX.

- For specification details, contact Datacap Systems, Inc. or visit their website at <https://datacapystems.com/>.
- For a quick reference to XML basics and a review of supported transactions please review XML Basics and Elements of a Transaction located in the appendix.

Check/ACH and Check Commerce

Check/ACH in collaboration with our third party ACH platform, **Check Commerce**, is one of the newest additions to the MercuryPay/Datacap Direct platform. Check/ACH is only available via a DSIClientX Direct integration. Where MercuryPay routes ACH check transactions to the Check processing host, Check Commerce underwrites the merchant account and processes the transactions as a gateway end-point.

- Using a Datacap DSI Client ActiveX direct integration, MercuryPay routes electronic check payments to Worldpay's third party ACH processing platform, **Check Commerce**.
- Merchants wishing to accept check payments via MercuryPay will contact their sales representative to help facilitate the opening of a check processing account with **Check Commerce**.
- Merchants will then contact **Check Commerce** directly by emailing ach@checkcommerce.com for operational Check/ACH support.

Supported Check Processing Transactions

ACH Transaction	Description
Authorization	Performs an ACH Debit.
VerificationOnly	Used in conjunction with Authorization transaction with an additional tag <ACHMethod>; Verification only is used to verify an ACH Checking account is valid but customer funds are not moved.
Return	Performs an ACH Refund using a prior successful Authorization TransactionID. Transaction amount must be for the full amount of the original ACH Authorization only (partial refunds are not allowed).
Void	Performs a void; requires using the prior successful Authorization AcqRefData value (Transaction ID value) and matching InvoiceNo.
Reversal	Performs a Reversal; use in the event of a system or timeout issue; requires using the prior successful Authorization AcqRefData value (Transaction ID value) and matching InvoiceNo.
Query	Queries the transaction status of an ACH transaction (Pending, Queued, Originated and Settled).

Supported Check/ACH Payment Types

Merchants may accept Check/ACH payments for various purposes, including recurring billing, non-face-to-face convenience payments, or face-to-face payments at the point of sale. The following check/ACH code types are supported by Check Commerce:

- Pre-arranged Payment and Deposit (PPD)
- Cash Concentration and Disbursement (CCD)
- Account Receivable (ARC)
- Back Office Conversion (BOC)
- Point of Purchase (POP)
- Internet-Initiated (WEB)
- Telephone-Initiated (TEL)

Additional details regarding these requirements are available directly from Check Commerce via email at ach@checkcommerce.com.

Process/Status Flow of a Check Transaction

There is a standard process flow that Check/ACH transactions follow which typically takes place over several days. ACH Authorization transactions go through the same process flow as if a merchant had accepted a physical check that they then deposit at their bank.

- When an ACH Authorization is submitted, it will first go through some basic validation and verification processes at Check Commerce. If the transaction passes that verification, it will be accepted by Check Commerce and the processing platform will return a successful (**Approved/Accepted**) response in real-time. At that point, the transaction will be in a **"Pending"** transaction status.
- Later that day, the transaction status will move to a **"Queued"** status and then to an **"Originated"** status shortly thereafter.
- Once in an **Originated** status, this indicates that the ACH transaction has been submitted to the Federal Reserve for processing. Depending on the funding schedule the merchant has configured with Check Commerce, the transaction may stay in this status for several days.
- At the time of funding, Check Commerce updates the status accordingly which is indicated as a **"Settled"** status. This is the last step in the transaction flow and indicates that funds for the transaction are being deposited into your bank account.

The table below outlines this standard processing flow as the status of the transaction is processed.

Transaction Status	What does this status mean?
Pending	This means the transaction is waiting to be processed by Check Commerce. When you process a transaction in the morning, as an example, it will be in a Pending status until later in the afternoon. <ul style="list-style-type: none"> Transactions in a Pending status can be reversed or voided, but cannot be refunded.
Queued	This means the transaction has been downloaded and processed by Check Commerce later in the afternoon. <ul style="list-style-type: none"> Transactions in a Queued status cannot be reversed, voided, or refunded.
Originated	This means the transaction has been sent off to the Fed for ACH processing. The transaction will stay in this status until a Fed response is received (up to 5 business days). <ul style="list-style-type: none"> Transactions in an Originated status cannot be reversed, voided, or refunded.
Settled	This means the transaction has been funded by Check Commerce. <ul style="list-style-type: none"> Transactions in a Settled status can be refunded, but cannot be reversed or voided.

ACH Query—Check Status Response Values

Use ACH Query to determine the status of ACH payments. In addition to the standard Check Commerce transaction status values described above, (pending, queued, originated or settled), additional values described below may also be returned in the ACH Query response:

Transaction Status	Description
Voided	The original transaction has been voided.
Returned	The original transaction has been returned by the bank (e.g. Non-Sufficient Funds, etc.) Note: Whenever a transaction moves from an Originated or Settled status to a different status such as Returned or Error, the merchant should contact the customer to ask for another form of payment.
Pending	This means the transaction is waiting to be processed by Check Commerce. At this point, the transaction can be reversed or voided, but cannot be refunded.
Queued	This means the transaction has been processed by Check Commerce. At this point, the transaction cannot be reversed, voided, or refunded.
Unknown	Unknown error.
Error	The transaction may have been rejected with an error during the initial real-time request, or the transaction may have been rejected later in the process. Note: Whenever a transaction moves from an Originated or Settled status to a different status such as Returned or Error, the merchant should contact the customer to ask for another form of payment.
Originated	This means the transaction has been sent off to the Fed for ACH processing. The transaction will stay in this status until a Fed response is received (up to 5 business days). At this point, the transaction cannot be reversed, voided, or refunded.
Settled	This means the transaction has been funded by Check Commerce. At this point, the transaction can be refunded, but cannot be reversed or voided.

ACH Integration Details

Required and Optional Fields of an ACH XML Request/Response

The XML examples below highlight the **required** and **optional** (gray) fields considered the minimally required XML tags in support of ACH on MercuryPay.

- Contact your integration consultant for MerchantID.
- For certification testing, please use **RoutingNo** of 101002855 and the ACH **AcctNo** of 555000555

XML: ACH Example Request

```
<TStream>
<Transaction>
```

```

<TranType>ACH</TranType>    <!-- note new ACH TranType value -->
<TranCode>Authorize</TranCode>    <!-- Also Return, Void, Reversal, Verification and Query -->
<InvoiceNo>001422</InvoiceNo>
<MerchantID>62589001=TEST</MerchantID>    <!-- note ACH must be enabled on the MerchantID -->
<OperatorID>Op1</OperatorID>
<RefNo>001422</RefNo>    <!-- filler value only on the request -->
<Account>
    <RoutingNo>101002855</RoutingNo>    <!-- note new ACH/Check routing value -->
    <AcctNo>555000555</AcctNo>    <!-- note ACH/Check Account value -->
    <AcctType>Checking</AcctType><!-- note Account Type 'Checking' and 'Savings' -->
    <DriversLicense>CO12345</DriversLicense>
    <StateCode>CO</StateCode>
    <AltCustID>12345</AltCustID>
</Account>
<Amount>
<Purchase>41.70</Purchase>
</Amount>
<DemographicInfo>
    <CustFirstName>John</CustFirstName><!-- First and Last Name Required -->
    <CustLastName>Smith</CustLastName><!-- First and Last Name Required -->
    <CustAddr1>22 West Way</CustAddr1>
    <CustAddr2>#3</CustAddr2>
    <CustCity>Durango</CustCity>
    <CustState>CO</CustState>
    <CustZip>81301</CustZip>
    <CustPhone>9703331234</CustPhone>
    <DateOfBirth>01011962</DateOfBirth>
    <Ssn4>9620</Ssn4>
</DemographicInfo>
</Transaction>
</TStream>

```

XML ACH Example Response

```

<RStream>
<CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Approved</TextResponse>
    <UserTraceData></UserTraceData>
</CmdResponse>
<TranResponse>
    <MerchantID>62589001</MerchantID>
    <AcctNo>XXXXXX55</AcctNo>
    <CardType>ACH</CardType>
    <TranCode>Authorize</TranCode>
    <AuthCode>348-154</AuthCode><!-- note AuthCode with hyphen -->
    <RefNo>00125</RefNo>    <!-- Processor value in the Response -->
    <InvoiceNo>00125</InvoiceNo>
    <OperatorID>Op 1</OperatorID>
    <Amount>
    <Purchase>20.00</Purchase>
    <Authorize>20.00</Authorize>
    </Amount>
    <AcqRefData>23639914</AcqRefData>    <!-- note Transaction ID used for Reversals, Voids and Queries -->
</TranResponse>
</RStream>

```

ACH XML Examples

ACH Authorize

The *ACH Authorize* performs a consumer verification and ACH debit from the specified account. Use **ACH Query** to confirm the status (i.e. Pending, Queued, Originated and Settled) of the ACH transaction request.

Request:

```
<TStream>
  <Transaction>
    <TranType>ACH</TranType>
    <TranCode>Authorize</TranCode>
    <InvoiceNo>1620223</InvoiceNo>
    <MerchantID>62589001=TEST</MerchantID>
    <RefNo>1</RefNo>
  <Account>
    <RoutingNo>101002855</RoutingNo>
    <AcctNo>555000555</AcctNo>
    <SequenceNo>0005</SequenceNo>
    <AcctType>Checking</AcctType>
    <DriversLicense>CO12345</DriversLicense>
    <StateCode>CO</StateCode>
    <AltCustID>12345</AltCustID>
  </Account>
  <Amount>
    <Purchase>10.00</Purchase>
  </Amount>
  <DemographicInfo>
    <CustFirstName>John</CustFirstName>
    <CustLastName>Smith</CustLastName>
    <CustAddr1>22 West Way</CustAddr1>
    <CustAddr2>#3</CustAddr2>
    <CustCity>Durango</CustCity>
    <CustState>CO</CustState>
    <CustZip>81301</CustZip>
    <CustPhone>9703331234</CustPhone>
    <DateOfBirth>01011962</DateOfBirth>
    <Ssn4>9620</Ssn4>
  </DemographicInfo>
</Transaction>
</TStream>
```

Response:

```
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Approved</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>62589001</MerchantID>
    <AcctNo>XXXXXXX55</AcctNo>
    <CardType>ACH</CardType>
    <TranCode>Authorize</TranCode>
    <AuthCode>153-685</AuthCode>
    <RefNo>1</RefNo>
    <InvoiceNo>1620223</InvoiceNo>
  <Amount>
    <Purchase>10.00</Purchase>
    <Authorize>10.00</Authorize>
```

```

</Amount>
<AcqRefData>860478545</AcqRefData>
</TranResponse>
</RStream>

```

Authorize with VerificationOnly

ACH Authorization with *VerificationOnly* is used to verify a checking account but does not initiate a purchase. (The ACH Authorization performs a consumer verification and ACH debit from the specified account.) The *VerificationOnly* is sent with an additional tag **<ACHMethod>** and does not use the **<DateOfBirth>**.

Request:

```

<TStream>
  <Transaction>
    <TranType>ACH</TranType>
    <TranCode>Authorize</TranCode>
    <ACHMethod>VerificationOnly</ACHMethod>
    <InvoiceNo>1620223</InvoiceNo>
    <MerchantID>62589001=TEST</MerchantID>
    <RefNo>1</RefNo>
  </Transaction>
  <Account>
    <RoutingNo>101002855</RoutingNo>
    <AcctNo>555000555</AcctNo>
    <SequenceNo>0005</SequenceNo>
    <AcctType>Checking</AcctType>
    <DriversLicense>CO12345</DriversLicense>
    <StateCode>CO</StateCode>
    <AltCustID>12345</AltCustID>
  </Account>
  <Amount>
    <Purchase>10.00</Purchase>
  </Amount>
  <DemographicInfo>
    <CustFirstName>John</CustFirstName>
    <CustLastName>Smith</CustLastName>
    <CustAddr1>22 West Way</CustAddr1>
    <CustAddr2>#3</CustAddr2>
    <CustCity>Durango</CustCity>
    <CustState>CO</CustState>
    <CustZip>81301</CustZip>
    <CustPhone>9703331234</CustPhone>
    <Ssn4>9620</Ssn4>
  </DemographicInfo>
</Transaction>
</TStream>

```

Response:

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Approved</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>62589001</MerchantID>
    <AcctNo>XXXXXXX55</AcctNo>
    <CardType>ACH</CardType>
    <TranCode>Authorize</TranCode>
    <AuthCode>153-685</AuthCode>
  </TranResponse>
</RStream>

```



```

<RefNo>1</RefNo>
<InvoiceNo>1620223</InvoiceNo>
<Amount>
  <Purchase>10.00</Purchase>
  <Authorize>10.00</Authorize>
</Amount>
<AcqRefData>860478545</AcqRefData>
</TranResponse>
</RStream>

```

ACH Void

Use ACH Void to cancel a previous ACH Authorization or ACH Return transaction that is still in a "Pending" status. ACH Void requires Authorization AcqRefData value (Transaction ID value) and matching InvoiceNo.

Request:

```

<TStream>
  <Transaction>
    <TranType>ACH</TranType>
    <TranCode>Void</TranCode>
    <InvoiceNo>1620223</InvoiceNo><! - - note AcqRefData and Invoice must match - - >
    <MerchantID>62589001=TEST</MerchantID>
    <RefNo>1</RefNo>
    <AcqRefData>860478545</AcqRefData><! - - note AcqRefData and Invoice must match - - >
  <Account>
    <RoutingNo>101002855</RoutingNo>
    <AcctNo>555000555</AcctNo>
    <SequenceNo>0005</SequenceNo>
    <AcctType>Checking</AcctType>
    <DriversLicense>CO12345</DriversLicense>
    <StateCode>CO</StateCode>
    <AltCustID>12345</AltCustID>
  </Account>
  <Amount>
    <Purchase>10.00</Purchase>
  </Amount>
  <DemographicInfo>
    <CustFirstName>John</CustFirstName>
    <CustLastName>Smith</CustLastName>
    <CustAddr1>22 West Way</CustAddr1>
    <CustAddr2>#3</CustAddr2>
    <CustCity>Durango</CustCity>
    <CustState>CO</CustState>
    <CustZip>81301</CustZip>
    <CustPhone>9703331234</CustPhone>
    <DateOfBirth>01011962</DateOfBirth>
    <Ssn4>9620</Ssn4>
  </DemographicInfo>
</Transaction>
</TStream>

```

Response:

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Approved</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
</TranResponse>

```

```

<MerchantID>62589001</MerchantID>
<AcctNo>XXXXXXXXX55</AcctNo>
<CardType>ACH</CardType>
<TranCode>Void</TranCode>
<AuthCode>Voided</AuthCode>
<RefNo>1</RefNo>
<InvoiceNo>1620223</InvoiceNo>
<Amount>
  <Purchase>10.00</Purchase>
  <Authorize>10.00</Authorize>
</Amount>
<AcqRefData>860478546</AcqRefData>
</TranResponse>
</RStream>

```

ACH Reversal

Use ACH Reversal to cancel a previous ACH Authorization that is in an unknown or "Pending" status. Use cases for ACH Reversal include system and or network Timeouts. *ACH Reversal* requires Authorization AcqRefData value (Transaction ID value) and matching InvoiceNo.

Request:

```

<TStream>
<Transaction>
  <TranType>ACH</TranType>
  <TranCode>Reversal</TranCode>
  <InvoiceNo>1620223</InvoiceNo><!-- note AcqRefData and Invoice must match -->
  <MerchantID>62589001=TEST</MerchantID>
  <RefNo>1</RefNo>
  <AcqRefData>860478545</AcqRefData><!-- note AcqRefData and Invoice must match -->
</Account>
  <RoutingNo>101002855</RoutingNo>
  <AcctNo>555000555</AcctNo>
  <SequenceNo>0005</SequenceNo>
  <AcctType>Checking</AcctType>
  <DriversLicense>CO12345</DriversLicense>
  <StateCode>CO</StateCode>
  <AltCustID>12345</AltCustID>
</Account>
<Amount>
  <Purchase>10.00</Purchase>
</Amount>
<DemographicInfo>
  <CustFirstName>John</CustFirstName>
  <CustLastName>Smith</CustLastName>
  <CustAddr1>22 West Way</CustAddr1>
  <CustAddr2>#3</CustAddr2>
  <CustCity>Durango</CustCity>
  <CustState>CO</CustState>
  <CustZip>81301</CustZip>
  <CustPhone>9703331234</CustPhone>
  <DateOfBirth>1962</DateOfBirth>
  <Ssn4>9620</Ssn4>
</DemographicInfo>
</Transaction>
</TStream>

```

Response:

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>

```

```

<DSIXReturnCode>000000</DSIXReturnCode>
<CmdStatus>Approved</CmdStatus>
<TextResponse>Approved</TextResponse>
<UserTraceData></UserTraceData>
</CmdResponse>
<TranResponse>
  <MerchantID>62589001</MerchantID>
  <AcctNo>XXXXXX55</AcctNo>
  <CardType>ACH</CardType>
  <TranCode>Reversal</TranCode>
  <AuthCode>397-548</AuthCode>
  <RefNo>1</RefNo>
  <InvoiceNo>1620223</InvoiceNo>
  <Amount>
    <Purchase>10.00</Purchase>
    <Authorize>10.00</Authorize>
  </Amount>
  <AcqRefData>860478546</AcqRefData>
</TranResponse>
</RStream>

```

ACH Return

Use ACH Return to credit a full refund of a previous ACH Authorization transaction. Partial refunds are not supported. ACH Return can only be used on transactions with a Queued, Originated or Settled status.

Request:

```

<TStream>
<Transaction>
  <TranType>ACH</TranType>
  <TranCode>Return</TranCode>
  <InvoiceNo>1525988</InvoiceNo>
  <MerchantID>62589001=TEST</MerchantID>
  <RefNo>1</RefNo>
<Account>
  <RoutingNo>101002855</RoutingNo>
  <AcctNo>555000555</AcctNo>
  <SequenceNo>0002</SequenceNo>
  <AcctType>Checking</AcctType>
  <DriversLicense>CO12345</DriversLicense>
  <StateCode>CO</StateCode>
  <AltCustID>12345</AltCustID>
</Account>
  <Amount>
    <Purchase>10.00</Purchase>
  </Amount>
<DemographicInfo>
  <CustFirstName>John</CustFirstName>
  <CustLastName>Smith</CustLastName>
  <CustAddr1>22 West Way</CustAddr1>
  <CustAddr2>#3</CustAddr2>
  <CustCity>Durango</CustCity>
  <CustState>CO</CustState>
  <CustZip>81301</CustZip>
  <CustPhone>9703331234</CustPhone>
  <DateOfBirth>01011962</DateOfBirth>
  <Ssn4>9620</Ssn4>
</DemographicInfo>
</Transaction>
</TStream>

```

Response:

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Approved</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>62589001</MerchantID>
    <AcctNo>XXXXXX55</AcctNo>
    <CardType>ACH</CardType>
    <TranCode>Return</TranCode>
    <AuthCode>643-496</AuthCode>
    <RefNo>1</RefNo>
    <InvoiceNo>1525988</InvoiceNo>
    <Amount>
      <Purchase>10.00</Purchase>
      <Authorize>10.00</Authorize>
    </Amount>
    <AcqRefData>860478542</AcqRefData>
  </TranResponse>
</RStream>

```

ACH Query

Use ACH Query to confirm the transaction status of an ACH transaction (**Pending, Queued, Originated and Settled**).

Request:

```

<TStream>
  <Transaction>
    <TranType>ACH</TranType>
    <TranCode>Query</TranCode>
    <MerchantID>62589001=TEST</MerchantID>
    <InvoiceNo>1620223</InvoiceNo>
    <AcqRefData>860478545</AcqRefData>
  </Transaction>
</TStream>

```

Response:

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Accepted</TextResponse><!-- Pending, Queued, Originated and Settled -->
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>62589001</MerchantID>
    <CardType>ACH</CardType>
    <TranCode>Query</TranCode>
    <InvoiceNo>1620223</InvoiceNo>
    <Amount></Amount>
    <AcqRefData>860478547</AcqRefData>
    <DateProcessed>06132019</DateProcessed>
  </TranResponse>
</RStream>

```

2. Datacap NETePay and dsiEMVUS/dsiPDCX

Datacap dsiEMVUS and dsiPDCX—a Quick API Comparison

The **dsiEMVUS** is a locally installed Windows ActiveX control designed to communicate with Datacap's NETePay server while interfacing with an approved EMV enabled and P2PE Encrypted PIN Pad device. This semi-integrated solution gives developers quick access to the often complicated, ever-evolving chip card regulations while insulating the payment application from handling cardholder data.

- Transaction requests are passed from the POS to the dsiEMVUS which then is used to drive the prompts on the EMV device for cardholder input.
- Transactions are sent as EMVSale, EMVPreAuth, EMVReturn etc., but the manner in which the card data is entered and the actual type of card used is managed by the PIN Pad at the time of the transaction (Insert, Swipe or Tap).
- The dsiEMVUS can be used for Credit and Debit transactions as well as Admin commands such as EMVParamDownload, BatchSummary/BatchClose and Store and Forward; in addition, other operational commands are available such as GetSignature and GetDevicesInfo.
- The dsiEMVUS also supports IncrementalAuths, Partial Reversals and Pay at the Table. A grocery suite of alternative transaction flows was recently added called EasyCheckout which includes pre-dip, on-screen line item details and on-screen cardholder payment options.
- The current list of EMV supported devices for MercuryPay includes 20+ models of Verifone, Ingenico, IDTech with a wide range of form factors from simply designed EMV capable insert devices to the latest in counter-top multi-function PIN pads; unattended Kiosk, Tablet and WiFi hand-helds are also available.
- Datacap has developed comprehensive testing application for EMV. Details on this follow below. The application is available under their download menu under the Utilities/Diagnostics dropdown:
<http://www.datacapipay.com/downloadmenu/>
- **What cannot be processed using the dsiEMVUS?** There is a subset of transaction types that currently cannot be processed using the dsiEMVUS Client, including Gift, StoreCard, EBT (government sponsored Electronic Benefits) and FSA (Flexible Spending/Health Savings Account Cards). For this subset of transactions the dsiPDCX is used.
- For a deeper dive into technical details related to the dsiEMVUS on Datacap's developer portal visit <https://www.dsidevportal.com/api/emvus/>

The **dsiPDCX** is installed and run locally, typically in-tandem with the dsiEMVUS and in-store NETePay as a way of seamlessly managing the merchant's card acceptance and payment processing. On MercuryPay, the dsiPDCX processes Gift (PrePaid), StoreCard, EBT and FSA transactions.

- Where the **dsiEMVUS** specializes in handling EMV Credit and Debit transactions the **dsiPDCX** specializes in handling non-EMV Credit, Debit, Gift, EBT and FSA transactions. When the ActiveX controls are running in-tandem, the dsiEMVUS is preferred for managing all Credit and Debit transactions, leaving Gift, EBT, and FSA to be handled by the dsiPDCX.
- The dsiPDCX, designed to drive multiple peripheral devices, also insulates a POS application from handling cardholder data by directly controlling the card account input device (MSR or PIN Pad).
- A testing utility is not available for the dsiPDCX from Datacap.
- For a deeper technical dive into the method calls used to integrate to the dsiPDCX on Datacap's developer portal visit <https://www.dsidevportal.com/api/pdcx/>

As a quick step by step way to heighten an understanding of how the Datacap components work together, the following install and configure details is provided below.

Datacap Authorized Reseller Application

Before we take a deeper dive into the Datacap solutions, it is important that your business and application are listed correctly on Datacap's Payment Systems Configuration Server (PSCS). This information is required by the Worldpay from FIS IP support team when building your merchant's deployment files.

In order to do this, Datacap provides a short *Reseller Application* form on their website which should be completed well in advance of your first installation. Please complete and submit this online application located at <http://datacapsystems.com/reseller-application>. If you have any questions filling out this form, please contact your Implementations Consultant.

Follow this guideline when completing the application to ensure it's processed as quickly as possible:

1. **POS/ECR Systems You're Currently Authorized To Sell:** Enter your company/product name and version. This should be the same product/version you have certified to Worldpay from FIS IP.
2. **Existing Usage Program Partner Relationships:** Check all that apply, and be sure to check the box next to "Worldpay from FIS IP (Mercury)" in other parts of the Datacap site.)
3. **Additional Information/ Comments:** Make note that you are a Worldpay from FIS IP/Mercury Reseller.

To avoid production delays, we strongly recommend that you complete and submit the *Datacap Authorized Reseller Application* as soon as you start your integration.

NETePay™ 5.07.33 and dsiEMVUS® 1.53/dsiPDCX® 1.73

Note: as Datacap is continually striving to enhance and improve their components, newer versions are always made available on the Datacap web site at <https://www.datacapsystems.com/>.

- It is recommended always to implement the latest NETePay and client controls. Release notes of enhancements and bug fixes are also always available at <http://www.datacapipay.com/downloadmenu/>.

Quick Reference Install and Configure Instructions

There are 6 main steps required to download and install the Datacap components for development testing. You will be downloading and installing the Datacap software first and then accessing their Payment Systems Configuration Server (PSCS) to load the necessary parameters required for testing.

- **Before you begin,** be sure you have a test **DeploymentID** from your Implementation Consultant—you will need this to complete the installation in step 4 below.
- Have your EMV enabled device ready and any required drivers installed. At the end of the installation of NETePay/dsiEMVUS, you will be required to perform a EMV ParamDownload to your device as a final step in this process.

Step 1: Install the Production NETePay 5.07.33

- If you already have the current production release in your test environment, you may proceed to Step 2.
 - Download and install the production NETePay 5.07.33. In a later step, you will add an executable file that enables communication with our test platform.
 - **Go to:** <http://datacapipay.com/epay50/3570/>. This is the NETePay software downloads page specific to Worldpay from FIS IP (Mercury).
 - Scroll down the page to Step 3 – Install NETePay Worldpay from FIS IP (Mercury) US EMV + Debit – MPH 5.07.33 and click download on the right side of the screen.

Note: At this time, **do not** select the **NETePay 5 Director** for development testing as it is unsupported. For development work, please proceed with the classic install of NETePay. We do recommend always using the **Director** when doing production installs. The **NETePay 5 Director** provides a simplified installation process for production accounts installs. For details: <https://www.datacapsystems.com/blog/2018/1/9/automatic-updates>.

- Launch the downloaded .exe and NETePay will install to the default location: C:\Program Files(x86)\Datacap Systems\NETePay. A shortcut is made available on your desktop.

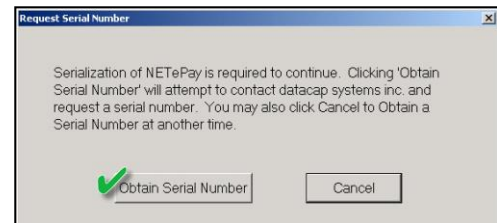
Step 2: Install the Developer/Test NETePay 5.07.33 .exe

The Developer (Dev) NETePay executable is required to redirect your production NETePay to our certification/testing host at x1.mercurycert.net. This is available at the link below.

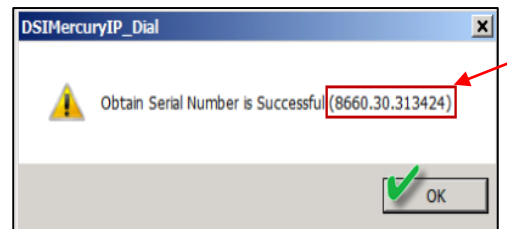
- Go to: <http://files.datacapepay.com/software/dev-netepay/mercury/version5dev/USEMV/>
 - In the Parent Directory, click on the **50733/** and then **DSIMercuryIP_Dial.exe**.
 - Save this file to your desktop. This is the "Dev NETePay."
 - Delete the production NETePay from the default location: **C:\Program Files(x86)\Datacap Systems\NETePay**. If there is a shortcut for this file on your desktop, you may delete it.
 - Copy and paste the "Dev NETePay" (**DSIMercuryIP_Dial.exe**) from your desktop into the **C:\Program Files(x86)\Datacap Systems\NETePay**. Create a shortcut for this file on your desktop if desired.

Step 3: Launch the Dev 5.07.33 NETePay

When you launch the NETePay for the first time, it will prompt to obtain a serial number from Datacap's **PSCS** (Payment Systems Configuration Server) system.



Click **Obtain Serial Number** and on the confirmation screen, click OK. A new dialog box saying **Configuration Not Set – Use Setup | Merchant Parameters**



Click **OK** to move to the "Test Host" NETePay main status window.



Step 4: Load PSCS parameters to your NETePay and the PIN Pad

Open NETePay and select **Setup**. (Clicking either the blue cylinder or setup brings you to the **Setup Merchant Parameters** screen).

On the **Setup Merchant Parameters** screen, note the parameters are grayed-out but indicate the settings that are available from the PSCS Deployment file. Click **Load New Parameters**.

The 'Setup Merchant Parameters' dialog box contains several sections of settings, many of which are grayed out. The 'Load New Parameters' button at the bottom center is highlighted with a red circle.

- Merchant Setup:** Merchant Category (Retail), Store And Forward (Enable, Max Transactions: 20, Max Purchase Limit: 500, Maximum Time Length for Stored Transactions: 48).
- Vehicle Tag Service:** None, iDriveThru.
- Pay at The Table Support:** Disable, PinPad Ip Address, PinPad Ip Address and MAC.
- Transport:** IP Only, IP with DIAL Backup, DIAL Only.
- Listen on Port (s):** Both (9000/9100), Credit/Debit (9000), PrePaid/Gift (9100).
- EMV Support:** None, Credit Only, Credit and Swiped PIN Debit.
- Transaction Support:** All, Signature, None.
- CVM Support:** All, Signature, None.
- Dial Backup Information:** Comm Port (1), Dial Prefix, Mercury Authorization Phone Number (18778471343), Assigned BIN (6) Digits (023400), Canadian BIN (6) Digits (000000), Duplicate Handling on DIAL (Enhanced Dupe Check (InvoiceNo)).
- IP Connection Information:** Connect Timeout to Mercury (3) Seconds.
- NETePay for Mercury:** Use Client/Server Password.
- Max Lanes:** 2, Lane Setup.
- Buttons:** OK, Load New Parameters (circled in red), PSCS.

This step will activate your deployment file on the Datacap PSCS server. On the following screen, click **Yes to Attempt Activation?**

The dialog box displays an error message: "Parameter Load Failed - Error (PSCS): Deployment needs to be activated - not found for this Serial Number". It asks "Attempt Activation?" with "Yes" and "No" buttons. The "Yes" button is circled in red.

Click **I Have My DeploymentID**. (This ID is provided to you by your Implementations Consultant).

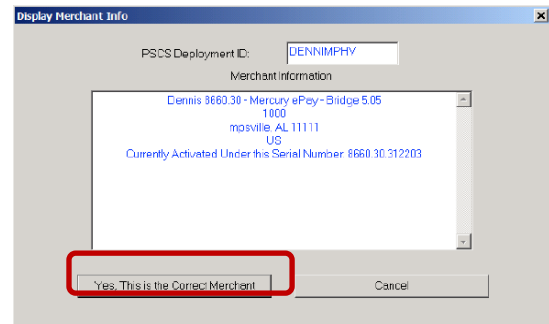
The 'Collect Merchant Information' dialog box has two buttons: "I Have My Deployment ID" and "I Do NOT Have My Deployment ID". The "I Have My Deployment ID" button is circled in red.

In the dialog box that displays, enter the Deployment ID you received, and then click **OK**.

The 'Enter Deployment ID' dialog box has a text field for "Deployment ID" and "OK" and "Cancel" buttons.

A new screen will display with your test merchant data used in the DeploymentID set up. Confirm and click **Yes. This is the Correct Merchant**.

This will start the download process of the deployment parameters.



If NETePay successfully retrieves the parameters associated with the entered DeploymentID from the PSCS server, you will see the following dialog box. Click **OK** to return to main NETePay screen.



Note that "TEST HOST" is displayed in the upper left corner and there is a running timer on the right. Minimize this window. Do not close out of the NETePay; NETePay must *always* be running in the background.

- With the new parameters in place in your local NETePay 5.07.33, you will now add the dsiEMVUS v1.53 client control so you can do the final step which is to load the new parameters to your EMV enabled device.

Step 5: Download the current dsiEMVUS (v1.53) control and (if required) dsiPDCX 1.73

The latest dsiEMVUS client control is located on the Datacap downloads menu page at <http://www.datacapipay.com/downloadmenu/>. It may be accessed in three ways:

- Under the NETePay 5.0: Worldpay from FIS IP (Mercury) US EMV + Debit – MPH 5.07.33
- Directly under **Client Controls**: dsiEMVUS 1.53.
- Additionally, as a convenience for users of the EMV Test Application, the **dsiEMVUS w/Test App 1.53** is available under the Utilities/Diagnostics menu.

Step 6: Perform EMV ParamDownload to your supported Device

The final step of the install is to load the parameters to the peripheral device you are using. To do this an EMVParamDownload is required.

- Use the **GetDeviceInfo** command to confirm the secure device ID specific to the hardware device you are using.
- There are a vast number of devices supported; information on each device available at: <https://www.datacapsystems.com/compatible-devices>.
- It is a Worldpay from FIS certification requirement to include in your software an integrated mechanism to send the EMVParamDownload command.

- To familiarize you with the XML command needed to do this, and the other features and functions built into the NETePay 5.07.33, the Datacap test application may be used. This application is described below.

Datacap's EMV Testing Application

For EMV testing with new or existing features, the Datacap dsiEMVUS Test application is an easy way to learn about transactions and functionality supported by dsiEMVUS. This tool illustrates the XML format behind each request and response and request/response is recorded in log files located at: **C:\Program Files (x86)\US EMV Client**.

Datacap's EMV test application is available on their site at:

- <http://www.datacapipay.com/downloadmenu/> and select **Utilities/Diagnostics**. Click **dsiEMVUS 1.53 w/US test app** and follow the install procedures.

Once installed, the user interface will display:

Note: Datacap's Test Application is a great learning and troubleshooting tool; it is not intended to be used in any way as part of your integrated solution. The screen capture below illustrates the EMV Test Ap interface:

EMV US Test Application Configuration

Here is a summary of common MercuryPay specific configurations when using Datacap's EMV Test Application. As this application is continually being enhanced, be sure to download the latest version from <http://www.datacapipay.com/downloadmenu/> and use the below as a general guide.

Feature	Detail/Description
1 Processor and Supported Device	Select by clicking on the Mercury(E2E) device you are using
2 Merchant ID (MID)	Here, test MID 62589008=TEST, enabled for both encryption and tokenization. Additional test "Quick Access" MIDs are available in our MercuryCert.Net Testing Guide.
Terminal ID	Not required
Operator ID	This is the end-user or clerk identifier.

3	IP/HostName	127.0.0.1 (or the IP address where NETePay resides)
	IpPort	9000 is the default
4	Com Port	Enter the device COM port. If using a USB device, a device driver is used to create a virtual COM port.
5	Amount	Use amounts under \$10.99 or greater than 50.00. Change your amount and InvoiceNo with each transaction to avoid duplicates (AP DUPE).
	TIP Amount	Allows for an added gratuity amount; alternatively you can use a transaction modifier to "prompt for TIP" on the device.
	Cash Back	Allows for added cash back on debit; alternately you can use a prompt for CASH BACK.
6	Reference No	Required: is returned in the response and increments from the host
7	Invoice No	Required: POS increment this number for every transaction to avoid unintended duplicates.
	Transaction Modifiers	Required: check AllowPartialAuth and Use Tokens. Check others as applicable.
	Collect Name	The CollectNames modifier sets a tag CollectData: CardHolderName used to optionally extract the cardholder name from the card if available. This will also be made available in the draft Print Data.
	More Transaction Modifiers	Use these to disable credit or debit transactions, and to suppress the "Amount OK?" prompt
8	Clear Response Fields on New Transaction	Check this box to clear all the fields before the next transaction Uncheck this box to keep the fields (i.e. for ByRec transactions)
	GetDevicesInfo	Use to get the supported SecureDeviceID
	Get Signature	Independent Command for capturing an electronic signature
9	PIN Pad Reset	Confirms communication with device and resets pad message
10	EMV ParamDownload	Used to load parameters from the local NETePay to the device.

EMV Test Device – Setup

Worldpay from FIS supports multiple EMV certified devices for dsiEMVUS, and the list continues to grow. Current details on supported devices are also available at <https://www.datacapsystems.com/compatible-devices>. Install any required USV device drivers. Connect the device to your computer.

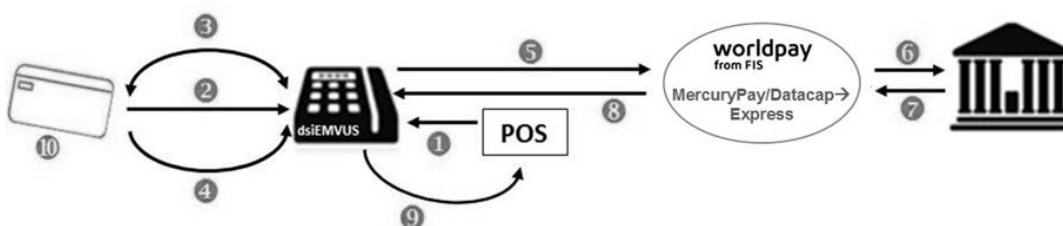
- **Devices with USB cables may require a driver.** Install the driver prior to connecting your device and reboot your computer if requested.
- Drivers can be downloaded from the Datacap download menu. Go to <http://www.datacapexpress.com/downloadmenu/> and click **Device Drivers**.

EMV Devices, Flow and Functionality with Datacap's dsiEMVUS

The Datacap **dsiEMVUS** is a locally installed Windows ActiveX control, which serves as the application interface between the POS, the EMV device, and Datacap's US EMV-certified **in-store NETePay**. The *dsiEMVUS*:

- Directly controls an EMV device through a PC port (Serial/RS232 or USB using a virtual COM port) to manage all PIN pad interaction.
- Uses simple XML formatted communication for transaction processing and returns only non-sensitive XML data available for POS payment logs.

The POS sends transaction XML commands to the *dsiEMVUS* which then takes control of the remaining processing flow:



dsiEMVUS Data Flow via Worldpay from FIS

1. The POS initiates an EMV transaction to the *dsiEMVUS* which then takes control of the EMV enabled device. Based on the type of request, screen forms loaded on the device take the cardholder through various prompts.
2. The cardholder inserts the chip card into device and follows the prompts.
3. The chip card and card reader exchange/negotiate information to:
 - Determine transaction type (credit, debit)
 - Read an displays application data
 - Determines preferred Card Verification Method (PIN, Signature, none)
4. The chip on the card generates an ARQC and passes it along with a cryptogram to *dsiEMVUS* which builds a complete Authorization request.
Note a single ARQC is used in "Quick Chip" devices.
5. *dsiEMVUS*, accelerated by Datacap's NETePay, sends the Authorization request to Worldpay from FIS /MercuryPay for processing.
6. Worldpay from FIS/MercuryPay routes the Authorization request to the Issuer.
7. The Issuer sends the Authorization response back to Worldpay from FIS.
8. Worldpay from FIS sends the Authorization response message back to the local NETePay and *dsiEMVUS*.
9. *dsiEMVUS* sends response and complete receipt information to the POS.
10. The cardholder removes the card from the EMV device. (With Quick Chip, the cardholder may remove the card in step 5.)

Standard EMV Differences

EMV is more than just another POS entry method—it is a culture shift for the consumer and for merchants, and may require a new way of thinking about their business. For the ISV, it means new challenges with device communications, revisiting existing POS code, new transaction requirements and a whirlwind of information. We understand and are right there with you. We are committed to simplifying these challenges.

EMV and the Consumer Experience

1. In addition to a conventional magnetic stripe reader, EMV-enabled devices have a separate slot only used for inserting and reading the chip card. These devices also support contactless and keyed entry. Until 2019 only Magstripe grade contactless was used in our devices; **card mandates now require that all contactless on an EMV enabled device must also be EMV grade.**
2. Chip cards must stay in the EMV-enabled reader until the Pad confirms the chip has been read.
3. **A swipe reader cannot read a chip and the chip insert reader cannot read a magnetic stripe.**
 - If a magstripe card is inserted into an EMV-enabled device, the device will prompt the cardholder to swipe the card.
 - If an EMV chip card is swiped into an EMV-enabled device, a code on the magnetic stripe will trigger the device to prompt: **USE CHIP READER.**
 - If a chip card is inserted into an EMV-enabled device but cannot be read correctly, the device will prompt the cardholder to **USE MAG STRIPE** followed by **REMOVE CARD**. This is called a "technical fallback."
4. **PIN entry is not just for Debit transactions.** EMV Credit may also be verified using PIN entry as well. Contactless transactions may not require a PIN until a certain dollar threshold has been reached.
5. **EMV Contactless** is now the required, mandated standard for all EMV enabled devices. Devices deployed prior to 2019 may only be enabled for the legacy magstripe grade of contactless and will need to be upgraded or replaced.
6. On-device cardholder Application selection, as in Visa Credit or US Debit, is common on select EMV Debit enabled devices.
7. Card brand receipt requirements have become less stringent. Whether a transaction is APPROVED, DECLINED, ERROR or CANCELLED, card brand established receipt best practices. A good rule of thumb is if the Datacap **<PrintData>** is returned in the transaction XML response, receipts may be generated—typically one for the cardholder and one for the merchant to retain—and always upon request.

The card brand regulations vary on specific parameters of the receipt requirements. Please see the Receipt Requirements section of this document for recommended best practices for printing receipts.

EMV Data Flow and Terminology

EMV chip cards contain an embedded integrated circuit chip (**ICC**), which is a microprocessor inside the card. When a chip card is inserted into a chip card entry device, the card and device share physical contact points which enable a two-way exchange of EMV data. In addition to inserting a chip card into a reader, there are **contactless** chip cards which have Near Field Communication (**NFC**) capabilities. Chip cards that combine both contact and contactless are referred to as **dual-interface** cards.

Note There are two “grades” of contactless: EMV grade and magnetic stripe (**MSD**) grade. With the 2019 card brand mandates only EMV grade is now permissible on EMV enabled devices: “Visa, Mastercard, Discover, and American Express have published revised contactless standards requiring EMV payment terminals and POS Systems using EMV PIN pads to **retire the use of legacy (MSD) contactless** (for Mastercard and Visa) and to **enable support for the EMV mode of contactless acceptance** for all card brands.”

The chip on the card is configured with one or more **Application Identifiers (AIDs)**:

- Each AID contains an ISO-registered ID,
- Identifies the card brand (e.g., MasterCard, Visa, or American Express),
- Indicates the card product type (credit, debit or ATM)
- Indicates the preferred card verification methods (**CVM**) which can be PIN, SIGNATURE or NONE.

The AID also contains cardholder agreement factors such as amount limits, daily usage counters and risk management features.

Both the EMV device and EMV card maintain a list of AIDs they support:

- The Issuer loads the chip with its preferred applications and AIDs
- The device contains a list of AIDs it supports, which may be specific to the business environment.

The card may contain multiple AIDs, referred to as **candidate lists**. The device also contains one or more AIDs. The device and the card compare their lists and agree on the “best” AID, based on a hierarchical structure. This selection happens automatically without cardholder interaction, and is called **Application Selection**. This selection process determines if the receipt output requires a **Signature** line, allows the transaction to be **PIN Verified** or to allow **No CVM** (no card verification method required).

Note with the Datacap *dsiEMVUS* and MercuryPay, all of this is handled seamlessly and certified on your behalf, right down to the required receipt information.

PIN entry CVMs for EMV credit card verification is managed in two different ways, yet from the end-user and POS perspective, the experience and interaction will appear the same:

- **Online PIN** is when the encrypted *PINBlock* and key are sent along with the transaction request to the processor for decryption and authorization. In this case, PIN decryption is a function of the Issuer’s host system.
- **Offline PIN** means the chip communicates directly with the device to verify the PIN entry at the device level. The PIN is securely encrypted on the chip itself. When the card is inserted, the chip communicates with the device that an offline PIN is available if the device supports it. If it does, the device will prompt for PIN entry. The PIN entered on the device is compared to the PIN stored on the chip card, and only the result of the comparison is transmitted to the Issuer host system.

The EMV request and response cryptogram: The chip uses a combination of EMV tag data elements, a uniquely generated application cryptographic master key, and a message authentication code to create a unique cryptogram. This is called the Authorization Request Cryptogram (**ARQC**) and is sent to the Issuer along with the key, key index and details on how the ARQC was generated. Only the Issuer can break down and read the ARQC.

EMV Impacts on Transaction Processing

The following section outlines details related to the impact of EMV on common processing flows including gratuities, using ZeroAuth and IncrementalAuth, handling Cashback and chip Fallback scenarios.

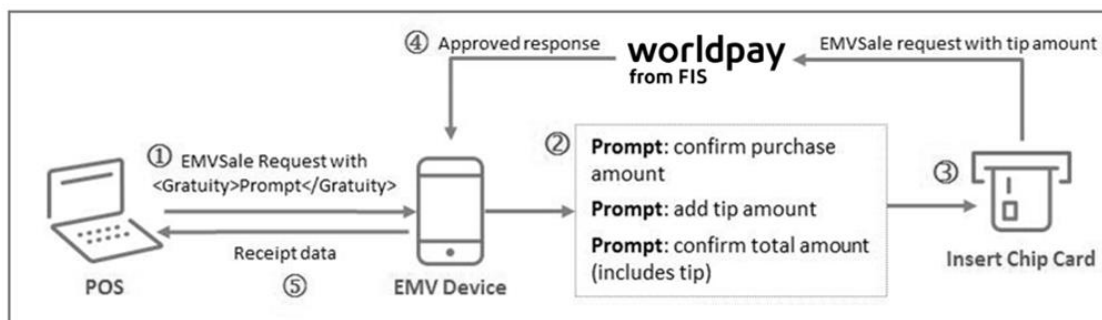
Gratuities, ZeroAuth, IncrementalAuth, Cashback and Fallback

Adding gratuities (tips)

For restaurants and other environments that accept gratuities, adding an EMV credit tip can be done in two ways—in the flow of the transaction using a gratuity prompt, or by adding a gratuity in a follow up Adjust/PreAuthCapture.

Prompt for Tip in the flow of the transaction

Send an *EMVSale* and include `<Gratuity>Prompt</Gratuity>`, which displays a prompt on the device asking the consumer to add a tip.



1. A request is sent in the *EMVSale* which includes the purchase amount and the gratuity prompt
2. The device prompts the consumer to:
 - confirm the purchase amount
 - add a tip amount
 - confirm the new total amount
3. The consumer inserts the card and the transaction is sent for approval
4. The response confirms the approval and includes three amount tags:


```

      <Purchase>5.00</Purchase>
      <Gratuity>1.25</Gratuity>
      <Authorize>6.25</Authorize>
      
```
5. Receipt data is sent from the EMV device to the POS.

Modifying the Tip using a follow up Adjust or PreAuthCapture transaction

This option requires the use of MercuryPay's tokenization—*AdjustByRecordNo* or *PreAuthCaptureByRecordNo*. The steps are similar to the non-EMV two-step *PreAuth/PreAuthCapture* method commonly used in restaurants. With MToken implemented and a *RecordNo* requested, send an *EMVSale* (or *EMVPreAuth*), then:

- Send an initial *EMVSale/EMVPreAuth*.
- When authorized, store the returned token for the adjustment/capture.
- Build out a sales receipt including a tip-line for the gratuity and a line for customer signature.
- Present the receipt to the customer for signature and gratuity.
- Closing out the check, using *AdjustByRecordNo/PreAuthCaptureByRecordNo* to send the gratuity amount: `<Gratuity>2.00</Gratuity>`

Note The Datacap PSCS EMV set-up should be set to **Credit** only and **Signature**. Credit CVMs that require PIN entry and U.S. PIN Debit do not allow gratuity to be added in a subsequent transaction using a token.

Running a tab—ZeroAuth

Running a tab refers to the POS feature of identifying a customer and continuously adding to the purchase amount as that customer orders additional items over a period of time until the customer finishes and the tab is closed. There are multiple ways for POS applications to handle tabs, and how this feature is implemented is determined by a balance between business requirements, need for fraud protection and card brand regulations. The enhanced fraud protection of the chip read and preferred processing rates associated with card present transactions make the adoption of EMV in tab environments appealing. However, there is a minor difference with EMV cards—the customer may need to insert their card more than once:

- **To open the tab**, the customer or clerk inserts the card and the POS sends a *ZeroAuth*. *ZeroAuth* validates that the card has not been reported lost or stolen—without charging the card. For additional fraud protection *ZeroAuth* may be sent with the customer's AVS data.

Note This is best used with tokenization. A token is returned and stored for subsequent use. With tokenization, should a consumer leave before closing the tab, the token may be used to finalize the transaction. Although this is not optimal, it can help to reduce loss due to so called "Dine and Dash" behavior.

- **To close the tab**, customer or clerk inserts the card a second time and sends an *EMVSale* for the final amount.

Tip modification can be handled either in the above flow by including a device prompt, or by using the token returned from the *EMVSale* and sending a *AdjustByRecordNo*.

As an alternative to the *ZeroAuth* initiation of a tab, the initial transaction can be used for validating the card. This method requires that a reversal of the original *EMVSale* be sent prior to sending and adjusting the final total amounts.

Running a tab – Incremental Auth

Another option for running a tab is by implementing *IncrementalAuth*. This was added to the MercuryPay platform in 2019.

Incremental authorizations have long been a useful transaction supported by the Car Rental and Lodging industries. In 2017, Visa and MasterCard expanded the eligible markets to include Restaurants, Bars, and Nightclubs.

Worldpay Integrated Payments, in partnership with Datacap Systems, is pleased to be able to support incremental authorizations for our many partners and resellers working in the hospitality space. Combined with the use of MercuryPay's token technology *MToken*, incremental authorizations in bars and restaurants can be a beneficial option for supporting seamless bar tab handling. If there is ever an item that needs to be removed from the running tab, the Partial Reversal/decrement makes for an easy correction before closing the tab.

The following details will introduce you to this bar tab handling transaction flow and XML tags required to support incremental authorizations and partial reversals using Datacap's NETePay™ and dsiEMVUS®.

IncrementalAuth Bar Tab Transaction Flow

PreAuth → IncrementalAuth → PartialReversal (if needed) → PreAuthCapture

The incremental authorization sequence begins with a standard *PreAuth* transaction, familiar to hospitality POS developers and now available using the customer's chip card. In this case we will use the EMV equivalent, *EMVPreAuth*.

The *EMVPreAuth* establishes the "open tab" with a single card swipe or insert. It is important for your POS to maintain and use the transaction response data from this initial authorization. The initial *AuthCode* and transaction reference data returned within the *AcqRefData* field, along with the originating auth amount and *LastAuthorized* amount, are required throughout the incremental transaction sequence.

- The *AcqRefData* (Acquirer Reference Data) includes a transaction identifier which is used by Visa and MasterCard to link the incremental auths to the final completion. This identifier is also defined as the *TransactionID*.

After the Initial PreAuth, IncrementalAuth Requires Three Amount Tags

With the original EMVPreAuth completed, the *IncrementalAuth* TranCode (or, when using tokenization, *IncrementalAuthByRecordNo*) is used to increment the original authorization.

IncrementalAuth uses three amount tags: *OriginalAuthorized*, *LastAuthorized* and *Increment* amount.

```
<Amount>
<OriginalAuthorized>2.00</OriginalAuthorized>  <!-- Initial tab amount -->
<LastAuthorized>4.00</LastAuthorized>  <!-- Current running total -->
<Increment>4.00</Increment>  <!-- New amount to add to the tab -->
</Amount>
```

The *OriginalAuthorized* always equals the amount of the initial PreAuth. The *LastAuthorized* (which corresponds to the *Authorized* amount on the previous transaction response) is the "running total" amount of the customer's tab. The *Increment* is how much is being added to the *LastAuthorized* amount.

When the customer is ready to close the tab, the cashier presents a sales draft to the customer showing the total Authorized amount, tip line and signature agreement. With gratuity added, the final transaction is processed as a standard PreAuthCapture referencing the original AuthCode and AcqRefData from the initial PreAuth. The capture "closes" the tab. If the customer changes their mind and wishes to add another round, a new tab must be opened and a new authorization established.

- Once a *PreAuth* and *IncrementalAuth* sequence has been captured, that is, once the tab has been closed, it cannot be reopened. In this case, always start a new tab sequence with a new *EMVPreAuth*.

Use PartialReversal to Decrement the Amount (if needed)

In the event that the *LastAuthorized* amount needs to be adjusted, a *PartialReversal* (or when using tokenization *PartialReversalByRecordNo*) is used. For *PartialReversals*, the *Decrement* amount tag is used in place of the *Increment* amount to subtract a partial amount from the *LastAuthorized*.

```
<Amount>
<OriginalAuthorized>2.00</OriginalAuthorized>  <!-- Initial tab amount -->
<LastAuthorized>4.00</LastAuthorized>  <!-- Current running total -->
<Decrement>2.00</Decrement>  <!-- Amount to subtract from the tab -->
</Amount>
```

Additional IncrementalAuth Notes

- The transaction sequence always begins with a *EMVPreAuth* followed by the incremented "rounds" to be added to the running tab and completed with a *PreAuthCapture*.
- The transaction data from the originating *EMVPreAuth* response is necessary to store and use in all subsequent *Increment*, *Decrement* and *PreAuthCapture* requests. This includes:
 1. *TranCode*
 2. The token *RecordNo* and *Frequency*
 3. Original authorized amount
 4. *AcqRefData*
- Incremental amounts are always added to previously authorized amounts — and the sum of all incremental authorizations represent the total amount authorized for a given transaction.
- Once the original transaction tab has been captured, it cannot be "reopened". A new transaction sequence (PreAuth→IncrementalAuth→Capture) is required.
- Incremental authorizations are supported by Visa and Mastercard. American Express and Discover card brands will return and **Error: Only Supported for Visa, MC**


```

    <ProcessData>|14|510100501000</ProcessData>
<RecordNo>i7sPoRpJ1JzmaHpvVg6FdDExEysjvgIqbWDacw2BG6ciEgUQB SIQEFJS</RecordNo>
    <EntryMethod>CHIP</EntryMethod>
    <Date>01/19/2019</Date>
    <Time>16:11:09</Time>
    <ApplicationLabel>VISA CREDIT</ApplicationLabel>
    <AID>A000000003101001</AID>
    <TVR>0000008000</TVR>
    <IAD>06010A03A49002</IAD>
    <TSI>E800</TSI>
    <ARC>Z3</ARC>
    <CVM>PIN VERIFIED</CVM>
</TranResponse>
<PrintData>
    <Line1>.MERCHANT ID: *****5004</Line1>
    <Line2>.</Line2>
    <Line3>.          PREAUTH          </Line3>
    <Line4>.</Line4>
    <Line5>.VISA          *****0076</Line5>
    <Line6>.ENTRY METHOD: CHIP</Line6>
    <Line7>.DATE: 01/19/2019  TIME: 16:11:09</Line7>
    <Line8>.</Line8>
    <Line9>.INVOICE: 400</Line9>
    <Line10>.REFERENCE: 10001</Line10>
    <Line11>.AUTH CODE: 000294</Line11>
    <Line12>.</Line12>
    <Line13>.AMOUNT          USD$ 2.10</Line13>
    <Line14>.          =====</Line14>
    <Line15>.TOTAL          USD$ 2.10</Line15>
    <Line16>.</Line16>
    <Line17>.          APPROVED - THANK YOU          </Line17>
    <Line18>.</Line18>
    <Line19>.BY ENTERING A VERIFIED PIN, CARDHOLDER</Line19>
    <Line20>.AGREES TO PAY ISSUER SUCH TOTAL IN</Line20>
    <Line21>.ACCORDANCE WITH ISSUER'S AGREEMENT WITH</Line21>
    <Line22>.CARDHOLDER</Line22>
    <Line23>.</Line23>
    <Line24>.</Line24>
    <Line25>.APPLICATION LABEL: VISA CREDIT</Line25>
    <Line26>.AID: A000000003101001</Line26>
    <Line27>.TVR: 0000008000</Line27>
    <Line28>.IAD: 06010A03A49002</Line28>
    <Line29>.TSI: E800</Line29>
    <Line30>.ARC: Z3</Line30>
    <Line31>.CVM: PIN VERIFIED</Line31>
    <Line32>.TRAN ID: 8238953</Line32><!-- Note new PrintData field -->
</PrintData>
</RStream>

```

Incremental Auth #1: Original Auth \$2.10 Increment \$3.00, Total Amount \$5.10

Request:

```

<TStream>
    <Transaction>
        <HostOrIP>127.0.0.1</HostOrIP>
        <IpPort>9000</IpPort>
        <MerchantID>592885123</MerchantID>
        <OperatorID>Bar15</OperatorID>
        <LaneID>6</LaneID>
        <UserTrace>worldpaytest</UserTrace>
        <TranType>Credit</TranType>
        <TranCode>IncrementalAuthByRecordNo</TranCode><!-- Note new TranCode -->
    </Transaction>
</TStream>

```

```

    <SecureDevice>EMV_P400_MERCURY</SecureDevice>
    <ComPort>9</ComPort>
    <InvoiceNo>400</InvoiceNo>
    <RefNo>10001</RefNo>
    <AuthCode>000294</AuthCode><!-- always save and use original AuthCode -->
    <Amount><!-- Note new amount fields -->
        <OriginalAuthorized>2.10</OriginalAuthorized>
        <LastAuthorized>2.10</LastAuthorized>
        <Increment>3.00</Increment>
    </Amount>
    <SequenceNo>0010010010</SequenceNo>
    <PartialAuth>Allow</PartialAuth>
    <RecordNo>i7sPoRpJ1JzmaHpvVg6FdExEysjvgIqbWDacw2BG6ciEgUQBSIQEFJS</RecordNo>
    <Frequency>OneTime</Frequency>
    <AcqRefData>b8238953#20#20#20#20#20#20#20e00</AcqRefData>
        <!-- always save and use original AcqRefData -->
    <ProcessData>|14|510100501000</ProcessData>
</Transaction>
</TStream>

```

IncrementalAuth #1 Response: Running total authorized \$5.10

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>worldpaytest</UserTrace>
  </CmdResponse>
  <TranResponse>
    <MerchantID>592885123</MerchantID>
    <AcctNo>476173XXXXXX0076</AcctNo>
    <ExpDate>XXXX</ExpDate>
    <CardType>VISA</CardType>
    <TranCode>IncrementalAuth</TranCode><!-- note new response TranCode -->
    <AuthCode>000295</AuthCode><!-- Note new AuthCode returned. Do not use. -->
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>10001</RefNo>
    <InvoiceNo>400</InvoiceNo>
    <Amount>
      <Authorize>5.10</Authorize><!-- note running total returned -->
    </Amount>
    <AcqRefData>Kb8238953#20#20#20#20#20#20#20e00</AcqRefData>
        <!-- this is the original AcqRefData -->
    <RecordNo>n/kd6JTRhy5Ro5OI+QCyd8hDM1nSCiArKkpZCduvlw8iEgUQBSIQEFJW</RecordNo>
    <ProcessData>|00|510100701000</ProcessData>
    <ProcTransactionID>8238957</ProcTransactionID><!-- Note new field unique to this tran -->
  </TranResponse>
</RStream>

```

Incremental Auth #2: Original Auth \$2.10, Last Auth \$5.10 Increment \$3.00,

Total Amount \$8.10

Request:

```

<TStream>
  <Transaction>
    <HostOrIP>127.0.0.1</HostOrIP>
    <IpPort>9000</IpPort>
    <MerchantID>592885123</MerchantID>
    <OperatorID>Bar15</OperatorID>

```

```

<LaneID>6</LaneID>
<UserTrace>worldpaytest</UserTrace>
<TranType>Credit</TranType>
<TranCode>IncrementalAuthByRecordNo</TranCode><!-- Note new TranCode -->
<SecureDevice>EMV_P400_MERCURY</SecureDevice>
<ComPort>9</ComPort>
<InvoiceNo>400</InvoiceNo>
<RefNo>10001</RefNo>
<Auth Code>000294</Auth Code><!-- always save and use the original Auth Code -->
<Amount>
  <OriginalAuthorized>2.10</OriginalAuthorized>
  <LastAuthorized>5.10</LastAuthorized>
  <Increment>3.00</Increment>
</Amount><!-- Note new Amount fields -->
<SequenceNo>0010010010</SequenceNo>
<PartialAuth>Allow</PartialAuth>
<RecordNo>n/kd6JTRhy5Ro5OI+QCyd8hDM1nSCiArKkpZCduvlw8iEgUQBSIQEFJW</RecordNo>
<Frequency>OneTime</Frequency>
<AcqRefData>b8238953#20#20#20#20#20#20#20#20e00</AcqRefData>
  <!-- always save and use original AcqRefData -->
<ProcessData>|00|510100701000</ProcessData>
</Transaction>
</TStream>

```

IncrementalAuth #2 Response: Running total Authorized \$8.10

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>worldpaytest</UserTrace>
  </CmdResponse>
  <TranResponse>
    <MerchantID>592885123</MerchantID>
    <AcctNo>476173XXXXXX0076</AcctNo>
    <ExpDate>XXXX</ExpDate>
    <CardType>VISA</CardType>
    <TranCode>IncrementalAuth</TranCode><!-- new TranCode -->
    <Auth Code>000296</Auth Code>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>10001</RefNo>
    <InvoiceNo>400</InvoiceNo>
    <Amount>
      <Authorize>8.10</Authorize><!-- running total -->
    </Amount>
    <AcqRefData>Kb8238953#20#20#20#20#20#20#20#20e00</AcqRefData>
      <!-- original AcqRefData -->
    <RecordNo>QWI+5WN/V55KEwzZn37vWdCERfnTLn8ZAfR8Z2IUfN8iEgUQICIQDNgm</RecordNo>
    <ProcessData>|00|510100701000</ProcessData>
    <ProcTransactionID>8238961</ProcTransactionID>
      <!-- Note new field, value unique to this tran -->
    </TranResponse>
  </RStream>

```

Partial Reversal Request Original Auth \$2.10, Last Auth \$8.10 Decrement \$1.00, Total Amount \$7.10

```

<TStream>
  <Transaction>
    <HostOrIP>127.0.0.1</HostOrIP>

```



```

<IpPort>9000</IpPort>
<MerchantID>592885123</MerchantID>
<OperatorID>Bar15</OperatorID>
<LaneID>6</LaneID>
<UserTrace>worldpaytest</UserTrace>
<TranType>Credit</TranType>
<TranCode>PartialReversalByRecordNo</TranCode>
    <!-- New TranCode -->
<SecureDevice>EMV_P400_MERCURY</SecureDevice>
<ComPort>9</ComPort>
<InvoiceNo>400</InvoiceNo>
<RefNo>10001</RefNo>
<AuthCode>000294</AuthCode>
    <!-- always save and use original AuthCode -->
<Amount>
    <OriginalAuthorized>2.10</OriginalAuthorized>
    <LastAuthorized>8.10</LastAuthorized>
    <Decrement>1.00</Decrement>
</Amount>
    <!-- note new amount values -->
<SequenceNo>0010010010</SequenceNo>
<PartialAuth>Allow</PartialAuth>
<RecordNo>QWl+5WN/V55KEwzZn37vWdCERfnTLn8ZAfR8Z2lUfN8iEg UQlCIQDNgm</RecordNo>
<Frequency>OneTime</Frequency>
<AcqRefData>b8238953#20#20#20#20#20#20#20e00</AcqRefData>
    <!-- always save and use the original AcqRefData -->
<ProcessData>|00|510100701000</ProcessData>
</Transaction>
</TStream>

```

Partial Reversal Response: Total Amount \$7.10

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>worldpaytest</UserTrace>
  </CmdResponse>
  <TranResponse>
    <MerchantID>592885123</MerchantID>
    <AcctNo>476173XXXXXX0076</AcctNo>
    <ExpDate>XXXX</ExpDate>
    <CardType>VISA</CardType>
    <TranCode>PartialReversal</TranCode><!-- New response TranCode -->
    <AuthCode>000294</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>10001</RefNo>
    <InvoiceNo>400</InvoiceNo>
    <Amount>
      <Authorize>7.10</Authorize><!-- Running total -->
    </Amount>
    <AcqRefData>Kb8238962#20#20#20#20#20#20#20e00</AcqRefData><!-- New AcqRefData - do not
use! Use original -->
    <RecordNo>|lFr/tPhNcxjzN289df0jeN+reU78kFbbKKXkiWe5YEiEgUQBSIQEFJd</RecordNo>
    <ProcessData>|00|510100701000</ProcessData>
    <ProcTransactionID>8238962    </ProcTransactionID><!-- Note new field unique to this tran -->
  </TranResponse>
</RStream>

```

Final tab scenario: Guest asks to close the tab; the above total, \$7.10 was presented in a sales draft/check to the guest for tip and signature.

Final Capture Transaction/Close Tab: PreAuthCapture 7.10 + added Gratuity 2.00, Total Amount \$9.10

Request

```
<TStream>
  <Transaction>
    <HostOrIP>127.0.0.1</HostOrIP>
    <IpPort>9000</IpPort>
    <MerchantID>592885123</MerchantID>
    <OperatorID>Bar15</OperatorID>
    <LaneID>6</LaneID>
    <UserTrace>worldpaytest</UserTrace>
    <TranType>Credit</TranType>
    <TranCode>PreAuthCaptureByRecordNo</TranCode><!-- Standard PreAuthCapture TranCode -->
    <SecureDevice>EMV_P400_MERCURY</SecureDevice>
    <ComPort>9</ComPort>
    <InvoiceNo>400</InvoiceNo>
    <RefNo>10001</RefNo>
    <AuthCode>000294</AuthCode><!-- Use the original AuthCode -->
    <Amount>
      <Purchase>7.10</Purchase>
      <Gratuity>2.00</Gratuity><!-- Use standard PreAuthCapture amount fields -->
    </Amount>
    <SequenceNo>0010010010</SequenceNo>
    <PartialAuth>Allow</PartialAuth>
    <RecordNo>IIfr/tPhNcxjzN289df0jeN+reU78kFbbKKXkiWe5YEiEgUQBSIQEFJd</RecordNo>
    <Frequency>OneTime</Frequency>
    <AcqRefData>b8238953#20#20#20#20#20#20#20#20e00</AcqRefData><!-- Use original AcqRefData -->
    <ProcessData>|00|510100701000</ProcessData>
  </Transaction>
</TStream>
```

Final PreAuthCapture Response: purchase \$7.10, Gratuity \$2.00, Authorized \$9.10

```
RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>worldpaytest</UserTrace>
  </CmdResponse>
  <TranResponse>
    <MerchantID>592885123</MerchantID>
    <AcctNo>476173XXXXXX0076</AcctNo>
    <ExpDate>XXXX</ExpDate>
    <CardType>VISA</CardType>
    <TranCode>PreAuthCapture</TranCode>
    <AuthCode>000294</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>0001</RefNo>
    <InvoiceNo>400</InvoiceNo>
    <Amount>
      <Purchase>7.10</Purchase>
      <Authorize>9.10</Authorize>
      <Gratuity>2.00</Gratuity>
    </Amount>
    <AcqRefData>b8238975#20#20#20#20#20#20#20#20e00</AcqRefData>
  <RecordNo>cJ0ri6KPKMoJ0MFJYANKgpFXIsCdSXilBKUY6CPN5ciEgUQICIQDNhD</RecordNo>
  <ProcessData>|15|510100201000</ProcessData>
```

```

        <ProcTransactionID>8238975    </ProcTransactionID>
    </TranResponse>
</RStream>

```

Cashback and EMV

For most merchants, cashback is a feature of PIN Debit only. If your POS includes a prompt for cashback, the customer will perform the following steps on the EMV device:

1. Confirm the original purchase amount.
2. Select *Yes* or *No* on Cashback.
3. If *Yes*, enter an actual amount.
4. Reconfirm the total amount.
5. Then *Insert*, *Swipe* or *Tap* the card.

Note in the above process flow, the type of card is *unknown* when cashback is selected in Step 2. This process flow is an EMV requirement in anticipation of EMV PIN Debit: the cashback amount and confirmation must be completed *before* the type of card is negotiated when inserted into the device.

However this poses a problem today—if the POS prompts for cashback and the customer selects an amount, inserts a *credit* chip card or swipes a non-EMV *credit* card and selects credit, the transaction will fail with the error message, **Cashback Not Supported**.

To avoid this problem, merchants may need to ask the cardholder if the card is debit and if they want cash back *before* proceeding to run the transaction.

Entry method "fallback" and prompting for manual entry

Fallback is a PIN Pad device mechanism and Chip interaction whereby a bad chip read is detected. When this occurs, the device may "fallback" to magstripe entry.

- If the device unable to read the card at all (chip or magstripe), the POS may prompt for manual entry.
- To do this, after failed attempts or if the device returns "**Unsuccessful Card Read from PinPad**," the POS can resend the request and initiate a manual entry using **<AcctNo>Prompt</AcctNo>**.

Note Failed card reads could indicate fraudulent activity. Prompting for manual entry will incur a greater processing rate and indicates potential fraud. Always accompany a prompt for manual with zip code and street address AVS data.

EMV Commands, Functionality, TranCodes and Secure Devices,

Supported SecureDevices for MercuryPay EMV

- Use **GetDevicesInfo()** to return a list of Datacap Secure Devices supported for a given version of dsiEMVUS and by processor, along with COM interface (RS232, USB, Bluetooth, and WiFi) and the Secure Device Identifier (SecureDevice).
- *SecureDeviceID* tag is required with each request to the dsiEMVUS. The value follows a standard convention of "EMV_Device Name_MERCURY."
- The Datacap developer portal also lists updated supported secure devices die dsiEMVUS and dsiPDCX. <https://www.dsidevportal.com/secure-devices/>

EMV Commands, Functionality and TranCodes

- **Server Version:** Used to confirm the *NETePay* Version.

- **GetDevicesInfo:** Used to determine which *SecureDevices* are supported in the version of *dsiEMVUS* along with the device capabilities and characteristics.
- **GetSignature:** Used to capture an electronic signature. Signature and corresponding receipt must be stored locally per card brand regulations for future receipt generation.
- **EMVParamDownload:** loads NETePay deployment parameters and security information into the device.

Note PSCS Deployment file (PSCS is Datacap's configuration server) is first downloaded to local NETePay; local NETePay loads these files into the supported EMV *SecureDevice* using *EMVParamDownload*. This command should be performed when a new device is first installed, when a device is replaced, or as requested by the service provider.

- **EMVPadReset** is used *programmatically performed before every transaction* to ensure that a card was not left in the chip reader. If there *is* a card in the reader, the PIN pad will display **Remove Card** and waits until the card is removed.

Required/Optional EMV Functions and Transaction Modifiers

Function	Required/ Optional	Notes
Prompt for Manual	O	Prompt for manual; use as fallback for swipe
Prompt for Tip	O	Use to prompt the cardholder for a tip (gratuity) which is added to the total sale amount and sent as a single transaction.
Prompt for CASH BACK	O	Use with Debit transactions to prompt for Cash Back
Allow Partial Auth	R	This a required card brand mandate for card-present purchases to be partially approved.
Admin Functions <ul style="list-style-type: none"> • BatchSummary • BatchClose 	O	These functions are for POS systems that include a local, merchant-initiated settlement function. Use <i>BatchSummary</i> to retrieve current batch counts and amounts for comparisons and use <i>BatchClose</i> to close the current batch.
Additional Modifiers and Functions	As Required	<ul style="list-style-type: none"> • Print Blank Tip Line • Collect Data—Cardholder Name • Encrypted Card Response • Suppress Amount • Demand Credit/Debit • GetDeviceInfo • ServerVersion • GetSignature

EMV TranCodes

TranCode	Required/ Optional	Notes
EMVSale	R	Card present; all Cardholder Verification Methods (CVM) enabled; <i>EMVSale</i> or <i>EMVPreAuth</i> are used to send the request to the device regardless of the entry method; if inserted, the negotiation between the chip, device and cardholder will determine if the transaction is ran as credit or debit.
EMVPreAuth		
EMVReturn	R	Card present return or credit of funds to card holder; returns do not reference a previous sale and are intended for use outside of the current open batch. Best practice is to match last four of card number.
EMVVoidSale (Reversal)	R	<ul style="list-style-type: none"> • <i>EMVVoidSale</i> and <i>EMVVoidReturn</i> may be supported with either card-present or <i>ByRecordNo</i> • <i>EMVVoidReturn</i> may alternatively be supported by <i>EMVSale</i>, or <i>SaleByRecordNo</i>
EMVVoidReturn	R	
EMVVoiceAuth	O	When required to call the card brand authorization service to obtain an <i>AuthCode</i> ; typically manually entered.
EMVZeroAuth	O	Card verification without authorization
AdjustByRecordNo	O	<i>ByRecordNo</i> TranCodes use a token (<i>RecordNo</i>) to adjust a gratuity on an <i>EMVSale</i> , reverse an <i>EMVSale</i> , or void an <i>EMVReturn</i>
VoidReturnByRecordNo	O	
VoidSaleByRecordNo	O	
SaleByRecordNo	O	Initiates a sale or return using a stored token/ <i>RecordNo</i> through the <i>dsiEMVUS</i> .
ReturnByRecordNo		

Note: XML examples appear later in this guide

EMVSale, EMVPreAuth and associated TranCodes

- *EMVSale* may be used to request a token by including *<RecordNo>* and *<Frequency>* in the request. **SaleByRecordNo** is also supported when using a stored token.
- The token can then be used for adding a tip (gratuity) with **AdjustByRecordNo**. Tokens may be requested as either *OneTime* or *Recurring* based your business needs. *AdjustByRecordNo* is not available for PIN Credit transactions or for any PIN Debit transactions. Applicable in restaurant environments and was introduced in place of *PreAuth—PreAuthCapture*.
- *EMVPreAuth* is applicable in restaurant environments in the same manner as a non EMV *PreAuth* is used—required when supporting *IncrementalAuths*; may be used to request a **OneTime** token by including *<RecordNo>* and *<Frequency>* in the request. **PreAuthCaptureByRecordNo** is supported when using a stored token.
- *IncrementalAuthByRecordNo* and *PartialReversalByRecordNo* are used after an initial *EMVPreAuth* in bar tab handling followed by a final *PreAuthCaptureByRecordNo*.

EMVVoidSale

- Always card-present on the EMV-enabled device. This transaction is sent as a reversal. Although the *TranCode* is *VoidSale*, the transaction request must include the *Process* and *AcqRefData* elements. This allows the request to be sent directly to the Issuer. The XML response will indicate “REVERSED”.
- *VoidSaleByRecordNo* (using a token) is available for card-not-present *VoidSales/Reversal*.

EMVReturn / EMVVoidReturn

- Use *EMVReturn* to run a card-present refund on an *EMVSale*.
- *EMVVoidReturn* voids a *Return*; card present.
- *VoidReturnByRecordNo* (using a token) is available for card-not-present *VoidReturns*.

EMVVoiceAuth

- A voice authorization is primarily used when Issuer returns a “Decline-Call referral” message with a *TextResponse* such as “CALL CENTER,” “CALL AMEX” or “CALL DISCOVER.” The merchant then contacts the automated voice-authorization system (using the number included in their MercuryPay Welcome kit) to obtain a six-digit authorization code (*AuthCode*). Receiving the *AuthCode* assures the merchant that the card is good and has credit available. Completing the voice authorization captures the funds for the merchant at settlement.

EMVZeroAuth

- Enables card verification without charging the card. Used to determine if a card has been reported lost or stolen; may be used in with AVS and CVV; may be used to initiate a bar tab.

Using Datacap's dsiPDCX® as a Complement to dsiEMVUS

The *dsiPDCX* will complement your integration to the *dsiEMVUS* if your application supports:

- MercuryGift and StoreCard
- CardLookup, FSA
- EBT Foodstamps, EBT Cash or eWIC
- Card-on-file programs using MercuryPay Tokenization

All transactions supported by *dsiEMVUS* will continue to be sent to *dsiEMVUS*. When implementing *dsiPDCX* together with *dsiEMVUS*, a common approach is to create two separate objects in the code, then determine at runtime which object the XML should be routed to.

- Use the same MerchantID (MID) for both *dsiEMVUS* and *dsiPDCX*.
- Share the same local NETePay. All transactions can be sent to the local, in-Store *NETePay* address **127.0.0.1** (or, if networked, the IP address of the computer that is running the *NETePay*); *NETePay* will then direct the transaction to MercuryPay's servers.
- Note that the Secure Device IDs follow a *dsiPDCX* specific convention (All CAPS): "DEVICENAME_MERCURY_E2E." For a current list of *dsiPDCX* SecureDevice IDs go to <https://www.dsidevportal.com/secure-devices/dsipdcx/>.
- Use **EMVParamDownload** to initialize the device. **There is no need to use the SecureDeviceInit for dsiPDCX** on a device after an *EMVParamDownload* (*dsiEMVUS*) has been performed. Executing a *SecureDeviceInit* will override the EMV parameters and may result in errors when EMV transactions are run.
- For more information about integrating to *dsiPDCX*, refer to Datacap's developer portal. <http://www.dsidevportal.com/>

MercuryPay's Tokenization with dsiEMVUS

TranCodes that end with **ByRecordNo** use MercuryPay's secure tokenization technology, MToken™. Merchant ID's are configured to enable the return of a token with each supported transaction, which can then be used for subsequent supported transactions (the supported transactions all end with "ByRecordNo".)

- For Gift/PrePaid, Debit, and EBT a *ByRecordNo* *TranCode* is unavailable. The Merchant ID configuration in this case ignores the token request and does not generate a *RecordNo* in the response
- Tokens may only be used with EMV CVMs returned as SIGNATURE or NONE. Card brands will reject transaction submitted *ByRecordNo* for CVMs returned as PIN VERIFIED.

Details of a dsiEMVUS XML Request and Response

The EMV transaction request and response use the same XML format as all of MercuryPay's Datacap solutions. The primary difference with EMV is that a *SecureDevice* is defined and used to complete the request. The transaction request is then sent to the *dsiEMVUS* client control which then communicates directly with the *SecureDevice*, builds out the XML with encrypted data and any other prompted options and then routes this through the local *NETePay* for processing.

- The Account element and other options such as Credit or Debit ApplicationID selection are handled at the device level.
- All Worldpay from FIS IP/MercuryPay EMV transactions use *SecureDevices* with point-to-point encryption enabled. We highly recommend using tokenization as well. The following example is an **EMVSale** and includes a token request (*RecordNo*). Each numbered item in this example is explained below:

dsiEMVUS XML Request

```

<TStream>
  <Transaction>
    <HostOrIP>127.0.0.1</HostOrIP>    <!--use to communicate with the local NETePay See Note-->
    <IpPort>9000</IpPort>
    ① <MerchantID>755847002</MerchantID>
    ② <LanelID>02</LanelID>    <!--for environments with multiple input devices-->
    ③ <TranCode>EMVSale</TranCode>    <!--indicates being sent to an EMV-enabled device-->
    ④ <SecureDevice>EMV_VX805_MERCURY</SecureDevice>    <!--EMV-enabled device-->
    ⑤ <ComPort>9</ComPort>
    ⑥ <InvoiceNo>452</InvoiceNo>
    <RefNo>452</RefNo>    <!-- RefNo = InvoiceNo in the request, as filler value-->
    ⑦ <Amount>
      <Purchase>8.18</Purchase>
    </Amount>
    ⑧ <SequenceNo>0010010000</SequenceNo>    <!--Required but reserved for future use-->
    ⑨ <RecordNo>RecordNumberRequested</RecordNo>    <!--token request-->
    <Frequency>OneTime</Frequency>
    ⑩ <PartialAuth>Allow</PartialAuth>
    ⑪ <Memo>Your POS Memo v1.0 EMVUS</Memo>
    <OperatorID>User</OperatorID>    <!--configurable-->
    <UserTrace>DEMO</UserTrace>
  </Transaction>

```

Note: <HostOrIP>: 127.0.0.1 is a local machine IP address indicating that the NETePay is installed on the same computer as the POS application. In networked environments using multiple POS workstations and a single NETePay server, this HostOrIP will always be the IP address of the computer where the NETePay is installed.

1. **<MerchantID> (MID)** MercuryPay merchant accounts are set up with a primary Merchant ID that is used to identify the account both internally and externally for POS configuration and for use in transmitting banking information and deposits.

MID settings are configured by Worldpay from FIS IP with a range of attributes that parallel the POS system's functionality. If not in sync with the POS system, these configuration flags can render the merchant location unable to process. For example, if the POS is processing using point to point encryption, then the MID must be enabled to allow point to point encryption.
2. **<LanelID>** is **required for multi-terminal merchants** for all card-present transactions sent to MercuryPay. Each terminal or input device must have a unique, alphanumeric identification with a maximum of 3 characters. No special characters, dashes or slashes.
3. **<TranCode>** indicates the type of transaction being submitted (*EMVSale*, *EMVReturn*, etc.) All EMV TranCodes start with "EMV".
4. **<SecureDevice>** This is the unique identifier for the device being used. Use the [GetDevicesInfo\(\)](#) command to get the *SecureDeviceID* for your device.
5. **<ComPort>** This is the actual or virtual COM Port the device is connected to. There are two ways with Datacap's NETePay solution to connect a device:
 - **RS-232/Serial** devices require a COM port value. Set this to the port number the device is connected to.
 - **USB** devices require a USB port and the appropriate software driver which will create a virtual COM port. The device driver is available from your Implementations Consultant.
 - **For TranCloud**, an IP/Ethernet interface is used.
6. **<InvoiceNo>** and **<RefNo>**

- The **InvoiceNo** tag in the transaction XML is a locally generated, incrementing transaction number with a very specific duplicate logic function on the Worldpay from FIS IP/MercuryPay platform. The local InvoiceNo counter does not reset and the end of business day. (The only time the InvoiceNo should not be incremented is in the rare case of a timeout.)
- Implementing the *InvoiceNo* as described is a certification requirement.
- The Worldpay from FIS IP/MercuryPay *InvoiceNo* logic may turn out to be different from your existing POS transaction numbering logic, especially in the case of split tenders.
- Each transaction that is processed with separate cardholder data, including each individual split tender, requires a unique *InvoiceNo*.

For example, if the POS presents a ticket number #1234 for \$100 and this ticket is split among four credit cards, then the POS would need to generate four separate receipts, each with its own unique *InvoiceNo* specific to each of the four cardholders.

InvoiceNo format: Length: 16 max, right 10 must be unique; Data Type: Numeric only

InvoiceNo logic: although a maximum length of 16 is allowed, only the right-ten digits are submitted for processing and are critical for identifying duplicate transactions. Therefore, these right 10 numbers must be unique per transaction. In addition, all-numeric filler zeroes, alpha characters, dashes or special characters may not be used.

- **RefNo** is used as an initial "filler value" in the transaction requests but is a true batch reference value assigned by Worldpay from FIS IP/MercuryPay in the transaction response. The response *RefNo* indicates that an approved transaction has been "referenced" and logged in the merchant's current batch file.
- It is important to log when an actual *RefNo* value is returned as you will use that *RefNo* value for subsequent transactions such as *VoidSales* or *Adjusts*. **Use the actual returned RefNo value** whenever submitting a subsequent transaction that modifies an originally referenced transaction. For example, use the actual *RefNo* value when running an *EMVVoidSale* against an *EMVSale*, or when using *AdjustByRecordNo* for a tip modification.
- *RefNo* is the item number logged and incremented within each batch for each approved transaction. The first transaction is assigned, for example *RefNo* 1001 or 5001; the second transaction is *RefNo* 1002, or 5002 and so on. Upon batch close, the *RefNo* counter resets back to 1001 or 5001 for the next batch.
- For initial transaction requests such as *EMVSale*, we recommend including the *RefNo* with a "filler value" (coded as *RefNo=InvoiceNo*). The transaction response will then return the **actual**, batch-specific *RefNo* value. The **actual** *RefNo* value is used in any subsequent transaction as a "reference to" the original transaction, such as *EMVVoidSale*.

7. **<Amount>** This contains the requested purchase amount and is always presented in two decimal places (0.00). It may also contain **gratuity**. To prompt for gratuity, include `<Gratuity>Prompt</Gratuity>`.
8. **<SequenceNo>** This is required to be included in the XML request, and is currently reserved for future use.
9. **<RecordNo> and <Frequency>** These two tags are required if implementing MercuryPay's tokenization, MToken. On the *MercuryPay* platform, tokens are referred to as **RecordNo**. A *RecordNo* will be requested with two tags:
 - The **Frequency** of the token requested (**OneTime** or **Recurring**), and
 - Either a request to receive a token (**RecordNumberRequested**) or the actual token returned in the original response.
10. **<PartialAuth>** Use of this data element on *EMVSale* is mandated by the Card Brands in card-present environments in response to the growing use of Prepaid and Card Branded Debit cards. Use of this tag will "allow" a partial approval if the balance on the card is less than the purchase amount requested.

- By implementing PartialAuth, the transaction will approve for the available balance on the card and will not decline the transaction. *The PARTIAL AP response requires you to build into your application split-tender functionality that recognizes the partial approval and prompts for a balance due.* If the cardholder is unable to complete the balance due transaction, the ISV is required to send a *Reversal/VoidSale* of the partially approved amount.

In the transaction response, two data elements can be used to recognize and calculate the Partial Authorization:

- The POS will be alerted to the partial approval with a *TextResponse* of **PARTIAL AP**, and
- The **<AuthorizeAmount>** will be less than the original purchase amount. It is important to compare the *AuthorizeAmount* to the original *Purchase Amount* to determine the balance due.

- <Memo>/<POSPackageID>** Placed in all XML transaction requests, the *Memo* tag and **<POSPackageID>** are used to identify the POS name and version. For dsiEMVUS integrations, this tag is now **<POSPackageID>**. This will help our technical support teams to more quickly identify and troubleshoot potential issues. Dynamically build your **<POSPackageID>** tag so that it pulls the product name and version from your application.

dsiEMVUS XML Response

All transactions routed through Worldpay from FIS IP/MercuryPay return a similar XML response. This response, the **<RStream>**, contains only non-sensitive, truncated or tokenized cardholder data elements, and may be used to validate the transaction, build receipts, build local logs and records for internal POS reporting, and for storing for subsequent use.

The response XML is divided into three sections:

- <CmdResponse>** indicates the origin of the response (Processor, Server, Client), the status of the authorization response (Approved, Declined, Success, Error) and important verbiage in the **<TextResponse>** field.
- <TranResponse>** provides all of the response details, transaction routing identifiers, and (if requested) a token for subsequent use.
- <PrintData>** provides information to be printed on the receipt.

```
<?xml version="1.0"?>
```

```
<RStream>
```

```
  <CmdResponse>
```

```
    ① <ResponseOrigin>Processor</ResponseOrigin>
```

```
    ② <DSIXReturnCode>000000</DSIXReturnCode>
```

```
    ③ <CmdStatus>Approved</CmdStatus>
```

```
    <TextResponse>APPROVED</TextResponse><!--useful for troubleshooting declines/errors-->
```

```
    <SequenceNo>0010010000</SequenceNo>
```

```
    <UserTrace>DEMO</UserTrace>
```

```
  </CmdResponse>
```

```
  <TranResponse>
```

```
    ① <MerchantID>755847002</MerchantID>
```

```
    ② <AcctNo>*****0010</AcctNo>
```

```
    <AuthCode>30592A</AuthCode>
```

```
    ③ <CardType>VISA</CardType>
```

```
    <TranCode>EMVSale</TranCode>
```

```
    ④ <CaptureStatus>Captured</CaptureStatus>
```

```
    ⑤ <RefNo>1001</RefNo><!--note actual reference number is returned-->
```

```
    <InvoiceNo>452</InvoiceNo>
```

```
    <OperatorID>User</OperatorID>
```

```
    ⑥ <Amount>
```

```
      <Purchase>8.18</Purchase>
```

```
      <Authorize>8.18</Authorize>
```

```
    </Amount>
```

```

⑦ <RecordNo>dyowoK8BIOAG3UQ7atfo/8ej6JShzNBtEkxo1EbSmaAiEgUQECIQGRZI</RecordNo>
    <!--token is returned in the response -->
⑧ <AcqRefData>b8238953#20#20#20#20#20#20#20e00</AcqRefData>
    <ProcessData>|00|410100500000</ProcessData>
⑨ <EntryMethod>CHIP</EntryMethod>
    <Date>07/06/2015</Date>
    <Time>10:12:38</Time>
    <ApplicationLabel>Visa Credit</ApplicationLabel>
    <AID>A0000000031010</AID>
    <TVR>8000008000</TVR>
    <IAD>06010A03602000</IAD>
    <TSI>6800</TSI>
    <ARC>00</ARC>
    <CVM>SIGN</CVM>
</TranResponse>
<PrintData> <!--print data used for receipts -->
    <Line1>.MERCHANT ID: 337234005</Line1>
    <Line2>.CLERK ID: Test</Line2>
    <Line3>.</Line3>
    <Line4>.      SALE      </Line4>
    <Line5>.</Line5>
    <Line6>.*****0010</Line6>
    <Line7>.VISA      ENTRY METHOD: CHIP</Line7> <!--note entry method-->
    <Line8>.DATE: 07/09/2015 TIME: 07:40:26</Line8>
    <Line9>.</Line9>
    <Line10>.INVOICE: 07092015073925283</Line10>
    <Line11>.REFERENCE: 1001</Line11>
    <Line12>.AUTH CODE: 30592A</Line12>
    <Line13>.</Line13>
    <Line14>.AMOUNT      USD$ 8.18</Line14>
    <Line15>.      =====</Line15>
    <Line16>.TOTAL      USD$ 8.18</Line16>
    <Line17>.</Line17>
    <Line18>.  APPROVED - THANK YOU  </Line18>
    <Line19>.</Line19>
    <Line20>.I AGREE TO PAY THE ABOVE TOTAL AMOUNT</Line20>
    <Line21>.ACCORDING TO CARD ISSUER AGREEMENT</Line21>
    <Line22>.(MERCHANT AGREEMENT IF CREDIT VOUCHER)</Line22>
    <Line23>.</Line23>
    <Line24>.</Line24>
    <Line25>.</Line25>
    <Line26>.x_____</Line26> <!--Chip and Signature-->
    <Line27>.  Cardholder Signature  </Line27>
    <Line28>.</Line28>
    <Line29>.</Line29>
    <Line30>.Application Label: CREDIT VISA</Line30>
    <Line31>.AID: A0000000031010</Line31>
    <Line32>.TVR: 0000008000</Line32>
    <Line33>.IAD: 06010A03608000</Line33>
    <Line34>.TSI: F800</Line34>
    <Line35>.ARC: 00</Line35>
    <Line36>.CVM: SIGN</Line36>
</PrintData>
</RStream>

```

<CmdResponse> tags

1. **<Response Origin>** indicates the source of the response:
 - **Client** indicates that the response is from the *dsiEMVUS* control
 - **Server** indicates it is from Worldpay from FIS IP/MercuryPay's platform servers or the locally installed NETePay

- **Processor** indicates that the response is from Worldpay from FIS IP processing networks
2. **<DSIXReturnCode>** contains either a server-specific or dsiEMVUS-specific TCP/IP six-digit code corresponding to a single or dynamically generated message text. See the appendix for details on these codes.

3. **<CmdStatus>** and **<TextResponse>**

<CmdStatus> indicates the outcome of the transaction:

- For Credit transactions: the value returned will be **Approved** or **Declined**.
- For Admin requests: the value returned will be **Success** or **Error**. Error text responses are listed in the Appendix.

<TextResponse> is used to qualify the **<CmdStatus>**.

Approved Text Responses

Value	Description
APPROVED	transaction approved by processor
PARTIAL AP	The response is based on the use of <PartialAuth> ALLOW</PartialAuth> in the request; may require a split tender and a balance-due prompt; always confirm <Authorize> amount approved.
AP NEW INFO or AP-NEW INFO	optional message; account information is updated—this is often the address or expiration date
AP* (Reports as AP DUPE)	Indicates that duplicate logic is enabled and has detected the same card, the same amount, and the same invoice number in a second transaction in the same batch; it is paired with <CmdStatus>Approved</CmdStatus> and assures that the card holder is charged only once.
AP – NOT CAPTURED	Credit, Debit response: Transactions are being sent to the back-end processing networks but are not being captured or logged with the issuing bank due to a back-end network outage. It requires the transaction to be resent for authorization.
REVERSED	Credit reversal response: The transaction was acknowledged as reversed by the card -issuing bank when sending a reversal (Void Sale + AcqRefData + ProcessData).

Declined Text Responses

Declined *TextResponse* values vary according to the source of the message and whether the decline was due to an issuer-level response or if there was an error in the processing of the transaction. Below is a condensed list:

Value	Description
DECLINE	Authorization was declined based on issuing bank response. Card brand regulations stipulate that declines should not be re-attempted but should be followed-up with a request for a second form of payment.
CALL xxx Referrals	Issuer has placed a notice on the card and requests the merchant to call for a Voice Authorization. xxx = "CENTER", "AMEX", or "Discover".
PIC UP	Authorization declined; authorization network requests picking up the card and reporting. See card brand regulations for details and appropriate action.
DECLINED-CV2 FAIL	Visa response for CVV2 mismatch. (AMEX uses INVLD CID)
INVALID EXP DATE	Expiration date entered is incorrect, invalid or no longer current.
INVALID PIN	Incorrect PIN entered on Debit or EBT transaction.
UNAUTH USER	Merchant setting does not allow debit (may also be seen with EBT).
NO TRAN FOUND	Failed Void Sale/Reversal attempt
INV ITEM NUMBER	Typically triggered by an invalid RefNo used in <i>VoidSales</i> or an <i>Adjust</i> .

Note: It is important that the POS interprets and passes appropriate response messages, as well as the corresponding alerts and prompts to the end-users of their system. Merchants should be able to see the *TextResponse* of decline messages such as **Call CENTER**, **PIC UP**, etc., so they can take appropriate action if needed.

<TranResponse> tags

The **<TranResponse>** section of the response contains transaction details:

1. **<MerchantID>** is returned without the "=" sign and "Nick Name".
2. **<AcctNo>** and **<AuthCode>**
 - **<AcctNo>** returns the Account number with all but the last four digits of the number masked (*****)and,
 - **<AuthCode>** may be logged and used in conjunction with a *Reversal* or *VoidSale* with *AcqRefData* and *ProcessData*.
3. **<CardType>** and **<TranCode>** indicate the card type entered and the *dsiEMVUS* transaction code processed.
4. **<Capture Status>** will inform the POS if the transaction was **Captured** or **Not Captured**.
5. **<RefNo>** and **<InvoiceNo>** The *RefNo* returned in the *TranResponse* is the actual, *true* transaction reference number returned for any approved transaction in the current open batch. For any subsequent transactions such as *VoidSale* or *Adjusts*, this *true* **<RefNo>** must be used in place of the filler value. The *InvoiceNo* is passed back directly from the original request.
6. **<Amount>** tags (Purchase and Authorized) always return both the original purchase amount and the actual authorized amount. Always compare these two amounts for partial approvals. Additional amount tags vary based on the transaction and *TranCode* used, such as *Gratuity*.
7. **<RecordNo>** is the returned dynamic token to use in place of card data for any follow-up transactions. With each use a new token is returned. *RecordNo* may be stored for use with next *ByRecordNo* transactions and then replaced with each new response token, discarding previous token. Min-Max length from 48 to 100 alpha-numeric and some special characters.
8. **<AcqRefData>** and **<ProcessData>** contain multiple subfield strings of card association required identifiers, market data and transaction routing references Allow Min-Max length from 1-200 alpha-numeric characters for *AcqRefData* and 12-40 alpha-numeric characters for *ProcessData*. Both fields must be submitted with all reversal requests.
9. **<EntryMethod>** through **<CVM>** are EMV specific tags:
 - **Entry Method:** Indicates how the card information was obtained:
 - Chip
 - Contactless (MSD/Non EMV Grade)
 - Swiped
 - Chip Fallback/Swiped – if the chip insert could not be read
 - Chip/Manual – manually entered chip card
 - Manual – manually entered non chip card

Date: Transaction date

Time: Transaction Date

Application Label: Indicates card brand and transaction type (credit, debit, etc.)

AID - Application Identifier

TVR - Terminal Verification Results

IAD - Issuer Application Data

TSI - Transaction Status Indicator

ARC - Application Response Code

CVM - Card Holder Verification Method (CVM)

Note: **Fallback** is a mechanism within the chip card reader that detects a bad chip read. If this occurs, the reader prompts the consumer to swipe the card instead. For example, the VX805 will display, "**APPL NOT AVAILABLE USE MAG STRIPE**" followed by, when the card is pulled out, "**Tap or Swipe card**". If the tap or swipe fails, manual entry may optionally be used.

<PrintData>

This section of the response contains the EMV receipt data that the Datacap solution makes available as a convenience for developers. Building out all of this data into your merchant receipts is considered a best practice although specific card brand receipt requirements vary. A receipt review is a certification requirement and card brand requirements may be discussed with your Implementations Consultant. More details are provided in the **Receipt Requirements** section below.

EMV Receipt Requirements

Per the card brands and EMVCo, if a transaction response indicates APPROVED, DECLINED, ERROR or CANCELLED, it is considered a best practice to be able to produce a receipt. See the table below for the minimal requirements.

Creating a receipt template that mirrors the combined requirements of all card brands was the direction taken with Datacap's <PrintData> portion of the dsiEMVUS response. If <PrintData> is returned in the response, this is an indication that a receipt is considered required and available for generation.

- The merchant's copy may be stored electronically.
- Customer copies must always be available upon customer request.

Note: EMV receipt data is returned to the POS in the <PrintData> section of the XML response. The POS application supplies the merchant receipt header information (typically merchant name, address, phone number and/or email) and the footer information (**Cardholder Copy** and **Merchant Copy**).

- If integrators prefer to build out custom receipts per card brand requirements, details are included in the Datacap's dsiEMVUS Interface Specification available upon request at <http://www.dsidevportal.com/>
- *ByRecordNo* transactions (that is, transactions using a token) do not generate *PrintData*, and receipt generation would follow standard non-EMV criteria.

Cancelled and Declined Receipts

The *dsiEMVUS* generates different XML responses for errors and cancelled transactions. It is important that these messages are communicated as required to the POS operator and that this information is passed along to the card holder via the receipt, if applicable.

MasterCard recommends a declined receipt as an optional requirement and American Express recommends that receipts be made available on declined transactions.

How dsiEMVUS handles Signature, PIN Verified and No CVM

- **Credit Chip and Signature.** If the cardholder verification method (CVM) is determined to be chip and signature (CVM: SIGN), or if the card presented is a non-chip card, then the *PrintData* will be generated with a "Cardholder Signature" line.
- **Card Brand update on Signature requirements:** EMV-enabled merchants are no longer required to collect a signature on transactions. The outlier is MasterCard. MasterCard still requires either Signature or PIN on contactless transactions over the \$50.00 limit. Signatures will still be considered compelling evidence in some disputes.
 - Approved fallback transactions remain the Issuers obligation on EMV enabled devices. Signature is irrelevant.
 - Key entered transactions will always be merchant's liability for the purposes of disputes; there are no representative rights in this case, so signature is also irrelevant in a dispute.
 - Merchants who accept gratuity should give careful consideration to their business needs before suppressing the signature line.

- CVM settings on the terminals should not be changed; terminals are still required to give the consumer the option between signature and PIN even if the signature is not collected.
- **Credit Chip and PIN.** If the cardholder is prompted to enter a PIN during a transaction and does so successfully, the *PrintData* will indicate that the cardholder was PIN VERIFIED. If PIN entry is unsuccessful, the *PrintData* will include a signature line for the cardholder to sign.
 - PIN Bypass is available on the latest generation of Verifone devices (P400) although the success or failure of the transaction may be determined by the card issuing bank.
 - A PIN Preferring option is available in the Datacap PSCS configuration server.
- **No CVM.** In the event of a “NO CVM”, which is allowed in quick-serve restaurants and kiosks, neither a signature line nor PIN prompt is required.
- **EMVReturn and EMVVoidSale.** The *PrintData* returns with “Merchant Signature”.
- **EMV Contactless.** A Datacap PSCS configuration setting is required to prompt for PIN over a certain dollar limit. This configuration defaults to \$50.00.

Worldpay from FIS IP EMV Card Brand/Network Minimum Receipt Requirement Summary

EMV Receipts Required by Card Brand

	American Express	VISA	MasterCard	Discover/JCB
Sale/Refund	Required	Required	Required	Required
Decline	Recommended		Optional	
Void	Recommended			

EMV Tags Required by Card Brand Network

	Application ID (AID)	EMV CHIP INDICATOR	APPLICATION PREF. NAME	APPL. LABEL*	PIN STATEMENT	CRYPTO-GRAM
AMEX (approved)	Required	Required	Required***	Required	Required***	Preferred
AMEX (declined/void**)	Required	---	Required***	Required	---	Preferred
VISA	Required	---	Required***	Required	---	---
MC (approved)	Required	---	Required***	Required	---	Optional
MC (declined)	Required	---	Required***	Required	---	Optional
DISCOVER/ JCB	Required	---	Optional	Optional	Optional	Optional

*Only needs to be printed if Application Preferred Name is not present. However, there is no requirement that prevents the printing of the Application Label in conjunction with the Application Preferred Name even when the Application Preferred Name is present.

**For void transactions, chip data is only required if the void occurs after the chip interaction occurs.

***Required where present.

Additional MercuryPay/Datacap Features

MercuryPay and Datacap continue to add new features to platform functionality. The section details **Store and Forward**, **Pay at the Table** and **Account Updater**.

Store and Forward Functionality

Overview

Store and Forward is a secondary, back-up option of credit card acceptance enabled typically in the event of internet connectivity down-time. Store and Forward may be applicable as a temporary solution for businesses needing to accept payments in environments without access to the internet, such as trade shows or farmer's markets. Optionally, business may elect to process offline transactions even with a working internet during times of peak business demand.

When Store and Forward is enabled, it allows merchants to store transactions offline either in the absence of a working Internet connection or based on specific business use case. Transactions are stored offline until either internet connectivity has been restored or the business need subsides. Then, with a local command, the offline transactions are forwarded to Vantiv for a valid card issuer authorization. From the cardholder's perspective, the transaction flow is unchanged, yet the important distinction for the merchant is that the transaction is not authorized in real time—and may in fact decline when forwarded.

Where there are benefits to this functionality in maintaining transaction up-time especially during times of internet uncertainty, there are also risks and an assumption of liability which need to be carefully considered.

Setting Risk Controls, Forcing Offline and Secure Storage

Two mechanisms control the parameters of Datacap's Store and Forward. First, the capability to enable Store and Forward is defined in Datacap's Configuration Server, PSCS, when the merchant's Deployment ID is configured and downloaded to the locally installed NETePay. Second, within the POS, the integrator will add three tags to the standard XML request: **<ForceOffline>**, **<MaxTransactions>** and **<OfflineTransactionPurchaseLimit>**. With these in place, and with the NETePay enabled, the transaction request will route to a secure offline database in the local NETePay for encrypted storage.

Although a valid card issuer authorization is not obtained and as long as all the configured maximum values are not exceeded, an offline "Approved" response is passed to the POS.

The following is supplied in the response to easily identify offline activity:

- ResponseOrigin: **StoreAndForward**
- TextResponse: **Transaction stored offline.**
- AuthCode: **OFFLIN**
- RefNo will create a new item number starting with **000000000001**
- RecordNo will follow this same pattern with **SAF:000000000001**
- All other standard transaction receipt information in the form of PrintData

Note: To clearly distinguish your POS application is in offline processing mode, or to guard against unintentionally enabling Store and Forward, consider altering the screen color of your user interface or highlighting the payment screen to indicate the system is processing offline.

Going back Online, Sending SAF_FowardAll and Comparing Totals

When the internet is reestablished, or the offline event is ready to go online, signifying "N" in **<ForceOffline>** tag, (or removing it altogether) allows NETePay to switch back to standard online processing. The previously completed offline requests remain securely stored.

Any time before the merchant's online batch is closed, the offline transactions must be "forwarded," to the processor, obtain a valid issuer authorization and be added to the merchant's open online batch. This is performed by sending the admin command: **SAF_ForwardAll**. NETePay sends the transactions to the processor and returns a Store and Forward summary report of all approved, declined or errored transactions.

Note: It is important for the POS to use this offline summary report, compare it with the local offline totals and make any necessary adjustments. As the processor has no visibility into a merchant's offline transactions, any discrepancies with the offline totals are the responsibility of the merchant!

If the offline transactions are not forwarded to the online batch before the batch is closed, an error will be passed back notifying the POS that there are still transactions in the offline batch and these should be forwarded BEFORE the batch is closed. After settlement, all offline transactions are securely deleted.

Balancing Risks and Waiver of Liability

When implementing and utilizing Store and Forward, it is important that merchants understand there are inherent risks when not obtaining an authorization at the time of the transaction. When enabling Store and Forward, the merchant accepts all liability, including loss of revenue due to declined or failed transactions, chargebacks, and penalties.

- It is important to acknowledge that transactions processed via Store and Forward are high risk and may be declined, error or otherwise fail to process when forwarded to the processor.
- Vantiv Integrated Payments makes no warranty, expressed or implied, with respect to services, processing or acceptance of Store and Forward transactions.
- The merchant assumes all liability when accepting to process in a Store and Forward/Offline manner.

Getting Started: Configuring the Store and Forward/Offline Settings In PSCS and in the POS

By default, NETePay installations are configured as online processing only. Offline operation is optional and elected, and must be provisioned through Datacap's PSCS system (Payment System Configuration Server) at the time the merchant's NETePay activation is performed. This configuration is typically handled by Vantiv's Technical Support team at the time of the merchant on-boarding.

The PSCS configuration process establishes four parameters for Store and Forward operation:

PSCS Parameter		Description
1.	Enable offline operation	Enables Store and Forward/Offline capability in NETePay
2.	Maximum stored transaction count	Sets the maximum number of stored offline transactions permitted before offline processing will be halted and settlement must be performed. This count cannot be exceeded by the integrator supplied <MaxTransactions> tag value.
3.	Total purchase limit for offline transactions	The <u>net maximum dollar amount</u> of all stored offline transactions allowed before settlement must be performed.
4.	Maximum time allowed to store offline transactions	This is the age of the oldest stored offline transaction allowed before settlement must be performed.

Note: The maximum stored transactions, total purchase limits and time allowed for storage are driven by each merchant's business risk tolerances and are set at the time of the NETePay stage file is created. These may be fine-tuned as needed by calling Vantiv's technical support.

Developers also have local control of the offline operation with three XML tags included in transaction requests to NETePay.

The merchant's POS, when enabling offline, will need to include the following with each transaction request:

XML Tag	Description
<ForceOffline>	This flag allows an integrator to force offline processing of transaction regardless of available Internet connection to a processor if <MaxTransactions> and <OfflineTransactionPurchaseLimit> have not been exceeded.
<MaxTransactions>	This is the maximum number of transactions which can be captured in offline mode before settlement must be performed.
<OfflineTransactionPurchaseLimit>	Maximum transaction amount allowed for offline processing. If purchase amount is greater, then the response will return a Validation error.

Forwarding Offline Transactions for Settlement

When the POS has accumulated transactions from an offline event and these transactions are within the configuration limits, then the POS will send a "forward" command to add the offline transactions to the merchant's batch. Always follow this order:

- Forward the offline transactions using **SAF_FowardAll** to the merchant's open batch file, noting the offline summary, and
- Perform the BatchSummary and BatchClose as usual

Note: If tip adjustments are supported, or if there is a need to void an offline transaction, these must be completed in the **ForceOffline** mode BEFORE performing the **SAF_FowardAll**.

- It is NOT recommended for a merchant to adopt Store and Forward when supporting Time initiated/automatic batch settlement.
- In the event that either PSCS or local configured offline transaction limits are reached, further offline transactions will receive an error. The settlement process described above is then required before additional offline processing can continue.
- The Datacap Store and Forward utility can hold and maintain a total of 9,999 transactions. This total amount is per item stored (i.e., EMV PreAuth and EMV Capture will count as 2).

EMV Transactions Allowed for Offline/Store and Forward Processing

- Sale, EMVPreAuth, SaleByRecordNo, VoidSale, VoidSaleByRecordNo
- Return, VoidReturn, VoidReturnByRecordNo
- VoiceAuth
- AdjustByRecordNo, EMVPreAuthCaptureByRecordNo

Additional XML Tags when Supporting Store and Forward

```
<ForceOffline>Y</ForceOffline>
<MaxTransactions>100</MaxTransactions>
<OfflineTranactionPurchaseLimit>00.00</OfflineTranactionPurchaseLimit>
```

Element	Req	Min	Max	Type	Description/Value
ForceOffline	O	1	1	AN	"Y" Force Offline processing of transaction regardless of available internet connection to Processor if <MaxTransactions> and <OfflineTranactionPurchaseLimit> have not been exceeded. "N" Process transactions online if Internet connection available to the processor.

MaxTransactions	O	1	8	N	Total number of offline transactions allowed to be stored
OfflineTransactionPurchaseLimit	O	3	10	AN	Amount (with 2 place decimal – eg 29.95) Maximum transaction amount allowed for offline processing. If purchase amount is greater, then purchase amount must process online.

Store and Forward Merchant Set up Best Practices

There are 4 Datacap and 3 local POS settings that need to be selected when enabling S&F.

- The Datacap setting are selected at the time of enrollment setup or anytime thereafter if required by our technical support team and in conjunction with the Merchant's sale's liaison.
- It is highly recommended to make sure the merchant understands the shift in liability during Offline mode.
- These settings are driven by the merchant's business risk tolerance and may vary from merchant to merchant. These should be discussed carefully with the merchant as they can impact business flow, settlement and funding. Here is a table of best practice recommendations:

Datacap Store and Forward Settings

	Datacap Parameter	Description	Best Practice when in Offline Mode
1.	Enable offline operation	Enables Store and Forward/Offline	This must be "on" to allow S&F/Offline
2.	Maximum stored transaction count	Sets the maximum number of stored offline transactions permitted before offline processing will be halted and settlement must be performed. This count cannot be exceeded by the integrator supplied <MaxTransactions> tag value.	Best to figure slightly higher than average number of transactions/day. Example: Store gets 100 trans /day. Set at 125 or slightly higher. When this value is reached the POS will receive a notification and are required to send the ForwardAll command in order to continue offline processing. The POS will be setting a parallel value.
3.	Total purchase limit for offline transactions	The <u>net maximum dollar amount</u> of all stored offline transactions allowed before settlement must be performed.	This is a total amount, not a per transaction amount; (per transaction amount is set at the POS) Best practice would be to set this at daily average total sales, depending on risk tolerance of merchant. If set too low could cause an interruption in card acceptance as in the above, but may be preferred. If the max is reached, then location must send the ForwardAll command to move Offline trans to open batch.
4.	Maximum time allowed to store offline transactions	This is the age of the oldest stored offline transaction allowed before settlement must be performed.	The Card brands all recommend no greater than 24 hours. If Max time is reached, an notification will be passed to the POS and the location must send the ForwardAll command to move offline transactions to open batch.

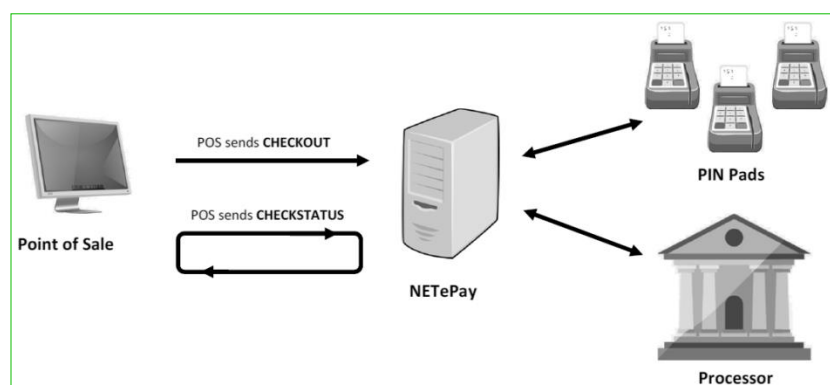
Local POS Store and Forward Settings

	Local POS Settings	Description	Best Practice when in Offline Mode
1.	Force Offline	This function allows the POS to enter offline processing of transaction regardless of available Internet connection to a processor if Max Transactions <i>and</i> Offline Transaction Purchase Limit have not been exceeded.	This is basically the "on" button for Offline. When on, all transactions are stored locally, unauthorized and require a ForwardAll command to move into the open batch. POS should alert end user that the POS is in Offline mode.

2. Max Transactions	This is the maximum number of transactions which can be captured in offline mode before settlement must be performed.	Base on average daily transaction count and cannot exceed the value configured at Datacap. When this value is reached the POS will receive a notification and are required to send the ForwardAll command in order to continue offline processing.
3. Offline Transaction Purchase Limit	Maximum transaction amount allowed for offline processing. If purchase amount is greater, then the response will return a Validation error.	This is a per transaction max amount. If average ticket is \$25.00 set in this range +/- risk tolerance to say \$40.00. Similar to a floor limit. If purchase amount is greater, then the response will return a Validation error and will need to be rerun at a lower amount.

Datacap's Pay-At-The-Table

Bringing the payment to the cardholder, be that while the customer is waiting in line to pick up lunch or tableside after a meal, enhances the customer experience and can reduce the time involved with checkout process. Datacap Systems, Inc., a longtime partner with Worldpay, has developed their NETePay and dsiEMVUS software to support pay-at-table and line busting with minimal changes required in your POS application.



- The Pay-At-Table transaction flow sequence is adaptable to your current software:
 - Once the customer asks for the check, send a *Checkout* command from the POS to a supported device using a pad's IP or MAC address. This puts the transaction request "*InQueue*", and the pad will now display your custom message, such as "Please begin Table 21"; your POS is immediately returned a *PayAtTheTableID* for tracking.
 - The PIN pad is then brought to the table and the checkout process is handed over to the customer following the prompts on the display—they confirm the amount, add a tip, process their card and complete the sale. All during the customer checkout process, the POS, using the *PayAtTheTableID* can send regular *CheckStatus* commands to the PIN pad. If the transaction has not yet been completed the status will return "*InQueue*" or if the transaction is in-flight to the processor, the status will return "*InProcess*". Completed transaction will return the standard authorization message: Approved, Declined or Error.
 - In the event that an "*InQueue*" transaction needs to be cancelled, use the *CancelCheckout* command.
- The Datacap solution uses an asynchronous communication process method. Because of this, Pay-At-Table is designed to be flexible for restaurant and retail environments as it allows **multiple transactions to be queued up to a single PIN pad device or multiple transactions to be sent to multiple devices**. This allows the POS to service multiple tables/customers simultaneously.
- Additionally, a custom message can be sent to the device specific to each transaction, (or table or check number)—which is great for split-tenders at a single table. This message, and any subsequent queued messages, is displayed on the screen from the onset of the transaction sequence.

This section of the guide details the PIN Pad Pay-at-Table transaction flow and XML tags required to communicate with supported PIN Pads and process Wi-Fi based transactions using Datacap's NETePay and dsiEMVUS.

Datacap's Pay-At-The-Table Integration Prerequisites

1. Datacap Systems 5.07.33 (or greater) NETePay downloaded and installed
2. Datacap Systems dsiEMVUS 1.39
3. Requires Pay-At-Table to be enabled on the Datacap configuration server, PSCS and downloaded to the local NETePay and EMVParamdownload to the Pad.
4. Wi-Fi enabled P2PE/EMV PIN Pads: Legacy models: Ingenico iSMP4, iWL258 (built in printer), Tetra Line: Link 2500 or Move 5000 (built in Printer); networked and configured to your local Wi-Fi router.

Note: Latest components always available at <http://www.datacapepay.com/downloadmenu/>

Pay-at-Table video demonstrations available at <https://www.datacapsystems.com/pay-at-table>

Pay-at-Table Transaction Flow:

Checkout (queues an EMVSale) → CheckStatus (to determine the outcome) → Checkout

The *Checkout* TranCode works with the dsiEMVUS and your POS to queue up a pay-at-table transaction (EMVSale) at the designated PIN Pad and returns to the POS a local identifier called the **PayAtTheTableID**.

The *Checkout* XML request mirrors an *EMVSale* transaction request (with tip prompt) plus the following additional pay-at-table tags:

1. Port Values: *IpPort* 9400 and *PinPadIpPort* 12000 are standard when using Wi-Fi
2. TranCode value is "Checkout"; a TranType is not required
3. *AutoPrintCustomer* and *AutoPrintMerchant* are used to communicate a print commands to the PIN Pad. Values are "No", "Yes" and "ApprovalsOnly". (String 1-20 Chars)
4. The *<Question>* tag is used for a PIN Pad display message at the start of a Checkout. Defaults to "Begin Payment". May also be useful for table or ticket number. (String 1-20 Chars)
5. *HeadLine1-4* and *FooterLine1-2* used for Receipt header and footer. (String 1-40 Chars)

Pay-at-Table XML Example: Checkout -- Request

```
<TStream>
<Transaction>
  <HostOrIP>127.0.0.1</HostOrIP>
  <IpPort>9400</IpPort><!-- WiFi Port -->
  <PinPadIpAddress>192.168.0.18</PinPadIpAddress> <!-- May use either IP or MAC -->
  <PinPadMACAddress>54E14060E551</PinPadMACAddress>
  <PinPadIpPort>12000</PinPadIpPort> <!-- PIN Pad Port on local network -->
  <MerchantID>592885123</MerchantID>
  <LaneID>001</LaneID>
  <OperatorID>52</OperatorID>
  <UserTrace>Worldpay1</UserTrace>
  <TranCode>Checkout</TranCode> <!-- New Tag -->
  <SecureDevice>EMV_IWL258_MERCURY</SecureDevice>
  <ComPort>1</ComPort>
  <Account>
    <AcctNo>Prompt</AcctNo> <!-- Note Prompt for AcctNo and Gratuity -->
  </Account>
  <InvoiceNo>0002</InvoiceNo>
  <RefNo>0002</RefNo>
  <Amount>
```



```

    <Purchase>10.00</Purchase>
    <Gratuity>Prompt</Gratuity> <!-- Note Prompt for AcctNo and Gratuity -->
  </Amount>
  <SequenceNo>0010010010</SequenceNo>
  <PartialAuth>Allow</PartialAuth>
  <RecordNo>RecordNumberRequested</RecordNo>
  <Frequency>OneTime</Frequency>
  <AutoPrintCustomer>ApprovalsOnly</AutoPrintCustomer>
  <AutoPrintMerchant>Yes</AutoPrintMerchant> <!-- for pads with printers -->
  <Question>Begin Payment</Question> <!-- Use to customize screen message -->
  <HeaderLine1>MERCHANT NAME</HeaderLine1>
  <HeaderLine2>MERCHANT STREET</HeaderLine2>
  <HeaderLine3>CITY, STATE ZIP</HeaderLine3>
  <HeaderLine4>123-456-7890</HeaderLine4>
  <FooterLine1>Thanks for shopping with us</FooterLine1>
  <FooterLine2>Please come again</FooterLine2>
</Transaction>
</TStream>

```

Pay-at-Table XML Example: Checkout – Response

This response confirms the status of the request and returns the **PayAtTheTableID**. The *PayAtTheTableID* (String, 1-40 chars) is to be stored and used in subsequent *CheckStatus* or cancel requests.

Note: The *TextResponse* "InQueue" indicates that the transaction request has been sent to the pad, and is awaiting cardholder input.

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>PATT Server</ResponseOrigin> <!-- local PATT Server -->
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Success</CmdStatus>
    <TextResponse>InQueue</TextResponse> <!-- InQueue indicated in-flight -->
    <PayAtTheTableID>20190120105414000006</PayAtTheTableID>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>Worldpay1</UserTrace>
  </CmdResponse>
</RStream>

```

Pay-at-Table XML Example: CheckStatus – Request

CheckStatus is used to "poll" the status of the Pay-at-Table transaction. It is recommended to poll the command at regular intervals to determine the outcome of the transaction.

Note: The CheckStatus uses the *PayAtTheTableID* as well as the PIN Pad's IP or MAC Address in the request.

```

<TStream>
  <Transaction>
    <HostOrIP>127.0.0.1</HostOrIP>
    <IpPort>9400</IpPort> <!-- WiFi Port -->
    <TranCode>CheckStatus</TranCode> <!-- New "polling" TranCode -->
    <PayAtTheTableID>20190120105414000006</PayAtTheTableID>
    <PinPadIpAddress>192.168.0.18</PinPadIpAddress> <!-- May use either IP or MAC -->
    <PinPadMACAddress>54E14060E551</PinPadMACAddress>
    <PinPadIpPort>12000</PinPadIpPort> <!-- PIN Pad Port on local network -->
    <MerchantID>5928885123</MerchantID>
    <LaneID>001</LaneID>
    <SequenceNo>0010010010</SequenceNo>
    <OperatorID>52</OperatorID>
    <UserTrace>Worldpay1</UserTrace>
  </Transaction>
</TStream>

```

Pay-at-Table XML Example: CheckStatus – Response

If the cardholder has completed their interaction with the device and transaction has been authorized the CmdStatus from *CheckStatus* will return a standard processing message "Approved", "Declined" or "Error".

If the transaction had not yet been authorized, the *CheckStatus* Text Response will return "*InQueue*". If the customer has completed their interaction, submitted the transaction but the response is "in-flight" to the processor, the response on the *CheckStatus* Text Response will return "*InProcess*".

```
<RStream>
  <CmdResponse>
    <ResponseOrigin>PATT Server</ResponseOrigin> <!-- local PATT Server -->
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Success</CmdStatus>
    <TextResponse>InProcess</TextResponse> <!-- This field tells you the status -->
    <PayAtTheTableID>20190120105414000006</PayAtTheTableID>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>Worldpay1</UserTrace>
  </CmdResponse>
</RStream>
```

Pay-at-Table XML Example: CancelCheckout – Request

Use *CancelCheckout* to initiate a cancel to an "*InQueue*" *Checkout* request. Only if the status of the response from *CheckStatus* is "*InQueue*" can the transaction be canceled using the *CancelCheckout* command. The transaction cannot be canceled if the *CheckStatus* is "*InProcess*" or completed. Once the transaction is "*InProcess*", it may only be canceled using the PIN Pad's cancel button.

Note: Always send a follow-up *CheckStatus* after the *CancelCheckout* to verify the cancel command.

```
<TStream>
  <Transaction>
    <HostOrIP>127.0.0.1</HostOrIP>
    <IpPort>9400</IpPort>
    <TranCode>CancelCheckout</TranCode> <!-- Cancel TranCode -->
    <PayAtTheTableID>20190120105414000006</PayAtTheTableID> <!-- Note new ID -->
    <PinPadIpAddress>192.168.0.18</PinPadIpAddress>
    <PinPadMACAddress>54E14060E551</PinPadMACAddress>
    <PinPadIpPort>12000</PinPadIpPort> <!-- PIN Pad Port on local network -->
    <MerchantID>592885123</MerchantID>
    <LaneID>001</LaneID>
    <SequenceNo>0010010010</SequenceNo>
    <OperatorID>52</OperatorID>
    <UserTrace>Worldpay1</UserTrace>
  </Transaction>
</TStream>
```

Pay-at-Table XML Example: CancelCheckout – Response

```
<RStream>
  <CmdResponse>
    <ResponseOrigin>PATT Server</ResponseOrigin> <!-- local PATT Server -->
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Success</CmdStatus>
    <TextResponse>Cancel Attempted Of InProcess Request.</TextResponse>
    <PayAtTheTableID>20190120105414000006</PayAtTheTableID>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>Worldpay1</UserTrace>
  </CmdResponse>
</RStream>
```

TextResponse contains an optional message from the PATT Server.

Additional XML Tags and Values when Supporting Pay at theTable

XML Tag	Element Type	Description
<TranCode> Checkout </TranCode>	String 1-40 chars	TranCode used in the Pay-at-Table process to queue up a pending transaction to a supported device. Returns <i>PayAtTheTableID</i> to POS.
<TranCode> CheckStatus </TranCode>	String 1-40 chars	TranCode used to check the status of the transaction once in-flight. Response includes <i>InQueue</i> , <i>InProcess</i> or, if completed, the authorization response.
<TranCode> CancelCheckout </TranCode>	String 1-40 chars	TranCode used to cancel a transaction once "In Queue". Cannot be used to cancel an "InProcess" transaction. CheckStatus should always be called after CancelCheckout to verify the cancel.
<IpPort>	Number 1-5 char	IP port number on server to be used for PATT transactions—use value 9400 ; default is 9000
<PinPadIpAddress>	String 7-15 chars	IP address of PIN pad on local network. 12000 is common
<PinPadMACAddress>	String 12-17 chars	Media Access Control address of PIN pad; delimited by colons, hyphens or with no delimiter
<AutoPrintCustomer>	String 1-20 chars	Communicates to a PIN pad with a printer to automatically print the "Customer Copy" receipt after a completed transaction. Values are No , Yes (approvals and declines, but not errors), and ApprovalsOnly .
<AutoPrintMerchant>	String 1-20 chars	Communicates to a PIN pad with a printer to automatically print the "Merchant Copy" receipt after a completed transaction. Values are No , Yes (approvals and declines, but not errors), and ApprovalsOnly .
<Question>	String 1-20 chars	The question or message displayed at the start of a checkout. Default value is "Begin Payment?"
<HeaderLine 1-4>	String 1-40 chars	Use for receipt header to be printed at the top of the receipt. 4 lines max. Common usage is merchant name, address and phone number.
<FooterLine 1-2>	String 1-40 chars	Footer lines to be printed at the bottom of the receipt. If using one of the <i>AutoPrint</i> options, "Merchant Copy" or "Customer Copy" need not be supplied.
<PayAtTheTableID>	String 1-24 chars	Response unique identifier from Checkout and used in subsequent calls.
<TextResponse> InQueue </TextResponse>		Successful Response from a CheckStatus request indicates customer has not completed the checkout and the transaction is not being processed
<TextResponse> InProcess </TextResponse>		Successful Response from a CheckStatus request indicates customer has completed the checkout and the transaction is in the authorization process.

Payment Account Updater Management Tool

Overview – New in 2019!

This section of the guide contains information for MercuryPay/Datacap point of sale developers about Payment Account Updater, an integrated service previously only available on the Express processing platform. Because the processing routing path now flows from MercuryPay to Express, and because of the simplified manner in which Datacap Systems, Inc. has engineered the Account Updater XML requests, it is possible for MercuryPay developers to take advantages of the Express Updater features using the specifications outlined in this guide.

Account Updater Service

Account Updater Service was originally developed by Visa and MasterCard for card-on-file/card-not-present business markets and was designed to update credit card information, such as expirations dates, changes in cardholder account status and card numbers. The Express Account Updater Service, now available using Datacap technologies, enables these same Visa and MasterCard updates in an integrated payment solution. POS developers supporting a card-on-file program can now easily maintain their merchant's cardholder billing records to prevent billing to expired or invalid cards. To do this, the MercuryPay/Datacap Account Updater Services takes advantage of the Express platform's secure storage capability referred to as "PASS."

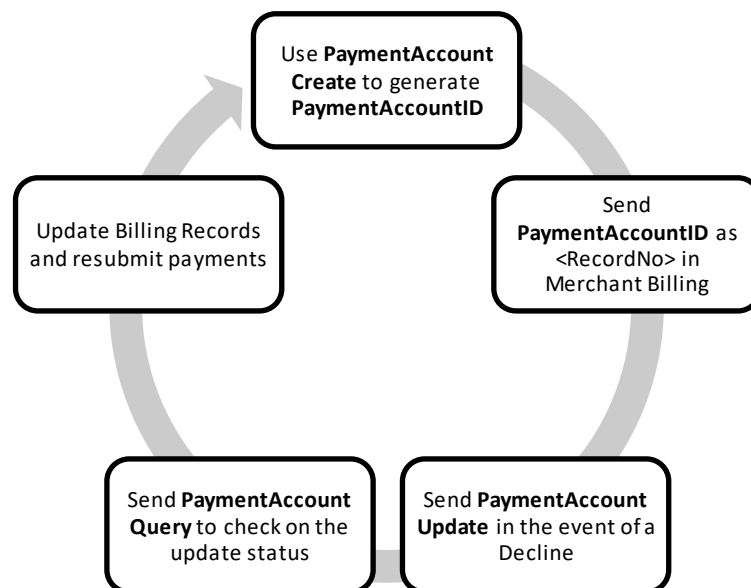
Note: Only Visa and MasterCard cards are eligible in the Account Updater service.

When implemented, the MercuryPay/Datacap Account Updater service allows enrolled merchants to create a Payment Account and, in the event of declined transaction, opt to query for applicable updates to these records. Account records are stored on Worldpay's Express PASS platform. **Payment Account Secure Storage**, PASS, uses a *PaymentAccountID/PASS record* which functions very much like a MercuryPay *MToken*. In this case though, the *PaymentAccountID* acts only as a reference to the actual cardholder account data which in turn allows this data to be updated.

Note: Enabling PASS for the merchant account, also referred to as Transform Tokenization, is a prerequisite for enabling the Account Updater Service on MercuryPay.

- Merchant accounts may be set up to automatically receive updates whereby then the POS simply sets aside declined transactions and queries the Payment Account service for the updated information
- The POS sets aside the declined accounts and manages the three phases of the process (Payment Account Create, Update and Query) programmatically

POS Application Requirements for the Account Updater Service



The Express Account Updater service is available to MercuryPay developers via a direct integration to Datacap's active-x control **DSIClientX** or, alternatively, building out a **SOAP** payload calling MercuryPay's Web Services platform. To support the Account Updater Service, developers will need to implement **PaymentAccount Create**, **PaymentAccount Update**, **PaymentAccount Query** and **PaymentAccount Delete**. Each of these will be outlined below and included request and response XML examples.

PaymentAccount Create, PaymentAccountID and ProcTransactionID

In order to implement the Express Account Update service the merchant's current card-on-file billing database will need to be upgraded to include an Express PASS record called the *PaymentAccountID*. The *PaymentAccountID/PASS* record replaces previously stored MTokens or clear text card data.

To do this, developers will first send a **PaymentAccount Create** request using either clear text card data or the *ProcTransactionID* and making a direct call to the DSIClientX control or the MercuryPay Web Service. The *ProcTransactionID* is a recent added additional value returned in MercuryPay Credit transactions in the XML *TranResponse* and captures the Express Processing TransactionID. **Example:**
<ProcTransactionID>13359804</ProcTransactionID>

The approved *PaymentAccount Create* will return a <Text Response> value of "PaymentAccount created" along with a newly generated *PaymentAccountID/PASS* record. The value is returned in the <RecordNo> tag and includes a pipe-delimited value of the *PaymentAccountID/PASS*, Expiration and CardType. The Datacap specific *PaymentAccountID/PASS* looks like this: **Example: <RecordNo>PID:878AA162-26B7-4EB5-A72C-AC7D3E004F69|1225|M/C</RecordNo>**

- This PID *RecordNo* should be stored exactly as it is returned and maintained in the merchant's online billing records. This full value will be used in place of MTokens or clear text for future payment requests. Additionally and if required, the pipe-delimited Expiration date (MMYY) and card type (VISA or M/C) may be stored separately for POS tracking purposes.
- With the *PaymentAccountID* in place, future card-on-file billing transaction requests will be sent as they always have been to the MercuryPay platform using the "ByRecordNo" TranCode. *SaleByRecordNo*, *PreAuthByRecordNo* or *ReturnByRecordNo* are supported.
- When using the PID value as the <RecordNo> for sending an Account Updater request, a <Frequency> value is no longer included in the request.

PaymentAccount Create XML Examples

The PaymentAccount Create is generated using clear text card data (Track1, Track2, and AcctNo) or the *ProcTransactionID*.

Important Note: PaymentAccount Create cannot be generated using an existing MToken. MTokens must be replaced with the PaymentAccountID/PASS record as outlined below.

Account Updater XML Request/Response Examples**Using PaymentAccount Create with ProcTransactionID**

Use the Create to generate a PaymentAccountID/PASS Record in order to run a SaleByRecordNo:

In the below examples the *ProcTransactionID* from a previously completed Credit Sale response is used to create a *PaymentAccountID/PASS* record which in turn is used to process the subsequent Credit Sale.

Note: Once a *PaymentAccountID/PASS* record is created it is used for subsequent Sale, PreAuth or Return "ByRecordNo" transactions.

- The *PaymentAccountID/PASS* does not expire or change and, unlike MToken, is not generated dynamically.
- The *ProcTransactionID* is unique per transaction and expires in 45 days.

XML Request: PaymentAccount Create (with ProcTransactionID)

```
<TStream>
  <Transaction>
```



```

    <MerchantID>62589003=TEST</MerchantID>
    <TranType>PaymentAccount</TranType> <!-- New TranType -->
    <TranCode>Create</TranCode> <!-- New TranCode -->
    <ProcTransactionID>860043302</ProcTransactionID> <!-- obtained in response from previous Credit Sale -->
    </Transaction>
</TStream>

```

XML Response: PaymentAccount Create

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>PaymentAccount created</TextResponse> <!-- Note response verbiage -->
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>62589003</MerchantID>
    <ExpDate>1220</ExpDate>
    <TranCode>Create</TranCode> <!-- TranCode is echoed in response -->
    <RecordNo>PID:DBBBBB3D-2E36-4055-9245-124A61AE2BF6|1220|M/C</RecordNo> <!-- This is the
PaymentAccountID/PASS record returned in the RecordNo tag -->
  </TranResponse>
</RStream>

```

XML Request: SaleByRecordNo (with RecordNo=PaymentAccountID/PASS Record)

```

<TStream>
  <Transaction>
    <MerchantID>62589003=TEST</MerchantID>
    <TranType>Credit</TranType>
    <TranCode>SaleByRecordNo</TranCode>
    <InvoiceNo>112</InvoiceNo>
    <RefNo>112</RefNo>
    <Memo>POS/Version</Memo>
    <Frequency>Recurring</Frequency> <!-- Always used in credit transactions -->
    <RecordNo>PID:DBBBBB3D-2E36-4055-9245-124A61AE2BF6|1220|M/C</RecordNo> <!-- In the RecordNo tag use
the complete PaymentAccountID/PASS Record -->
    <Amount>
      <Purchase>55.25</Purchase>
    </Amount>
  </Transaction>
</TStream>

```

XML Response: SaleByRecordNo (with PaymentAccountID/PASS Record)

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>62589003</MerchantID>
    <AcctNo>549999XXXXXX6781</AcctNo>
    <ExpDate>XXXX</ExpDate>
    <CardType>M/C</CardType>
    <TranCode>Sale</TranCode>
    <AuthCode>006450</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>0005</RefNo>
  </TranResponse>
</RStream>

```

```

<InvoiceNo>112</InvoiceNo>
<Memo>MPS Example XML v1.0</Memo>
<Amount>
  <Purchase>55.25</Purchase>
  <Authorize>55.25</Authorize>
</Amount>
<AcqRefData>Kb860043302#20#20#20#20e00</AcqRefData>
<RecordNo>PID:DBBBBB3D-2E36-4055-9245-124A61AE2BF6|1220|M/C</RecordNo> <!-- RecordNo returned is the
previously generated PaymentAccountID/PASS record -->
<ProcessData>|00|210100701000</ProcessData>
<ProcTransactionID>860041243</ProcTransactionID><!-- Generated unique per transaction -->
</TranResponse>
</RStream>

```

XML Request: PaymentAccount Create (with Track2 Card Data)

```

<TStream>
  <Transaction>
    <MerchantID>62589003=TEST</MerchantID>
    <TranType>PaymentAccount</TranType> <!-- New TranType -->
    <TranCode>Create</TranCode> <!-- New TranCode -->
    <InvoiceNo>121112</InvoiceNo> <!-- Invoice/RefNo optional -->
    <RefNo>121112</RefNo>
    <Account>
      <Track2>444522XXXXXX0007=25121015432198712345</Track2> <!-- Track1, Track2 or AcctNo may be used -->
    </Account>
  </Transaction>
</TStream>

```

XML Response: PaymentAccount Create

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturn Code>000000</DSIXReturn Code>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>PaymentAccount created</TextResponse> <!-- Note TextResponse -->
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>62589003</MerchantID>
    <ExpDate>1225</ExpDate>
    <TranCode>Create</TranCode> <!-- TranCode echoed in response -->
    <RecordNo>PID:67EA5A3F-F810-406A-AFEC-5D98D508F49E|1225|VISA</RecordNo> <!-- Note RecordNo returned
as the PaymentAccountID/PASS record; store this and use in subsequent requests -->
  </TranResponse>
</RStream>

```

XML Request: PaymentAccount Create (with ProcTransactionID)

```

<TStream>
  <Transaction>
    <MerchantID>62589003=TEST</MerchantID>
    <TranType>PaymentAccount</TranType> <!-- New TranType -->
    <TranCode>Create</TranCode> <!-- New TranCode -->
    <ProcTransactionID>860043302</ProcTransactionID> <!-- a new ProcTransactionID is returned in Credit
Sale transaction responses; it may be stored and used up to 45 days -->
  </Transaction>
</TStream>

```

XML Response: PaymentAccountCreate (with ProcTransactionID)

```

<RStream>

```

```

<CmdResponse>
  <ResponseOrigin>Processor</ResponseOrigin>
  <DSIXReturn Code>000000</DSIXReturn Code>
  <CmdStatus>Approved</CmdStatus>
  <TextResponse>PaymentAccount created</TextResponse> <!-- Note TextResponse -->
  <UserTraceData></UserTraceData>
</CmdResponse>
<TranResponse>
  <MerchantID>62589003</MerchantID>
  <ExpDate>1220</ExpDate>
  <TranCode>Create</TranCode> <!-- TranCode echoed in response -->
  <RecordNo>PID:DBBBBB3D-2E36-4055-9245-124A61AE2BF6|1220|M/C</RecordNo> <!-- Note RecordNo returned
as the PaymentAccountID; store this and use in subsequent requests -->
</TranResponse>
</RStream>

```

PaymentAccount Update

There are two cases to implement **PaymentAccount Update**.

1. The **first use case** is in the event that a transaction declines in the submitted card-on-file batch.

- Flag these declined transactions and send to the update service using *PaymentAccount Update* and handle when a successful update is supplied (typically 4-7 business days).
- The reason for these decline may be clear in the transaction response message (for example an invalid expiration date) or, with more generic declines, the reason may be less obvious as it could be related to the cardholder's available "open-to-buy" balance or a change in their account status (for example a standard 05 Decline).
- By sending a *PaymentAccount Update*, the Express Account Updater service is alerted to check for available account updates. To complete the process the *PaymentAccount Query* (discussed below) is used to poll the status.
- The following decline values may be used to identify and flag for *PaymentAccount Update*.

Type	Description
General Decline	DECLINE—Card issuer has declined the transaction. Do not honor, Insufficient Funds, Exceeds Max Amount
Call Referral Decline	CALL ND, CALL AMEX, CALL DISCOVER—Call referral contact card brand
Invalid Decline	INVLD CARD—Invalid Card
Expired Decline	INVLD EXP DATE—Card is expired. Invalid Card, Expired
Pick Up Decline	PIC UP—Card issuer has declined the transaction and wants to recover the card.

2. The **second use case** for implementing *PaymentAccount Update* is in the event that a cardholder proactively notifies the merchant that their current card-on-file has changed or needs to be updated.

- In this case, *PaymentAccount Update* requests can be sent including the updated cardholder data (Account and Expiration Date). The Express Account Updater service will then pass this information on to the card networks for confirmation and update the record. To complete the process, the *PaymentAccount Query* (discussed below) is used to poll the status.

Note: If a PaymentAccount /PASS record is updated while in an "In Progress" status, that second update may override the original update. It is recommended that merchants not be allowed to modify the card number or expiration date on file until the Update process is completed.

Either use case of *PaymentAccount Update* may be submitted during the business day, but only at the time of the merchant's daily *BatchClose* will Update requests be submitted and forwarded to the Express Account Updater service.

- Update requests on declined transactions are processed and available typically within four to five business days. In the case of merchant-supplied card data updates, this can take up to seven business days.
- Once the update process is complete, the *UpdateStatus* of the *PaymentAccount/PASS* record will remain in this status until a subsequent *PaymentAccount Update* is requested indicating a change in the account.

PaymentAccount Update XML Examples

XML Request: PaymentAccount Update (with PaymentAccountID/PASS Record)

Use this Update to alert the Express Account Updater service to check for updates when a Decline response is received.

```
<TStream>
  <Transaction>
    <MerchantID>62589003=TEST</MerchantID>
    <TranType>PaymentAccount</TranType> <!-- New TranType -->
    <TranCode>Update</TranCode> <!-- New TranCode -->
    <InvoiceNo>12111</InvoiceNo> <!-- Invoice/RefNo are optional -->
    <RecordNo>PID:878AA162-26B7-4EB5-A72C-AC7D3E004F69|1225|M/C</RecordNo> <!-- Use the
PaymentAccountID/PASS Record to reference the account-->
  </Transaction>
</TStream>
```

XML Response: PaymentAccount Update

```
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus> <!-- Indicates the request was successful -->
    <TextResponse>PaymentAccount updated</TextResponse> <!-- indicates the Account has been flagged for any
available updates in the Express Account Updater service. -->
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>62589003</MerchantID>
    <ExpDate>0000</ExpDate>
    <TranCode>Update</TranCode>
    <RecordNo>PID:878AA162-26B7-4EB5-A72C-AC7D3E004F69|0000|None</RecordNo> <!-- Note that the Exp Date
and Card Type are not returned in the response-->
  </TranResponse>
</RStream>
```

XML Request: PaymentAccount Update (with Account Data and Expiration Date)

Use this PaymentAccount Update to change an existing record if the updated information is supplied (In this example the Visa card is being updated to a different card, yet note, on the request, the numeric *PaymentAccountID/PASS* record is maintained).

```
<TStream>
  <Transaction>
    <MerchantID>62589003=TEST</MerchantID>
    <TranType>PaymentAccount</TranType> <!-- New TranType -->
    <TranCode>Update</TranCode> <!-- New TranCode -->
    <InvoiceNo>122</InvoiceNo>
    <RefNo>122</RefNo>
    <Account>
```

```

    <AcctNo>549999XXXXXX6781</AcctNo> <!-- In this case the data supplied is being requested to update the
account record -->
    <ExpDate>1220</ExpDate>
  </Account>
  <RecordNo>PID:568830C2-DCF6-4384-9FFD-FAC640E78860|1220|VISA</RecordNo> <!-- This request will
update this existing record The Exp and CardType will change-->
</Transaction>
</TStream>

```

XML Response: PaymentAccount Update

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>PaymentAccount updated</TextResponse> <!-- Indicates the update request has been submitted -->
  </CmdResponse>
  <TranResponse>
    <UserTraceData></UserTraceData>
    <MerchantID>62589003</MerchantID>
    <ExpDate>1220</ExpDate>
    <TranCode>Update</TranCode> <!-- TranCode echoed in response -->
    <RecordNo>PID:568830C2-DCF6-4384-9FFD-FAC640E78860|1220|M/C</RecordNo> <!-- Note the update change
request is indicated in the RecordNo -->
  </TranResponse>
</RStream>

```

PaymentAccount Query

To check the status of an update request, the payment application will send a **PaymentAccount Query**. Again the *PaymentAccountID*, saved and stored on the billing record is referenced in the *PaymentAccount Query* request.

- *PaymentAccount Query* may be implemented by polling the list of previously flagged *PaymentAccountIDs* sent through in the *PaymentAccount Update*. It is recommended to keep querying until receiving a status update.
- Details of the successful response will be indicated in the <UpdateStatus> tag.
- The <UpdateStatus> will include indications that the account is in process of being updated, has been updated and the type of update. The status indicates that the account is available for resubmission for payment or indicates that the account has not been updated (or will never be updated) and other action should be taken with the cardholder.
- If your payment application maintains non-sensitive, truncated card information, then that information may also be updated in the merchant's records to reflect the updated *PaymentAccountID*.
- The following are the valid response match type values resulting from a *PaymentAccount Query* request:

Response Match Type	Description
UpdateInProgress	Update in progress. Indicates the request has been submitted and pending. Continue to poll/query for status.
MatchNoChanges	Account match was found. There are no changes.
MatchAccountChange	Account match was found and account number has changed. The PaymentAccountID/PASS record has been updated with the new credit card number.
MatchExpirationChange	Account match was found and expiration date has changed. The PaymentAccountID/PASS record has been updated with the new expiration date.

MatchAccountClosed	Account match was found. Account has been closed. The PaymentAccountID/PASS record will not be updated. Use this value to identify and remove this PaymentAccountID from future billing. Contact the cardholder for updated payment information.
MatchContactCardholder	Account match was found. The PaymentAccountID/PASS record will not be updated. The merchant will need to contact the cardholder to inquire about the status of their account.
NoMatchParticipating	Account match not found on participating BIN
NoMatchNonParticipating	Account match not found on non-participating BIN
InvalidInfo	Invalid information. Invalid card number—check to confirm card data
NoResponse	No response. Request submitted, but no response
NotAllowed	Not allowed. Merchant not set up for Account Updater service
Error	Error
PASSUpdaterDisabled	The PaymentAccountID/PASS Updater is disabled
NotUpdated	Not Updated

PaymentAccount Query XML Examples

XML Request: PaymentAccount Query (with PaymentAccountID/PASS Record)

```

<TStream>
  <Transaction>
    <MerchantID>62589003=TEST</MerchantID>
    <TranType>PaymentAccount</TranType> <!-- New TranType -->
    <TranCode>Query</TranCode> <!-- New TranCode -->
    <InvoiceNo>1</InvoiceNo>
    <RefNo>1</RefNo>
    <RecordNo>PID:568830C2-DCF6-4384-9FFD-FAC640E78860|1220|M/C</RecordNo> <!-- Always use the
PaymentAccountID to Query -->
  </Transaction>
</TStream>

```

XML Response: PaymentAccount Query

In the MercuryPay/Express Certification environment, the following successful response with UpdateStatus "Not Updated" is the expected and only Query response available. Please note and code for the above match types as valid values for <Update Status>.

```

<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Success</TextResponse> <!-- Indicates the Query has been submitted with the status of the query
available in the StatusUpdate tag -->
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>62589003</MerchantID>
    <AcctNo>xxxxxxxxxxxx6781</AcctNo>
    <ExpDate>1220</ExpDate>
    <UpdateStatus>NotUpdated</UpdateStatus> <!-- Note other valid status options include UpdateInProgress,
MatchNoChanges, MatchAccountChange, MatchExpirationChange, MatchAccountClosed, MatchContactCardholder,
NoMatchParticipating, NoMatchNonParticipating, InvalidInfo, NoResponse, NotAllowed, Error, PASSUpdaterDisabled, as
well as NotUpdated -->
    <TranCode>Query</TranCode> <!-- TranCode echoed in response -->
    <RecordNo>PID:568830C2-DCF6-4384-9FFD-FAC640E78860|1220|M/C</RecordNo>
  </TranResponse>
</RStream>

```

PaymentAccount Delete

Use the **PaymentAccount Delete** to delete an existing PaymentAccount/PASS record in the Express Account Updater service. *PaymentAccount Delete* should be used with existing records and only after receiving a successful <UpdateStatus>. Do not use *PaymentAccount Delete* while an update request is "In Progress."

- If a PaymentAccount/PASS record is deleted using *PaymentAccount Delete* while in an "In Progress" status, that record's status detail will neither be captured in the Express Account Updater service nor returned to the payment application.
- If a merchant needs to resubmit an update (although not recommended) while it is "In Process" with a different card number or expiration date they should send a *PaymentAccount Delete* to remove the existing request and resubmit the corrected cardholder data using *PaymentAccount Create*.

PaymentAccount Delete XML Examples

XML Request: PaymentAccount Delete (with PaymentAccountID/PASS Record)

```
<TStream>
  <Transaction>
    <MerchantID>62589003=TEST</MerchantID>
    <TranType>PaymentAccount</TranType> <!-- New TranType -->
    <TranCode>Delete</TranCode> <!-- New TranCode -->
    <InvoiceNo>12111</InvoiceNo>
    <RefNo>12111</RefNo>
    <RecordNo>PID:7D60BEF7-C68A-4C50-8F12-3CFBC6401744|1225|M/C</RecordNo> <!-- This is the record to be
deleted -->
  </Transaction>
</TStream>
```

XML Request: PaymentAccount Delete

```
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>PaymentAccount deleted</TextResponse> <!-- Indicates the PaymentAccount Record
has been deleted and is no longer available for an update or query -->
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>62589003</MerchantID>
    <TranCode>Delete</TranCode> <!-- TranCode echoed in response -->
  </TranResponse>
</RStream>
```

Note: in the event that an Update or Query request is mistakenly sent on a PaymentAccountID/PASS record that has been deleted, the request will decline with a TextResponse message of "PaymentAccount record(s) not found."

```
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Declined</CmdStatus>
    <TextResponse>PaymentAccount record(s) not found</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>62589003</MerchantID>
    <ExpDate>0000</ExpDate>
    <TranCode>Update</TranCode>
```

```
<RecordNo>PID:12AE1CF2-CD2C-40A4-A661-340993718D7B|0000|None</RecordNo>
</TranResponse>
</RStream>
```

Standard Datacap EMV XML Request/Response Examples

EMV Credit

Note: XML examples are also available via Datacap's developer portal at <https://www.dsidevportal.com/>.

EMV Credit Transaction Example XML

The following transactions will be outlined:

EMVParamDownload	AdjustByRecordNo	EMVVoidReturn	BatchSummary
EMVSale	EMVVoidSale (Reversal)	VoidReturnByRecordNo	BatchClose
EMVSale with Gratuity (Tip)	VoidSaleByRecordNo	EMVVoiceAuth	
SaleByRecordNo	EMVReturn	EMVZeroAuth	

EMV ParamDownload

This is an administrative function that should be performed when first installing a device or when a device is replaced.

Request

```
<?xml version="1.0"?>
  <TStream>
    <Admin>
      <HostOrIP>127.0.0.1</HostOrIP>
      <IpPort>9000</IpPort>
      <MerchantID>755847002</MerchantID>
      <OperatorID>User</OperatorID>
      <UserTrace>DevInt</UserTrace>
      <TranCode>EMVParamDownload</TranCode>
      <SecureDevice>EMV_VX805_MERCURY</SecureDevice> <!-- use the correct SecureDeviceID -->
      <ComPort>9</ComPort>
      <SequenceNo>0010010010</SequenceNo>
    </Admin>
  </TStream>
```

Response

```
<?xml version="1.0"?>
  <RStream>
    <CmdResponse>
      <ResponseOrigin>127.0.0.1</ResponseOrigin>
      <DSIXReturnCode>000000</ DSIXReturnCode >
      <CmdStatus>Success</ CmdStatus>
      <TextResponse>SUCCESS</TextResponse>
      <SequenceNo>0010010010</SequenceNo>
      <UserTrace>DevInt</UserTrace>
    </CmdResponse>
    <TranResponse>
      <TranCode>EMVParamDownload</TranCode>
    </TranResponse>
  </RStream>
```

EMVSale

This example includes requesting a token.

Request

```
<?xml version="1.0"?>
<TStream>
```

```

<Transaction>
  <HostOrIP>127.0.0.1</HostOrIP><!--use to communicate with the NETePay when locally installed;
may vary if NETePay is networked on another computer-->
  <IpPort>9000</IpPort>
    <MerchantID>755847002</MerchantID>
    <OperatorID>User</OperatorID>
    <UserTrace>DEMO</UserTrace>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <LaneID>02</LaneID> <!--for environments with multiple input devices-->
    <TranCode>EMVSale</TranCode>
    <SecureDevice>EMV_VX805_MERCURY</SecureDevice>
    <ComPort>9</ComPort>
    <InvoiceNo>2001</InvoiceNo>
    <RefNo>2001</RefNo> <!--InvoiceNo=RefNo in the request, filler value-->
    <Amount>
      <Purchase>2.15</Purchase>
    </Amount>
    <SequenceNo>0010010010</SequenceNo>
    <PartialAuth>Allow</PartialAuth>
    <RecordNo>RecordNumberRequested</RecordNo> <!--token request-->
    <Frequency>OneTime</Frequency>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>DEMO</UserTrace>
  </CmdResponse>
  <TranResponse>
    <MerchantID>755847002</MerchantID>
    <AcctNo>*****0010</AcctNo>
    <CardType>VISA</CardType>
    <TranCode>EMVSale</TranCode>
    <AuthCode>30713A</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>1003</RefNo> <!--note actual RefNo-->
    <InvoiceNo>2001</InvoiceNo>
    <OperatorID>User</OperatorID>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <Amount>
      <Purchase>2.15</Purchase>
      <Authorize>2.15</Authorize>
    </Amount>
    <AcqRefData>b8238953#20#20#20#20#20#20#20e00</AcqRefData> <!--used for VoidSales-->
    <ProcessData>|00|410100501000</ProcessData> <!--used for VoidSales-->
    <RecordNo>SEXFNopTA/Di8VmT75Vgg32WTEtGB/SI5vd1Xl04nliEgUQESIQCAC+Y</RecordNo> <!--
token is returned in response-->
    <EntryMethod>CHIP</EntryMethod>
    <Date>07/31/2015</Date>
    <Time>10:07:22</Time>
    <ApplicationLabel>Visa Credit</ApplicationLabel>
    <AID>A0000000031010</AID>
    <TVR>0000008000</TVR>
    <IAD>06010A03600000</IAD>
    <TSI>F800</TSI>
    <ARC>00</ARC>
  </TranResponse>
</RStream>

```

```

        <CVM>SIGN</CVM>
    </TranResponse>
    <PrintData>
        <Line1>.MERCHANT ID: 755847004</Line1>
        <Line2>.CLERK ID: test</Line2>
        <Line3>.</Line3>
        <Line4>.    SALE    </Line4>
        <Line5>.</Line5>
        <Line6>.*****0010</Line6>
        <Line7>.VISA    ENTRY METHOD: CHIP</Line7>
        <Line8>.DATE: 07/31/2015 TIME: 10:07:22</Line8>
        <Line9>.</Line9>
        <Line10>.INVOICE: 2001</Line10>
        <Line11>.REFERENCE: 1003</Line11>
        <Line12>.AUTH CODE: 30713A</Line12>
        <Line13>.</Line13>
        <Line14>.AMOUNT    USD$ 1.00</Line14>
        <Line15>.&    =====</Line15>
        <Line16>.TOTAL    USD$ 1.00</Line16>
        <Line17>.</Line17>
        <Line18>.&    APPROVED - THANK YOU    </Line18>
        <Line19>.</Line19>
        <Line20>.I AGREE TO PAY THE ABOVE TOTAL AMOUNT</Line20>
        <Line21>.ACCORDING TO CARD ISSUER AGREEMENT</Line21>
        <Line22>.(MERCHANT AGREEMENT IF CREDIT VOUCHER)</Line22>
        <Line23>.</Line23>
        <Line24>.</Line24>
        <Line25>.</Line25>
        <Line26>.x_____</Line26>
        <Line27>.&    Cardholder Signature    </Line27>
        <Line28>.</Line28>
        <Line29>.</Line29>
        <Line30>.Application Label: Visa Credit</Line30>
        <Line31>.AID: A0000000031010</Line31>
        <Line32>.TVR: 0000008000</Line32>
        <Line33>.IAD: 06010A03600000</Line33>
        <Line34>.TSI: F800</Line34>
        <Line35>.ARC: 00</Line35>
        <Line36>.CVM: SIGN</Line36>
    </PrintData>
</RStream>

```

EMVSale with Gratuity (Tip)

Request

```

<?xml version="1.0"?>
<TStream>
    <Transaction>
        <HostOrIP>127.0.0.1</HostOrIP>
        <IpPort>9000</IpPort>
        <MerchantID>755847002</MerchantID>
        <OperatorID>User</OperatorID>
        <UserTrace>DEMO</UserTrace>
        <Memo>Demo POS 1.5 USEMV</Memo>
        <TranCode>EMVSale</TranCode>
        <SecureDevice>EMV_VX805_MERCURY</SecureDevice>
        <ComPort>9</ComPort>
        <InvoiceNo>2005</InvoiceNo>
        <RefNo>2005</RefNo>
        <Amount>
            <Purchase>2.10</Purchase>
            <Gratuity>Prompt</Gratuity><!--device will prompt cardholder to enter a gratuity-->

```



```

    </Amount>
    <SequenceNo>0010010010</SequenceNo>
    <PartialAuth>Allow</PartialAuth>
    <RecordNo>RecordNumberRequested</RecordNo>
    <Frequency>OneTime</Frequency>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>DEMO</UserTrace>
  </CmdResponse>
  <TranResponse>
    <MerchantID>755847002</MerchantID>
    <AcctNo>*****0010</AcctNo>
    <CardType>VISA</CardType>
    <TranCode>EMVSale</TranCode>
    <AuthCode>31013A</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>1006</RefNo>
    <InvoiceNo>2005</InvoiceNo>
    <OperatorID>User</OperatorID>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <Amount>
      <Purchase>2.10</Purchase>
      <Gratuity>0.40</Gratuity>      <!--Gratuity amount entered at prompt-->
      <Authorize>2.50</Authorize>  <!--Authorize amount is automatically calculated -->
    </Amount>
    <AcqRefData>b8238953#20#20#20#20#20#20e00</AcqRefData>
    <ProcessData>|00|410100501000</ProcessData>
  <RecordNo>dGMuw9MB2vk5KpFzCrKMvtvZ9ZBusLV/vzfqpw9ygiEgUQESIQAIBY</RecordNo>
  <EntryMethod>CHIP</EntryMethod>
  <Date>08/03/2015</Date>
  <Time>11:27:01</Time>
  <ApplicationLabel>Visa Credit</ApplicationLabel>
  <AID>A0000000031010</AID>
  <TVR>0000008000</TVR>
  <IAD>06010A03600000</IAD>
  <TSI>F800</TSI>
  <ARC>00</ARC>
  <CVM>SIGN</CVM>
</TranResponse>
<PrintData>
  <Line1>.MERCHANT ID: 755847004</Line1>
  <Line2>.CLERK ID: test</Line2>
  <Line3>.</Line3>
  <Line4>.SALE      </Line4>
  <Line5>.</Line5>
  <Line6>*****0010</Line6>
  <Line7>.VISA      ENTRY METHOD: CHIP</Line7>
  <Line8>.DATE: 08/03/2015 TIME: 11:27:01</Line8>
  <Line9>.</Line9>
  <Line10>.INVOICE: 2005</Line10>
  <Line11>.REFERENCE: 1006</Line11>
  <Line12>.AUTH CODE: 31013A</Line12>
  <Line13>.</Line13>

```

```

<Line14>.AMOUNT      USD$ 2.10</Line14>
<Line15>.TIP         USD$ 0.40</Line15>
<Line16>.           =====</Line16>
<Line17>.TOTAL       USD$ 2.50</Line17>
<Line18>.</Line18>
<Line19>.  APPROVED - THANK YOU  </Line19>
<Line20>.</Line20>
<Line21>.I AGREE TO PAY THE ABOVE TOTAL AMOUNT</Line21>
<Line22>.ACCORDING TO CARD ISSUER AGREEMENT</Line22>
<Line23>.(MERCHANT AGREEMENT IF CREDIT VOUCHER)</Line23>
<Line24>.</Line24>
<Line25>.</Line25>
<Line26>.</Line26>
<Line27>.x_____</Line27>
<Line28>.  Cardholder Signature  </Line28>
<Line29>.</Line29>
<Line30>.</Line30>
<Line31>.Application Label: Visa Credit</Line31>
<Line32>.AID: A0000000031010</Line32>
<Line33>.TVR: 0000008000</Line33>
<Line34>.IAD: 06010A03600000</Line34>
<Line35>.TSI: F800</Line35>
<Line36>.ARC: 00</Line36>
<Line37>.CVM: SIGN</Line37>

```

```
</PrintData>
```

```
</RStream>
```

SaleByRecordNo

This transaction uses a token and is card-not-present.

Request

```
<?xml version="1.0"?>
```

```
<TStream>
```

```
<Transaction>
```

```
<HostOrIP>127.0.0.1</HostOrIP>
```

```
<IpPort>9000</IpPort>
```

```
<MerchantID>166758005</MerchantID>
```

```
<OperatorID>User</OperatorID>
```

```
<UserTrace>dmeeker</UserTrace>
```

```
<TranType>Credit</TranType>
```

```
<TranCode>SaleByRecordNo</TranCode>
```

```
<SecureDevice>EMV_VX805_MERCURY</SecureDevice>
```

```
<ComPort>9</ComPort>
```

```
<InvoiceNo>10005</InvoiceNo>
```

```
<RefNo>10005</RefNo>
```

```
<Amount>
```

```
<Purchase>1.50</Purchase>
```

```
</Amount>
```

```
<SequenceNo>0010010010</SequenceNo>
```

```
<RecordNo>LHBuaFAfRtpZzv6lpTvAJildekD8FsWo20vbr4p1pzYiEgUQEylQC4tY</RecordNo>
```

```
<Frequency>OneTime</Frequency> <!--note use of RecordNo AND Frequency required-->
```

```
</Transaction>
```

```
</TStream>
```

Response

```
<?xml version="1.0"?>
```

```
<RStream>
```

```
<CmdResponse>
```

```
<ResponseOrigin>Processor</ResponseOrigin>
```

```
<DSIXReturnCode>000000</DSIXReturnCode>
```

```
<CmdStatus>Approved</CmdStatus>
```

```
<TextResponse>AP</TextResponse>
```

```
<SequenceNo>0010010010</SequenceNo>
```

```

        <UserTrace>dmeeker</UserTrace>
    </CmdResponse>
    <TranResponse>
        <MerchantID>166758005</MerchantID>
        <AcctNo>476173XXXXXX0010</AcctNo>
        <ExpDate>XXXX</ExpDate>
        <CardType>VISA</CardType>
        <TranCode>Sale</TranCode>
        <AuthCode>48882A</AuthCode>
        <CaptureStatus>Captured</CaptureStatus>
        <RefNo>1102</RefNo>
        <InvoiceNo>10005</InvoiceNo>
        <OperatorID>User</OperatorID>
        <Amount>
            <Purchase>1.50</Purchase>
            <Authorize>1.50</Authorize>
        </Amount>
        <AcqRefData>Kb8238953#20#20#20#20#20#20#20e00</AcqRefData>

        <RecordNo>2nOKR92NqT+80z8wuj6lw6uVKZ+urU57g SeRGJgTbJEiEgUQEylQDTZi</RecordNo>
<!--note dynamic generated token is new-->
        <ProcessData>|17|510100700000</ProcessData>
    </TranResponse>
</RStream>

```

AdjustByRecordNo

This transaction uses a token and is card-not-present.

Request

```

<?xml version="1.0"?>
<TStream>
    <Transaction>
        <HostOrIP>127.0.0.1</HostOrIP>
        <IpPort>9000</IpPort>
        <MerchantID>755847002</MerchantID>
        <OperatorID>User</OperatorID>
        <UserTrace>DEMO</UserTrace>
        <Memo>Demo POS 1.5 USEMV</Memo>
        <TranType>Credit</TranType>
        <TranCode>AdjustByRecordNo</TranCode>
        <SecureDevice>EMV_VX805_MERCURY</SecureDevice>
        <ComPort>9</ComPort>
        <InvoiceNo>1008</InvoiceNo>
        <RefNo>1008</RefNo>
        <AuthCode>30931A</AuthCode>
        <Amount>
            <Purchase>2.50</Purchase>
        </Amount>
        <SequenceNo>0010010010</SequenceNo>
        <RecordNo>dt76RZM5f2t/eOW6k2+yfgf7hTcv0BH/MVuzN05/qakiEgUQESIQADDa</RecordNo>
        <Frequency>OneTime</Frequency> <!--note use of RecordNo AND Frequency required-->
        <AcqRefData>b8238953#20#20#20#20#20#20#20e00</AcqRefData>
        <ProcessData>|00|410100501000</ProcessData>
    </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
    <CmdResponse>
        <ResponseOrigin>Processor</ResponseOrigin>
        <DSIXReturnCode>000000</DSIXReturnCode>
    </CmdResponse>
</RStream>

```

```

    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>DEMO</UserTrace>
  </CmdResponse>
  <TranResponse>
    <MerchantID>755847002</MerchantID>
    <AcctNo>476173XXXXXX0010</AcctNo>
    <ExpDate>XXXX</ExpDate>
    <CardType>VISA</CardType>
    <TranCode>Adjust</TranCode>
    <AuthCode>30931A</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>1008</RefNo>
    <InvoiceNo>1008</InvoiceNo>
    <OperatorID>User</OperatorID>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <Amount>
      <Purchase>2.50</Purchase>
      <Authorize>2.50</Authorize>
    </Amount>
    <RecordNo>GcCB9KyTbLz7hLwxXZ15QtXZtTOxbSx+BzbtDZZkNiMiEgUQESIQADDf</RecordNo> <!--
note new-->
    <ProcessData>|02|410100600000</ProcessData>
  </TranResponse>
</RStream>

```

EMVVoidSale (Reversal)

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <HostOrIP>127.0.0.1</HostOrIP>
    <IpPort>9000</IpPort>
    <MerchantID>755847002</MerchantID>
    <OperatorID>User</OperatorID>
    <UserTrace>DEMO</UserTrace>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <TranCode>EMVVoidSale</TranCode>
    <SecureDevice>EMV_VX805_MERCURY</SecureDevice>
    <ComPort>9</ComPort>
    <InvoiceNo>1003</InvoiceNo>
    <RefNo>1003</RefNo>
    <AuthCode>30713A</AuthCode>
    <Amount>
      <Purchase>1.70</Purchase>
    </Amount>
    <SequenceNo>0010010010</SequenceNo>
    <RecordNo>RecordNumberRequested</RecordNo>
    <Frequency>OneTime</Frequency>
    <AcqRefData>b8238953#20#20#20#20#20#20e00</AcqRefData>
    <ProcessData>|00|410100501000</ProcessData> <!-- request includes both AcqRefData and
ProcessData-->
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>

```

```

    <CmdStatus>Approved</CmdStatus>
    <TextResponse>REVERSED</TextResponse>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>DEMO</UserTrace>
  </CmdResponse>
  <TranResponse>
    <MerchantID>755847002</MerchantID>
    <AcctNo>*****0010</AcctNo>
    <CardType>VISA</CardType>
    <TranCode>EMVVoidSale</TranCode>
    <AuthCode>31006A</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>2055</RefNo>
    <InvoiceNo>1003</InvoiceNo>
    <OperatorID>User</OperatorID>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <Amount>
      <Purchase>1.70</Purchase>
      <Authorize>1.70</Authorize>
    </Amount>
    <ProcessData>|A4|410100500000</ProcessData>
  <RecordNo>WnRuOfs49wVjnoYITx9E7enGnbRI0BK0Rd0RUN5JUgYiEgUQESIQAC+h</RecordNo>
    <EntryMethod>CHIP</EntryMethod>
    <Date>07/31/2015</Date>
    <Time>10:09:14</Time>
    <ApplicationLabel>Visa Credit</ApplicationLabel>
    <AID>A0000000031010</AID>
    <TVR>0000008000</TVR>
    <IAD>06010A03800000</IAD>
    <TSI>A800</TSI>
    <CVM>SIGN</CVM>
  </TranResponse>
  <PrintData>
    <Line1>.MERCHANT ID: 755847004</Line1>
    <Line2>.CLERK ID: test</Line2>
    <Line3>.</Line3>
    <Line4>. VOID SALE </Line4>
    <Line5>.</Line5>
    <Line6>*****0010</Line6>
    <Line7>.VISA ENTRY METHOD: CHIP</Line7>
    <Line8>.DATE: 07/31/2015 TIME: 10:09:14</Line8>
    <Line9>.</Line9>
    <Line10>.INVOICE: 1003</Line10>
    <Line11>.REFERENCE: 1003</Line11>
    <Line12>.AUTH CODE: 31006A</Line12>
    <Line13>.</Line13>
    <Line14>.AMOUNT USD$ 1.70</Line14>
    <Line15>.</Line15>
    <Line16>.TOTAL USD$ 1.70</Line16>
    <Line17>.</Line17>
    <Line18>. APPROVED - THANK YOU </Line18>
    <Line19>.</Line19>
    <Line20>.I AGREE TO PAY THE ABOVE TOTAL AMOUNT</Line20>
    <Line21>.ACCORDING TO CARD ISSUER AGREEMENT</Line21>
    <Line22>.(MERCHANT AGREEMENT IF CREDIT VOUCHER)</Line22>
    <Line23>.</Line23>
    <Line24>.</Line24>
    <Line25>.</Line25>
    <Line26>.</Line26>
    <Line27>. Merchant Signature </Line27>
    <Line28>.</Line28>
    <Line29>.</Line29>
  </PrintData>

```



```

    <Line30>.Application Label: Visa Credit</Line30>
    <Line31>.AID: A0000000031010</Line31>
    <Line32>.TVR: 0000008000</Line32>
    <Line33>.IAD: 06010A03800000</Line33>
    <Line34>.TSI: A800</Line34>
    <Line35>.CVM: SIGN</Line35>
  </PrintData>
</RStream>

```

VoidSaleByRecordNo -- This transaction uses a token and is card-not-present.

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <HostOrIP>127.0.0.1</HostOrIP>
    <IpPort>9000</IpPort>
    <MerchantID>755847002</MerchantID>
    <OperatorID>User</OperatorID>
    <UserTrace>DEMO</UserTrace>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <TranType>Credit</TranType>
    <TranCode>VoidSaleByRecordNo</TranCode>
    <SecureDevice>EMV_VX805_MERCURY</SecureDevice>
    <ComPort>9</ComPort>
    <InvoiceNo>1004</InvoiceNo>
    <RefNo>1004</RefNo>
    <AuthCode>30726A</AuthCode>
    <Amount>
      <Purchase>1.50</Purchase>
    </Amount>
    <SequenceNo>0010010010</SequenceNo>
    <RecordNo>9ap+m72prd5m0DUWjsg XES+B9DFphjs97xB63Eiq/iEiEgUQESIQAC+o</RecordNo>
    <Frequency>OneTime</Frequency> <!--Note both RecordNo and Frequency required-->
    <AcqRefData>b8238953#20#20#20#20#20#20e00</AcqRefData>
    <ProcessData>|00|410100501000</ProcessData>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>REVERSED</TextResponse>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>DEMO</UserTrace>
  </CmdResponse>
  <TranResponse>
    <MerchantID>755847002</MerchantID>
    <AcctNo>476173XXXXXX0010</AcctNo>
    <ExpDate>XXXX</ExpDate>
    <CardType>VISA</CardType>
    <TranCode>Void Sale</TranCode>
    <AuthCode>30726A</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>1004</RefNo>
    <InvoiceNo>1004</InvoiceNo>
    <OperatorID>User</OperatorID>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <Amount>

```

```

        <Purchase>1.50</Purchase>
        <Authorize>1.50</Authorize>
    </Amount>
    <AcqRefData>K</AcqRefData>
<RecordNo>UOdEQVRjCYstot6lYyD60+mOSN+AAJik2jh6xR8VNliEgUQESIQAC+r</RecordNo>
    <ProcessData>|3|410100600000</ProcessData>
</TranResponse>
</RStream>

```

EMVReturn

Request

```

<?xml version="1.0"?>
<TStream>
    <Transaction>
        <HostOrIP>127.0.0.1</HostOrIP>
        <IpPort>9000</IpPort>
        <MerchantID>755847002</MerchantID>
        <OperatorID>User</OperatorID>
        <UserTrace>DEMO</UserTrace>
        <Memo>Demo POS 1.5 USEMV</Memo>
        <TranCode>EMVReturn</TranCode>
        <SecureDevice>EMV_VX805_MERCURY</SecureDevice>
        <ComPort>9</ComPort>
        <InvoiceNo>2003</InvoiceNo>
        <RefNo>2003</RefNo>
        <Amount>
            <Purchase>3.04</Purchase>
        </Amount>
        <SequenceNo>0010010010</SequenceNo>
        <PartialAuth>Allow</PartialAuth>
        <RecordNo>RecordNumberRequested</RecordNo>
        <Frequency>OneTime</Frequency>
    </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
    <CmdResponse>
        <ResponseOrigin>Processor</ResponseOrigin>
        <DSIXReturnCode>000000</DSIXReturnCode>
        <CmdStatus>Approved</CmdStatus>
        <TextResponse>AP</TextResponse>
        <SequenceNo>0010010010</SequenceNo>
        <UserTrace>DEMO</UserTrace>
    </CmdResponse>
    <TranResponse>
        <MerchantID>755847002</MerchantID>
        <AcctNo>*****0010</AcctNo>
        <CardType>VISA</CardType>
        <TranCode>EMVReturn</TranCode>
        <AuthCode>726529</AuthCode>
        <CaptureStatus>Captured</CaptureStatus>
        <RefNo>1005</RefNo>
        <InvoiceNo>2003</InvoiceNo>
        <OperatorID>User</OperatorID>
        <Memo>Demo POS 1.5 USEMV</Memo>
        <Amount>
            <Purchase>3.04</Purchase>
            <Authorize>3.04</Authorize>
        </Amount>
        <ProcessData>|20|410100501000</ProcessData>
    </TranResponse>
</RStream>

```

```

<RecordNo>UbrPuSVf8fBPvgmSP/N6LafZ4BVspalgrgrlrH2u0siEgUQESIQC+y</RecordNo>
  <EntryMethod>CHIP</EntryMethod>
  <Date>07/31/2015</Date>
  <Time>10:12:22</Time>
  <ApplicationLabel>Visa Credit</ApplicationLabel>
  <AID>A0000000031010</AID>
  <TVR>0000008000</TVR>
  <IAD>06010A03800000</IAD>
  <TSI>A800</TSI>
  <CVM>SIGN</CVM>
</TranResponse>
<PrintData>
  <Line1>.MERCHANT ID: 755847004</Line1>
  <Line2>.CLERK ID: test</Line2>
  <Line3>.</Line3>
  <Line4>.    REFUND    </Line4>
  <Line5>.</Line5>
  <Line6>.*0010</Line6>
  <Line7>.VISA    ENTRY METHOD: CHIP</Line7>
  <Line8>.DATE: 07/31/2015 TIME: 10:12:22</Line8>
  <Line9>.</Line9>
  <Line10>.INVOICE: 2003</Line10>
  <Line11>.REFERENCE: 1005</Line11>
  <Line12>.AUTH CODE: 726529</Line12>
  <Line13>.</Line13>
  <Line14>.AMOUNT    USD$ 3.04</Line14>
  <Line15>..=====</Line15>
  <Line16>.TOTAL    USD$ 3.04</Line16>
  <Line17>.</Line17>
  <Line18>.. APPROVED - THANK YOU </Line18>
  <Line19>.</Line19>
  <Line20>.I AGREE TO PAY THE ABOVE TOTAL AMOUNT</Line20>
  <Line21>.ACCORDING TO CARD ISSUER AGREEMENT</Line21>
  <Line22>.(MERCHANT AGREEMENT IF CREDIT VOUCHER)</Line22>
  <Line23>.</Line23>
  <Line24>.</Line24>
  <Line25>.</Line25>
  <Line26>..x_____</Line26>
  <Line27>.. Merchant Signature </Line27>
  <Line28>.</Line28>
  <Line29>.</Line29>
  <Line30>..Application Label: Visa Credit</Line30>
  <Line31>..AID: A0000000031010</Line31>
  <Line32>..TVR: 0000008000</Line32>
  <Line33>..IAD: 06010A03800000</Line33>
  <Line34>..TSI: A800</Line34>
  <Line35>..CVM: SIGN</Line35>
</PrintData>
</RStream>

```

EMVVoidReturn

Request

```
<?xml version="1.0"?>
```

```
<TStream>
```

```
<Transaction>
```

```
<HostOrIP>127.0.0.1</HostOrIP>
```

```
<IpPort>9000</IpPort>
```

```
<MerchantID>755847002</MerchantID>
```

```
<OperatorID>User</OperatorID>
```

```
<UserTrace>DEMO</UserTrace>
```

```
<Memo>Demo POS 1.5 USEMV</Memo>
```

```

    <TranCode>EMVVoidReturn</TranCode>
    <SecureDevice>EMV_VX805_MERCURY</SecureDevice>
    <ComPort>9</ComPort>
    <InvoiceNo>1005</InvoiceNo>
    <RefNo>1005</RefNo>
    <AuthCode>726529</AuthCode>
    <Amount>
      <Purchase>3.05</Purchase>
    </Amount>
    <SequenceNo>0010010010</SequenceNo>
    <RecordNo>RecordNumberRequested</RecordNo>
    <Frequency>OneTime</Frequency>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>DEMO</UserTrace>
  </CmdResponse>
  <TranResponse>
    <MerchantID>755847002</MerchantID>
    <AcctNo>*****0010</AcctNo>
    <CardType>VISA</CardType>
    <TranCode>EMVVoidReturn</TranCode>
    <AuthCode>VOIDED</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>1005</RefNo>
    <InvoiceNo>1005</InvoiceNo>
    <OperatorID>User</OperatorID>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <Amount>
      <Purchase>3.05</Purchase>
      <Authorize>3.05</Authorize>
    </Amount>
    <ProcessData>|A4|410100500000</ProcessData>
    <RecordNo>+tYAexT+EnW/0+q2Zr4NJr+CiJ1SGUUFHkPpXPh5EkiEgUQESIQAC/A</RecordNo>
    <EntryMethod>CHIP</EntryMethod>
    <Date>07/31/2015</Date>
    <Time>10:13:34</Time>
    <ApplicationLabel>Visa Credit</ApplicationLabel>
    <AID>A0000000031010</AID>
    <TVR>0000008000</TVR>
    <IAD>06010A03800000</IAD>
    <TSI>A800</TSI>
    <CVM>SIGN</CVM>
  </TranResponse>
  <PrintData>
    <Line1>.MERCHANT ID: 755847004</Line1>
    <Line2>.CLERK ID: test</Line2>
    <Line3>.</Line3>
    <Line4>. VOID REFUND </Line4>
    <Line5>.</Line5>
    <Line6>*****0010</Line6>
    <Line7>.VISA ENTRY METHOD: CHIP</Line7>
    <Line8>.DATE: 07/31/2015 TIME: 10:13:34</Line8>
    <Line9>.</Line9>
  </PrintData>
</RStream>

```

```

<Line10>.INVOICE: 1005</Line10>
<Line11>.REFERENCE: 1005</Line11>
<Line12>.AUTH CODE: VOIDED</Line12>
<Line13>.</Line13>
<Line14>.AMOUNT          USD$ 3.05</Line14>
<Line15>.<Line15>
<Line16>.TOTAL          USD$ 3.05</Line16>
<Line17>.</Line17>
<Line18>.<Line18>
<Line19>.</Line19>
<Line20>.I AGREE TO PAY THE ABOVE TOTAL AMOUNT</Line20>
<Line21>.ACCORDING TO CARD ISSUER AGREEMENT</Line21>
<Line22>.(MERCHANT AGREEMENT IF CREDIT VOUCHER)</Line22>
<Line23>.</Line23>
<Line24>.</Line24>
<Line25>.</Line25>
<Line26>.x_____</Line26>
<Line27>.<Line27>
<Line28>.</Line28>
<Line29>.</Line29>
<Line30>.Application Label: Visa Credit</Line30>
<Line31>.AID: A0000000031010</Line31>
<Line32>.TVR: 0000008000</Line32>
<Line33>.IAD: 06010A03800000</Line33>
<Line34>.TSI: A800</Line34>
<Line35>.CVM: SIGN</Line35>
</PrintData>
</RStream>

```

VoidReturnByRecordNo This transaction uses a token and is card-not-present.

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <HostOrIP>127.0.0.1</HostOrIP>
    <IpPort>9000</IpPort>
    <MerchantID>755847002</MerchantID>
    <OperatorID>User</OperatorID>
    <UserTrace>DEMO</UserTrace>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <TranType>Credit</TranType>
    <TranCode>VoidReturnByRecordNo</TranCode>
    <SecureDevice>EMV_VX805_MERCURY</SecureDevice>
    <ComPort>9</ComPort>
    <InvoiceNo>1006</InvoiceNo>
    <RefNo>1006</RefNo>
    <AuthCode>026569</AuthCode>
    <Amount>
      <Purchase>4.12</Purchase>
    </Amount>
    <SequenceNo>0010010010</SequenceNo>
  <RecordNo>cgA6E9xDudKDjjqpaf4ukajlgWYVr7i0XJMduJpeGvYiEgUQESIQAC/Y</RecordNo>
    <Frequency>OneTime</Frequency>
    <ProcessData>|20|410100501000</ProcessData>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>

```



```

    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>DEMO</UserTrace>
  </CmdResponse>
  <TranResponse>
    <MerchantID>755847002</MerchantID>
    <AcctNo>476173XXXXXX0010</AcctNo>
    <ExpDate>XXXX</ExpDate>
    <CardType>VISA</CardType>
    <TranCode>Void Return</TranCode>
    <AuthCode>VOIDDED</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>1006</RefNo>
    <InvoiceNo>1006</InvoiceNo>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <Amount>
      <Purchase>4.12</Purchase>
      <Authorize>4.12</Authorize>
    </Amount>
    <AcqRefData>K</AcqRefData>
    <RecordNo>xRTiAAsGs1FAYPb9sgviFooXHsJooHmFHA8/SK5VdfwiEgUQESIQA/c</RecordNo>
    <ProcessData>|A4|410100600000</ProcessData>
  </TranResponse>
</RStream>

```

EMVVoiceAuth

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <HostOrIP>127.0.0.1</HostOrIP>
    <IpPort>9000</IpPort>
    <MerchantID>755847002</MerchantID>
    <OperatorID>User</OperatorID>
    <UserTrace>DEMO</UserTrace>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <TranCode>EMVVoiceAuth</TranCode>
    <SecureDevice>EMV_VX805_MERCURY</SecureDevice>
    <ComPort>9</ComPort>
    <InvoiceNo>2005</InvoiceNo>
    <RefNo>2005</RefNo>
    <AuthCode>123456</AuthCode> <!--the authorization code obtained over the phone-->
    <Amount>
      <Purchase>5.25</Purchase>
    </Amount>
    <SequenceNo>0010010010</SequenceNo>
    <PartialAuth>Allow</PartialAuth>
    <RecordNo>RecordNumberRequested</RecordNo>
    <Frequency>OneTime</Frequency>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>

```

```

<TextResponse>AP</TextResponse>
<SequenceNo>0010010010</SequenceNo>
<UserTrace>DEMO</UserTrace>
</CmdResponse>
<TranResponse>
  <MerchantID>755847002</MerchantID>
  <AcctNo>*****0010</AcctNo>
  <Card Type>VISA</Card Type>
  <Tran Code>EMVVoiceAuth</Tran Code>
  <Auth Code>123456</Auth Code>
  <CaptureStatus>Captured</CaptureStatus>
  <RefNo>1007</RefNo>
  <InvoiceNo>2005</InvoiceNo>
  <OperatorID>User</OperatorID>
  <Memo>Demo POS 1.5 USEMV</Memo>
  <Amount>
    <Purchase>5.25</Purchase>
    <Authorize>5.25</Authorize>
  </Amount>
  <ProcessData>|18|410100501000</ProcessData>
<RecordNo>qubp/LoBoHZe2A6qwoaM5KOKc0b6K1x1xGaDiO2j2siEgUQ ESIQAC/s</RecordNo>
  <EntryMethod>CHIP</EntryMethod>
  <Date>07/31/2015</Date>
  <Time>10:17:40</Time>
  <ApplicationLabel>Visa Credit</ApplicationLabel>
  <AID>A0000000031010</AID>
  <TVR>000000C000</TVR>
  <IAD>06010A03800000</IAD>
  <TSI>A800</TSI>
  <CVM>SIGN</CVM>
</TranResponse>
<PrintData>
  <Line1>.MERCHANT ID: 755847004</Line1>
  <Line2>.CLERK ID: test</Line2>
  <Line3>.</Line3>
  <Line4>. POST AUTH </Line4>
  <Line5>.</Line5>
  <Line6>*****0010</Line6>
  <Line7>.VISA ENTRY METHOD: CHIP</Line7>
  <Line8>.DATE: 07/31/2015 TIME: 10:17:40</Line8>
  <Line9>.</Line9>
  <Line10>.INVOICE: 2005</Line10>
  <Line11>.REFERENCE: 1007</Line11>
  <Line12>.AUTH CODE: 123456</Line12>
  <Line13>.</Line13>
  <Line14>.AMOUNT USD$ 5.25</Line14>
  <Line15>..=====</Line15>
  <Line16>.TOTAL USD$ 5.25</Line16>
  <Line17>.</Line17>
  <Line18>. APPROVED - THANK YOU </Line18>
  <Line19>.</Line19>
  <Line20>.I AGREE TO PAY THE ABOVE TOTAL AMOUNT</Line20>
  <Line21>.ACCORDING TO CARD ISSUER AGREEMENT</Line21>
  <Line22>.(MERCHANT AGREEMENT IF CREDIT VOUCHER)</Line22>
  <Line23>.</Line23>
  <Line24>.</Line24>
  <Line25>.</Line25>
  <Line26>..x</Line26>
  <Line27>.. Cardholder Signature </Line27>
  <Line28>.</Line28>
  <Line29>.</Line29>
  <Line30>.Application Label: Visa Credit</Line30>

```

```

    <Line31>.AID: A0000000031010</Line31>
    <Line32>.TVR: 000000C000</Line32>
    <Line33>.IAD: 06010A03800000</Line33>
    <Line34>.TSI: A800</Line34>
    <Line35>.CVM: SIGN</Line35>
  </PrintData>
</RStream>

```

EMVZeroAuth

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <HostOrIP>127.0.0.1</HostOrIP>
    <IpPort>9000</IpPort>
    <MerchantID>755847002</MerchantID>
    <OperatorID>User</OperatorID>
    <UserTrace>DEMO</UserTrace>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <TranCode>EMVZeroAuth</TranCode>
    <SecureDevice>EMV_VX805_MERCURY</SecureDevice>
    <ComPort>9</ComPort>
    <InvoiceNo>2007</InvoiceNo>
    <RefNo>2007</RefNo>
    <Amount>
      <Purchase>0.00</Purchase>      <!--note the value is zero -->
    </Amount>
    <SequenceNo>0010010010</SequenceNo>
    <PartialAuth>Allow</PartialAuth>
    <RecordNo>RecordNumberRequested</RecordNo>
    <Frequency>OneTime</Frequency>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>DEMO</UserTrace>
  </CmdResponse>
  <TranResponse>
    <MerchantID>755847002</MerchantID>
    <AcctNo>*****0010</AcctNo>
    <CardType>VISA</CardType>
    <TranCode>EMVZeroAuth</TranCode>
    <AuthCode>30766A</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>2007</RefNo>
    <InvoiceNo>2007</InvoiceNo>
    <OperatorID>User</OperatorID>
    <Memo>Demo POS 1.5 USEMV</Memo>
    <Amount>
      <Purchase>0.00</Purchase>
      <Authorize>0.00</Authorize>      <!--the card is not charged -->
    </Amount>
    <AcqRefData>b8238953#20#20#20#20#20#20#20e00</AcqRefData>
  </TranResponse>
</RStream>

```

```

        <ProcessData>|38|410100501000</ProcessData>
<RecordNo>6UrbpaDlBxWiiXwqzc7I/ThNG+VQPzVLvWoKMWLtuYciEgUQESIQADAH</RecordNo>
    <EntryMethod>CHIP</EntryMethod>
    <Date>07/31/2015</Date>
    <Time>10:19:34</Time>
    <ApplicationLabel>Visa Credit</ApplicationLabel>
    <AID>A0000000031010</AID>
    <TVR>000000C000</TVR>
    <IAD>06010A03600000</IAD>
    <TSI>F800</TSI>
    <ARC>00</ARC>
    <CVM>SIGN</CVM>
</TranResponse>
<PrintData>
    <Line1>.MERCHANT ID: 755847004</Line1>
    <Line2>.CLERK ID: test</Line2>
    <Line3>.</Line3>
    <Line4>.    VERIFY CARD    </Line4>
    <Line5>.</Line5>
    <Line6>.*0010</Line6>
    <Line7>.VISA    ENTRY METHOD: CHIP</Line7>
    <Line8>.DATE: 07/31/2015 TIME: 10:19:34</Line8>
    <Line9>.</Line9>
    <Line10>.INVOICE: 2007</Line10>
    <Line11>.REFERENCE: 2007</Line11>
    <Line12>.AUTH CODE: 30766A</Line12>
    <Line13>.</Line13>
    <Line14>.AMOUNT    USD$ 0.00</Line14>
    <Line15>.*0010</Line15>
    <Line16>.TOTAL    USD$ 0.00</Line16>
    <Line17>.</Line17>
    <Line18>.    APPROVED - THANK YOU    </Line18>
    <Line19>.</Line19>
    <Line20>.I AGREE TO PAY THE ABOVE TOTAL AMOUNT</Line20>
    <Line21>.ACCORDING TO CARD ISSUER AGREEMENT</Line21>
    <Line22>.(MERCHANT AGREEMENT IF CREDIT VOUCHER)</Line22>
    <Line23>.</Line23>
    <Line24>.</Line24>
    <Line25>.</Line25>
    <Line26>.*0010</Line26>
    <Line27>.*0010</Line27>
    <Line28>.*0010</Line28>
    <Line29>.*0010</Line29>
    <Line30>.Application Label: Visa Credit</Line30>
    <Line31>.AID: A0000000031010</Line31>
    <Line32>.TVR: 000000C000</Line32>
    <Line33>.IAD: 06010A03600000</Line33>
    <Line34>.TSI: F800</Line34>
    <Line35>.ARC: 00</Line35>
    <Line36>.CVM: SIGN</Line36>
</PrintData>
</RStream>

```

BatchSummary

The EMV *BatchSummary* transaction request contains several additional data elements than a non-EMV *BatchSummary*, which are highlighted below:

Request

```

<?xml version="1.0"?>
<TStream>
    <Admin>

```

```

<HostOrIP>127.0.0.1</HostOrIP> <!--IP address for NETePay-->
<IpPort>9000</IpPort> <!--IP Port for NETePay-->
<MerchantID>755847002</MerchantID>
<OperatorID>User</OperatorID>
<LaneID>01</LaneID><!--LaneID for the device-->
<UserTrace>kmartin</UserTrace>
<Memo>Test POS</Memo>
<TranType>Administrative</TranType>
<TranCode>Batch Summary</TranCode>
<SecureDevice>EMV_VX805_MERCURY</SecureDevice> <!--SecureDevice ID-->
<ComPort>9</ComPort> <!--Device Com Port-->
<SequenceNo>0010010010</SequenceNo>

```

```
</Admin>
```

```
</TStream>
```

Response

```
<?xml version="1.0"?>
```

```
<RStream>
```

```
<CmdResponse>
```

```
<ResponseOrigin>Processor</ResponseOrigin>
```

```
<DSIXReturnCode>000000</DSIXReturnCode>
```

```
<CmdStatus>Success</CmdStatus>
```

```
<TextResponse>OK</TextResponse>
```

```
<SequenceNo>0010010010</SequenceNo>
```

```
<UserTrace>kmartin</UserTrace>
```

```
</CmdResponse>
```

```
<BatchSummary>
```

```
<MerchantID>755847002</MerchantID>
```

```
<OperatorID>User</OperatorID>
```

```
<BatchNo>0056</BatchNo>
```

```
<BatchItemCount>2</BatchItemCount>
```

```
<NetBatchTotal>2.40</NetBatchTotal>
```

```
<CreditPurchaseCount>1</CreditPurchaseCount>
```

```
<CreditPurchaseAmount>1.15</CreditPurchaseAmount>
```

```
<CreditReturnCount>0</CreditReturnCount>
```

```
<CreditReturnAmount>0.00</CreditReturnAmount>
```

```
<DebitPurchaseCount>1</DebitPurchaseCount>
```

```
<DebitPurchaseAmount>1.25</DebitPurchaseAmount>
```

```
<DebitReturnCount>0</DebitReturnCount>
```

```
<DebitReturnAmount>0.00</DebitReturnAmount>
```

```
</BatchSummary>
```

```
</RStream>
```

BatchClose

The EMV *BatchClose* transaction request contains several additional data elements than a non-EMV *BatchSummary*, which are highlighted below:

Request

```
<?xml version="1.0"?>
```

```
<TStream>
```

```
<Admin>
```

```
<HostOrIP>127.0.0.1</HostOrIP> <!--IP address for NETePay-->
```

```
<IpPort>9000</IpPort> <!--IP port for NETePay-->
```

```
<MerchantID>755847002</MerchantID>
```

```
<OperatorID>User</OperatorID>
```

```
<LaneID>01</LaneID> <!--LaneID for Device-->
```

```
<UserTrace>kmartin</UserTrace>
```

```
<Memo>Test POS</Memo>
```

```
<TranType>Administrative</TranType>
```

```
<TranCode>BatchClose</TranCode>
```

```
<SecureDevice>EMV_VX805_MERCURY</SecureDevice> <!--SecureDevice ID-->
```



```

    <ComPort>9</ComPort>
    <SequenceNo>0010010010</SequenceNo>
  <BatchSummary>
    <MerchantID>755847002</MerchantID>
    <OperatorID>User</OperatorID>
    <BatchNo>0056</BatchNo>
    <BatchItemCount>2</BatchItemCount>
    <NetBatchTotal>2.40</NetBatchTotal>
    <CreditPurchaseCount>1</CreditPurchaseCount>
    <CreditPurchaseAmount>1.15</CreditPurchaseAmount>
    <CreditReturnCount>0</CreditReturnCount>
    <CreditReturnAmount>0.00</CreditReturnAmount>
    <DebitPurchaseCount>1</DebitPurchaseCount>
    <DebitPurchaseAmount>1.25</DebitPurchaseAmount>
    <DebitReturnCount>0</DebitReturnCount>
    <DebitReturnAmount>0.00</DebitReturnAmount>
  </BatchSummary>
</Admin>
</TStream>

Response
<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Success</CmdStatus>
    <TextResponse>OK</TextResponse>
    <SequenceNo>0010010010</SequenceNo>
    <UserTrace>kmartin</UserTrace>
  </CmdResponse>
  <BatchClose>
    <MerchantID>755847002</MerchantID>
    <OperatorID>User</OperatorID>
    <BatchNo>0056</BatchNo>
    <BatchItemCount>2</BatchItemCount>
    <NetBatchTotal>2.40</NetBatchTotal>
    <CreditPurchaseCount>1</CreditPurchaseCount>
    <CreditPurchaseAmount>1.15</CreditPurchaseAmount>
    <CreditReturnCount>0</CreditReturnCount>
    <CreditReturnAmount>0.00</CreditReturnAmount>
    <DebitPurchaseCount>1</DebitPurchaseCount>
    <DebitPurchaseAmount>1.25</DebitPurchaseAmount>
    <DebitReturnCount>0</DebitReturnCount>
    <DebitReturnAmount>0.00</DebitReturnAmount>
    <ControlNo>0056</ControlNo>
  </BatchClose>
</RStream>

```

<!--Device Com Port-->

3. Datacap Systems TranCloud™

Overview

Datacap's **TranCloud** is an easy to integrate and install cloud-based EMV payments hub perfect for tablet and browser-based applications that use HTTP POST in either XML or JSON format. It works with any operating system as long as the POS can access an HTTPS endpoint. In addition to EMV, *TranCloud* supports Gift and EBT transactions.

- *TranCloud* uses a locally installed hardware device to communicate with Datacap's *TranCloud* server.



TranCloud Appliance
Dimensions 3 x 2.5 x 0.5"

- The *TranCloud* server maintains an active connection with the appliance. Transactions are initiated with a request to the *TranCloud* server, which then relays the request to the on-site *TranCloud* appliance; the appliance then drives the payments device, which interacts with the cardholder to accept payment.
- *TranCloud* completes the transaction flow by sending the transaction to the processor and then relays the response through the *TranCloud* server back to the POS application, and the approve/decline response to the device.
- No software needs to be installed in the merchant's environment - both the EMV client and the NETePay applications are contained within the *TranCloud* appliance. The POS issues HTTP POSTs to Datacap's cloud server, and *TranCloud* handles the rest.



TranCloud:

- Works with any operating system as long as the POS can access an HTTPS endpoint.
- Offers both a JSON and XML API.
- Has the same functionality as dsiEMVUS, and also supports Gift and EBT.
- Supports both MercuryPay's tokenization and non-validated point to point encryption.
- Is capable of synchronous communication.

The POS terminal communicates information to the TranCloud service using an HTTP request, and then the TranCloud service communicates with the TranCloud device to facilitate payments and/or peripheral device control. The TranCloud device drives one or more pin pads to accept a payment method from the consumer and then packages all of that information up and sends it to the payment processor. The response then flows back through the TranCloud service to the POS terminal.

Note: Communication between the service and device is done on TCP port 5080, so any installations need to make sure that port is open in their firewall.

The TranCloud devices are extremely simple to setup

- Attach the device to a router to receive a dynamic IP Address
- Attach the device to power
- Attach one or more pin pad devices
- Process payments
- The TranCloud device is pre-configured so that when it arrives at a merchant location the merchant simply plugs it in, pairs it with your POS, and starts processing.

Supported Devices

TranCloud supports both serial and IP devices on the MercuryPay platform.

Current Available Serial Devices

- Ingenico iUC285 (kiosk)
- Ingenico Move 5000* (WiFi)
- Link 2500*
- Ingenico Lane 7000/8000 (signature capture)
- Ingenico Lane 3000*

*Move, Link and Lane are a part of the next generation of Tetra devices from Ingenico.

There is no special setup needed. Simply connect the device to the *TranCloud* appliance.

Note When ordering the iUC285 it is important to order it with the correct serial COM setting enabled – the iUC285 cannot be corrected in the field. In addition, be sure to order *both* a serial cable and a USB cable – the USB cable is used for connecting the power.

Available IP Devices

- Ingenico Lane 3000, 7000 and 8000
- Ingenico Move 5000 (wireless PIN pad), Link 2500

The **Ingenico Move 5000 wireless PIN pad** is capable of EMV, magstripe, manual and MSD contactless card entry. It is compatible with Vantiv Integrated Payments EMV integrations using non-validated point to point encryption technology. With its built-in receipt printer, the iWL258 can be used as a pay-at-table device, with some additional programming functions added to your POS software.

- For a deeper technical dive into the API specifications, supported transactions and operations supported with Datacap's TranCloud payments hub, visit:
<https://www.dsidevportal.com/api/trancloud/#what-is-trancloud>

Preparing for Integration and Testing

Test Credentials

1. Datacap will register your account with the TranCloud service for all testing and production processing. They will provide you with:
 - a unique **username** and **password**, which identifies your application to the TranCloud Service.
 - a unique **TranDeviceID** (format is PT#####). Provide this ID to your Implementations Consultant, who will pair it with a test Merchant ID.
2. Test devices and test credentials will be provided to you by your Implementations Consultant.

Setup the TranCloud appliance

1. Connect the appliance to the router to receive a dynamic IP address.
2. Attach the appliance to power.
3. If you are using a serial device, connect it directly to the appliance. If you are planning to use more than one device, you will need one appliance per device.
4. If you are using an IP device (WiFi or Ethernet), connect it to your router (must be the same router as the appliance).

Configure the TranCloud and EMV PIN pad—TestMode

1. **Configure the TranCloud appliance and test device, and set to test mode.** This directs the appliance to the MercuryPay CERT servers instead of production servers. Below is an example of an XML test mode request and response:

Request

```
<?xml version="1.0"?>
<TStream>
<Admin>
  <MerchantID>337234005</MerchantID>
  <TerminalID>001</TerminalID>
  <OperatorID>TEST</OperatorID>
  <UserTrace>Dev1</UserTrace>
  <TranCode>TestMode</TranCode> <!--requests test mode-->
  <SecureDevice>CloudEMV1</SecureDevice> <!--note SecureDevice-->
  <SequenceNo>0010010010</SequenceNo>
  <TranDeviceID>PT999999999</TranDeviceID>
  <PinPadIPAddress>10.0.0.155</PinPadIPAddress> <!--enter the IP address of your PIN pad-->
  <PinPadIPPort>1200</PinPadIPPort>
```

```
</Admin>
</TStream>
```

Response

```
<?xml version="1.0"?>
<RStream>
  <ResponseOrigin>Processor</ResponseOrigin>
  <DSIXReturnCode>000000</DSIXReturnCode>
  <CmdStatus>Success</CmdStatus>
  <TextResponse>TEST MODE ENABLED</TextResponse>
  <SequenceNo>0010010010</SequenceNo>
  <UserTrace>Dev1</UserTrace>
</RStream>
```

2. **Download the EMV parameters.** This step downloads the latest EMV parameters and security information for your PIN pad. The **EMVParamDownload** command should be performed when a new device is first installed, when a device is replaced, or when requested by the service provider.

Request

```
<?xml version="1.0"?>
<TStream>
  <Admin>
    <MerchantID>337234005</MerchantID>
    <TerminalID>001</TerminalID>
    <OperatorID>TEST</OperatorID>
    <UserTrace>Dev1</UserTrace>
    <TranCode>EMVParamDownload</TranCode>
    <SecureDevice>CloudEMV1</SecureDevice>
    <SequenceNo>0010010010</SequenceNo>
    <TranDeviceID>PT9999999999</TranDeviceID>
    <PinPadIPAddress>10.0.0.155</PinPadIPAddress>
    <PinPadIPPort>1200</PinPadIPPort>
  </Admin>
</TStream>
```

Response

```
<?xml version="1.0"?>
<RStream>
  <ResponseOrigin>Processor</ResponseOrigin>
  <DSIXReturnCode>000000</DSIXReturnCode>
  <CmdStatus>Success</CmdStatus>
  <TextResponse>Success</TextResponse>
  <SequenceNo>0010010010</SequenceNo>
  <UserTrace>Dev1</UserTrace>
  <TranCode>EMVParamDownload</TranCode>
</RStream>
```

3. **Reset the PIN pad.** The **EMVPadReset** command must be performed before every transaction. This ensures that no card is in the EMV PIN pad chip card reader. This command should also be performed at the end of any card-related transaction to notify the cardholder to remove their card.

Request

```
?xml version="1.0"?>
<TStream>
  <Transaction>
    <MerchantID>337234005</MerchantID>
    <TerminalID>001</TerminalID>
    <OperatorID>TEST</OperatorID>
    <UserTrace>Dev1</UserTrace>
    <TranCode>EMVPadReset</TranCode>
    <SecureDevice>CloudEMV1</SecureDevice> <!--Note SecureDevice is CloudEMV1; CloudEMV2 is also
available-->
```

```

<SequenceNo>0010010010</SequenceNo>
<TranDeviceID>PT999999999</TranDeviceID>
<PinPadIPAddress>10.0.0.155</PinPadIPAddress>
<PinPadIPPort>1200</PinPadIPPort>
</Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <ResponseOrigin>Client</ResponseOrigin>
  <DSIXReturnCode>000000</DSIXReturnCode>
  <CmdStatus>Success</CmdStatus>
  <TextResponse>Reset Successful.</TextResponse>
  <SequenceNo>0010010010</SequenceNo>
  <UserTrace>Dev1</UserTrace>
</RStream>

```

TranCloud Transactions

Structure of a TranCloud XML Request and Response

For a complete list supported EMV, PrePaid/Gift, EBT, FSA and Admin transactions, refer to the Datacap TranCloud API at <https://www.dsidevportal.com/api/trancloud/#emv-transactions>.

The example below illustrates the typical XML tags required by MercuryPay for TranCloud transactions. This example shows an EMV Sale with a one-time use token request. The structure will look familiar to those of you familiar with other Datacap/MercuryPay XML formats. The key differences are:

1. The *MerchantID* is the **Device ID**.
2. The request includes the <SecureDevice> tag, which is set to **CloudEMV1** or **CloudEMV2**. This value routes the transaction to the TranCloud Server.
3. The <PinPadIpAddress> and <PinPadIpPort> tags must be included if you are using an IP PIN Pad.

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <MerchantID>PT7114732006</MerchantID><!--Merchant ID uses the Device ID-->
    <OperatorID>test</OperatorID>
    <UserTrace>DevInt</UserTrace>
    <TranCode>EMVSale</TranCode>
    <SecureDevice>CloudEMV1</SecureDevice><!--Note SecureDevice is CloudEMV1; CloudEMV2 is also
available-->
    <InvoiceNo>01</InvoiceNo><!--In request, Invoice No = RefNo-->
    <RefNo>01</RefNo>
    <Amount>
      <Purchase>5.01</Purchase>
    </Amount>
    <SequenceNo>0010010010</SequenceNo>
    <PartialAuth>Allow</PartialAuth>
    <RecordNo>RecordNumberRequested</RecordNo> <!--this is for tokenization-->
    <Frequency>OneTime</Frequency>
    <TranDeviceID>PT7114732006</TranDeviceID>
    <PinPadIpAddress>10.0.0.99</PinPadIpAddress> <!--required for IP PIN pads-->
    <PinPadIpPort>12000</PinPadIpPort> <!--required for IP PIN pads-->
  </Transaction>
</TStream>

```


4. Web Services Direct: SOAP or REST/JSON for non EMV

Overview

This section provides details on MercuryPay's Web Services methods. The Web Services platform was designed for developers who prefer using a hosted end-point and thereby by-passing the need to install a distributed client control or NETePay. Because of this, there is an easier integration path, but also has certain device, EMV and feature limitations.

The methods described here are specific the Web Services but the transaction payloads are derived from Datacap XML specifications. This method can be implemented in two ways:

SOAP

The XML transaction requests are inserted into a Simple Object Access Protocol (SOAP) envelope to make a **CreditTransaction()** call to MercuryPay's Web Service URLs. MercuryPay Web Services extracts the XML payload and routes the request to our processing servers for authorization. The transaction XML response is then re-inserted into a SOAP envelope and returned to the POS.

CreditTransaction()

Credit, U.S. PIN Debit, EBT, CardLookup, ZeroAuth, BatchSummary and BatchClose

Argument Description

Two arguments: **CreditTransaction**(trans as string, password as string)

1. **trans** is an XML string formatted using Datacap's standard *DSIClientX* specifications.
2. **password** is a string that Worldpay from FIS assigns for a particular Merchant ID used in part to authenticate the credentials when calling Web Services.
 - After processing, web service returns an XML string containing the transaction response data.
 - **BatchSummary** and **BatchClose** functions also use the **CreditTransaction()** method, along with the **BatchSummary** and **BatchClose** XML payloads.

REST/PaymentsAPI

MercuryPay's RESTful API is a lightweight data-interchange format that utilizes JSON RPC (remote procedure calls) developed in response to the demand from mobile developers. These JSON formatted requests are sent to our PaymentsAPI URLs for processing.

- Authentication JSON RPC occurs via https authorization header.
- Username: MerchantID
- Password: generated and stored by Worldpay form FIS
- Create string [Username]:[Password], Base64 encode, set Authorization header to "Basic [encoded string]"
- Content-Type: "application/json"

Post: transactions are posted to Mercury Web Services PaymentsAPI URL followed by the transaction type in the following format: **https://[w1.mercury server/PaymentsAPI]/[TranType]/[TranCode]** as, for example when posting to our test/cert platform: **https://w1.mercurycert.net/PaymentsAPI/Credit/Sale**

Comparing Integration Methods

Standard transaction processing functions are common to both Datacap/NETePay and Web Service integrations, but there are inherent differences. While there is an ease of integration and time saving convenience to using Datacap's NETePay and client controls, there are a number of Web Service methods that can be used to pull detailed batching records directly from MercuryPay's reporting logs.

Platform Functionality and Features	NETePay/ DSIClientX, dsiEMVUS, dsiPDCX	Web Services	
		SOAP	RESTful
Transaction Support: Credit, Debit, EBT, FSA, Gift/StoreCard, Incremental Auth, ZeroAuth, CardLookup	✓	✓	Credit, Debit and Gift

Director 5 Automatic Upgrades	NETePay	--	--
EMV (Credit, Debit, Contactless)	✓	--	--
Peripheral device support—Ingenico, IDTech, Verifone: P2PE, EMV, all CVMs	dsiEMVUS and dsiPDCX	Direct device integration required	Direct device integration required
Store and Forward	✓	--	--
Pay at Table	✓	--	--
Credentials on File		✓	--
Account Updater	DSIClientX Direct	✓	--
Batch Summary and BatchClose	✓	✓	✓
MercuryPay P2PE and MToken	✓	✓	✓
Canadian Debit and Canadian EMV	✓	--	--

Web Services SOAP Integration Method Details

MercuryPay's Web Services platform is designed for POS systems that do not utilize a Windows ActiveX control. The Web Services SOAP integration method uses the same XML transaction formats as described in the **Platform Integration Guide**, only inserted into a **Simple Object Access Protocol (SOAP)** envelope. For this method, additional merchant credentials are required, that is a Web Services Password.

- This SOAP-wrapped XML is then used to make a **CreditTransaction()** call to MercuryPay's Web Service servers.
- MercuryPay "unwraps" the transaction request, handles the transaction XML payload, and passes the request to the back-end processing networks.
- The transaction response is then returned to the POS as a standard XML response <RStream> wrapped in a SOAP envelope.

This document outlines connecting to the Web Services servers and the Web Services methods available.

MerchantID and Web Services Password

A **Merchant ID** and Web Services **password** are required for authenticating a transaction to the Web Services servers.

Merchant ID

Web Services Merchant IDs follow the same format as other integration methods. For security reasons on inbound transaction processing, MercuryPay uses a compound numeric **MerchantID** and an alpha "**Nickname**" joined by an "=" sign. The typical format for a MercuryPay merchant ID is **8843XXXXXX=NICKNA**

- Allow up to 24 Alpha-numeric characters total (no spaces) for MerchantID.
- Alpha characters in the *Nickname* are not case-sensitive.

Password

A unique Web Services password is provided by Worldpay from FIS IP upon merchant account set up.

- Allow up to 24 alpha-numeric and special characters.
- For ease of testing, passwords will be simplified, typically using "xyz".
- Production passwords are highly secured; passwords are managed at the time of merchant on-boarding and will be made available via the merchant's portal access utility such as MercuryView or iQ.

Web Services "Merchant ID not Found" and "Invalid Credentials" Errors:

If a transaction reaches the Web Services servers, *but the merchant account cannot be authenticated*, the error "Merchant ID not found" or "Invalid Credentials" will occur. This may occur when:

- The SOAP envelope is malformed, the XML is incorrect, or the *MerchantID* or Web Services password cannot be extracted.
- The *MerchantID* or Web Services password is incorrect. (Error: "Invalid Credentials Call 1800-846-4472.")
- The *MerchantID* or Web Services password is not present on the MercuryPay Web Services servers.

Web Services Primary and Secondary URLs

Two Primary and Secondary Web Services URLs are used for backup redundancy and reliability. MercuryPay's Web Services **W1** and **W2** servers are geographically separate, yet act as parallel, fully functioning entry points to MercuryPay's processing platform.

Server	URL
Development Testing W1 Primary Server (Only the primary is used in testing)	https://w1.mercurycert.net/ws/ws.asmx
Production Web Services W1	https://w1.mercurypay.com/ws/ws.asmx
Production Web Services W2	https://w2.backuppays.com/ws/ws.asmx
MercuryPay's Web Services Direct Libraries	
Development Test/Certification WSDL	https://w1.mercurycert.net/ws/ws.asmx?WSDL
Production WSDL	https://w1.mercurypay.com/ws/ws.asmx?WSDL

Failover between primary and secondary Web Services (WS) servers

MercuryPay's primary and secondary WS URLs are entry points to a much larger network and though the need to fail over is rare, determining the need to roll over to an alternative WS server is most typically determined

- In the event of a connection error response, or
- In the event of any response that is not *Approved*, *Captured* or *Declined*.

Web Services developers monitor and build in provisions to failover to the alternate Web Server in the event of a processing connection failure. If there is a connection failure, it is recommended to fail over to the alternate server and remain at that URL.

- ***It is not necessary to fail back to the previous server or to continue to "try and catch" the available server.*** If both WS servers are returning http response connectivity errors, contact Worldpay from FIS IP.
- If a POS system receives a WS http error (e.g. 503, 500, 404) indicating the connectivity to the WS server has failed, is lost or unavailable, then this is a good indication to immediately rollover to the secondary server. (Exceptions to this would of course include formatting or parsing errors, e.g. *400-Bad Request*, in which case, rolling over to an alternative WS server would not correct the root cause of the error.)
- Note that WS connectivity errors can happen at several points along the transaction routing path, making the diagnosis of the need to failover more complex. MercuryPay's WS servers receive, reformat and process requests to the back-end authorization networks. Connectivity error responses beyond MercuryPay's servers typically do not indicate a need to fail over.

Using MercuryPay's Web Services Libraries

Web Services are Web APIs that can be accessed over a network and/or the internet, and are executed on a remote system hosting the requested services.

- MercuryPay's Web Service development URL is <https://w1.mercurycert.net/ws/ws.asmx> with the full WSDL accessed at: <https://w1.mercurycert.net/ws/ws.asmx?WSDL>
- Any development environment supporting Web Services can add a reference to that URL and have access to the methods and classes it provides.

The Web Services Methods and Arguments

Note The XML string specifies the necessary transaction information (payload) that is sent as an argument of the methods.

CreditTransaction()	Argument Description
Send Credit, U.S. PIN Debit, EBT, CardLookup, BatchSummary and BatchClose.	<p>Two arguments: CreditTransaction(trans as string, password as string)</p> <ol style="list-style-type: none"> trans is an XML string formatted using MercuryPay's standard <i>DSIClientX</i> transaction details as outlined in the <i>Platform Integration Guide</i>. Password is a string that MercuryPay assigns for a particular Merchant ID. This is the web services password MercuryPay uses to authenticate merchants sending us transactions on our Web Services platform. <p>After processing, web service returns an XML string containing the transaction response data</p> <p>BatchSummary and BatchClose functions also use the CreditTransaction() method, along with the standard DSIClientX BatchSummary and BatchClose XML payloads in the WS request.</p>
GiftTransaction()	Argument Description
Send a gift card transaction	<p>Two arguments: GiftTransaction(trans as string, password as string)</p> <ol style="list-style-type: none"> trans is an XML string formatted with transaction details password is a string that MercuryPay assigns for a particular merchant ID <p>Example: s = x.GiftTransaction(trans,"xyz")</p> <p>Note IP Port 9100 must be included in the soap wrapped <i>GiftTransaction()</i> request for Prepaid/gift card transactions</p>

Note: MercuryPay is no longer supporting development work using CBatch(), CTranDetail(), CAllDetail().

Example: Web Services Transaction using CreditTransaction()

The Web Services method *CreditTransaction()* is used for all Credit, Debit, EBT, Foodstamp, FSA and Admin functions.

There are two arguments:

- tran** (transaction XML). This is the same XML outlined in the *DSIClientX Programming Interface Specifications* (with the MercuryPay specific exceptions) sent as a string.
- pw** (password). This is the web services password we use to authenticate merchants sending us transactions on our Web Services platform sent as a string.

Below is an example of a Web Services *CreditTransaction* string completely filled out using *MerchantID 112438931=WSTEST* and corresponding web services password of **xyz**. (In this example "w" is the instance of the Web Services for this project.)

```
w.CreditTransaction
("xmlVersion=""1.0""?&gt;
&lt;TStream&gt;
  &lt;Transaction&gt;
    &lt;MerchantID&gt;112438931=WSTEST &lt;/MerchantID&gt;
    &lt;LaneID&gt;02&lt;/LaneID&gt; &lt;!--required for multi-terminal environments--&gt;
    &lt;TranType&gt;Credit&lt;/TranType&gt;
    &lt;TranCode&gt;Sale&lt;/TranCode&gt;
    &lt;InvoiceNo&gt;1&lt;/InvoiceNo&gt;
    &lt;RefNo&gt;1&lt;/RefNo&gt;
    &lt;Memo&gt;Your Memo&lt;/Memo&gt;&lt;!--Also POSPackageID for POS name and version--&gt;
    &lt;Account&gt;
    &lt;Track2&gt;Track Data&lt;/Track2&gt;&lt;!--for PCI scope reduction use Encryption --&gt;
    &lt;Name&gt;MPS TEST&lt;/Name&gt;
    &lt;/Account&gt;
    &lt;Amount&gt;</pre

```

```

        <Purchase>1.23</Purchase>
      </Amount>
      <OperatorID>MPS</OperatorID>
    </Transaction>
  </TStream>", "xyz")

```

Web Services: POST

Developers may opt to not use the Web Services Library and instead code at a lower level using a POST command, sending the transaction via HTTP POST directly to the MercuryPay's Web Services platform. Reasons for doing this include:

- To make the code more light-weight
- Some programming environments have a web services implementation that is not compatible with MercuryPay's Web Services platform. Even though Web Services is an open standard, different implementations of the standard yield compatibility errors.
- Some older development environments do not support Web Services. Direct Posting is then the preferred option.

Example POST Credit Transaction

- The data that is POST-ed to our Web Services platform begins by specifying the method to be implemented in the headers. The headers tell the web server how to process the POST.
- After the header is the message body is the data or payload of the transaction to be handled by the method being specified. Web Services expects the data to be SOAP wrapped.

The example below illustrates a simple a credit sale request POST to MercuryPay's Web Services platform using the **CreditTransaction** method.

User-Agent: MPS Transact 1.2.0.4

Content-Type: text/xml; charset=utf-8

SOAPAction: http://www.mercurypay.com/CreditTransaction

```

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <CreditTransaction xmlns="http://www.mercurypay.com">
      <tran>&lt;?xml version="1.0"?&gt;
        &lt;TStream&gt;
          &lt;Transaction&gt;
            &lt;MerchantID&gt;112438931=TEST&lt;/MerchantID&gt;
            &lt;LaneID&gt;001&lt;/LaneID&gt;
            &lt;OperatorID&gt;Operator&lt;/OperatorID&gt;
            &lt;TranType&gt;Credit&lt;/TranType&gt;
            &lt;TranCode&gt;Sale&lt;/TranCode&gt;
            &lt;InvoiceNo&gt;10&lt;/InvoiceNo&gt;
            &lt;RefNo&gt;10&lt;/RefNo&gt;
            &lt;Memo&gt;MPS Memo 1.2.0.4&lt;/Memo&gt;
            &lt;Account&gt;
              &lt;Track2&gt;TrackData&lt;/Track2&gt;
              &lt;Name&gt;MPS TEST&lt;/Name&gt;
            &lt;/Account&gt;
            &lt;Amount&gt;
              &lt;Purchase&gt;1.75&lt;/Purchase&gt;
            &lt;/Amount&gt;
          &lt;/Transaction&gt;
        &lt;/TStream&gt;</tran>
      </pw>xyz</pw>
    </CreditTransaction>
  </soap:Body>
</soap:Envelope>

```


Guide to Credential on File/Card-on-File on MercuryPay WebServices Platform

Many point-of-sale (POS) applications support features that enable cardholders to securely store credit card information on the POS system for future use. With cardholder data being able to be replaced by a processor-generated or network token, and as security concerns balance convenience of having a card registered on file with a favorite merchant, this practice has become broadly adopted. The "Credential on File" card brand mandate new regulations are being put in place to address this broader marketplace and bring greater definition to subsequent recurring transactions being processed from payment applications that store cardholder information.

On the MercuryPay platform, supporting this mandate requires two modifications: 1. Implement a new tag, *RecurringData*, which defines what type of recurring transaction you are requesting and 2. Retain the corresponding response value returned in *RecurringData* and include in the next recurring request.

Four types of *RecurringData*:

1 Recurring

A merchant initiated transaction processed with a **variable amount** over **regular intervals**

2 Installment

A merchant initiated transaction processed with a **fixed amount** over **fixed period of time**.

3 Unscheduled

A merchant initiated transaction processed with a **fixed or variable amount** **without a regular interval** or schedule.

4 Cardholder Initiated

A cardholder initiated transaction processed using a **variable amount** and **without a regular interval** or schedule.

Credentials on File Transaction Flow

On the initial request use the ***RecurringData*** in the Request tag based on the recurring transaction type you are sending:

RecurringData = "Recurring", "Installment", "Unscheduled" or "CardHolderInitiated"

The response will echoe your requested value

RecurringData = "Recurring", "Installment", "Unscheduled" or "CardHolderInitiated"

And return a new recurring response identifier:

RecurringData = "Recurring|00|01234567891234",
"Installment|00|01234567891234", "Unscheduled|00|01234567891234" or
"CardHolderInitiated|00|01234567891234"

Retain the full response value from the ***RecurringData*** and use this in the next transaction in the recurring series as the ***RecurringData***.

Credentials on File Input Parameters and Response values

Parameter	Type	Length	Description
RecurringData	String	0-40	<blank> Recurring Recurring 00 Recurring 00 123456789012345 Installment Installment 00 Installment 00 123456789012345 Unscheduled Unscheduled 00 Unscheduled 00 123456789012345 CardHolderInitiated CardHolderInitiated 00 CardHolderInitiated 00 123456789012345 Case Sensitive. (Installment, Unscheduled and CardHolderInitiated are specific to Visa only)

Card-on-File Definitions, Cardholder Agreements and Technical Details

Recurring Payment

A Recurring payment is defined as a merchant-initiated transaction in a series of transactions which uses a stored credential, processed with a variable amount over regular intervals. Intervals may not exceed one year between transactions. Cardholder agreement/consent is required for the merchant to initiate recurring transactions.

Examples of Recurring payments include subscription payments and bill payments such as an electric bill, gym membership, pest control, monthly car insurance payment, mobile phone bill, etc.

Recurring Cardholder Agreement

When entering into a Recurring cardholder agreement, the following information must be clearly displayed at the time the cardholder gives their consent. This information is to be displayed separately from the general purchase terms and conditions. Where required by applicable laws or regulations, the merchant or its agent must also provide to the cardholder a record of the cardholder's consent.

- A truncated version of the stored credential (e.g., last 4 digit of the account number)
- Method in which the cardholder will be notified of any changes in the agreement
- How the stored credential will be used
- Expiration date of the agreement when applicable

Additionally, before processing the transaction, the merchant must obtain the cardholder's consent to an agreement that contains the following:

- Transaction amount (includes all taxes and charges) or how the transaction amount will be determined.
- Transaction amount cannot contain finance charges, which must be listed separately.
- Transaction currency
- If surcharging is permitted and assesses, cardholder acknowledgement of any surcharges and associated disclosures
- Cancellation and refund policies
- Location of the merchant outlet
- The fixed dates or intervals on which the transaction will be processed

- The cardholder agreement must be retained for the duration of the agreement and provide it to the issuer upon request

Recurring Payment Cancellation Procedures

The merchant must provide a simple cancellation procedure to the cardholder. If the cardholder's order was initially accepted online, the merchant may provide an online cancellation procedure. The merchant cannot complete a transaction when:

- The transaction is beyond the agreed upon duration by the cardholder
- The cardholder request that the merchant change the payment method
- Cardholder cancels according to the agreed cancellation policy
- The merchant receives a decline response

First Time a Recurring Payment Credential is Stored

- Merchant to ask the cardholder if they'd like the merchant to store their payment credential (card) for future transactions prior to submitting the initial/first authorization request
- The transaction must be authorized either for the amount due or, if payment not required, by submitting a ZeroAuth request (\$0.00). Credentials may not be stored if the authorization declines.

Note on Response: The numbers following Recurring [00]32104567893210 combines a piped delimited response code [00] and the network generated numeric transaction identifier. This identifier is card brand specific and, at this time, Visa, MasterCard and Discover return a network transaction identifier. It is important to return the full *RecurringData* response value in any subsequent COF transaction request.

For Visa, MasterCard and Discover, a new *RecurringData* network transaction ID is passed back with each response. This should be saved and used in the subsequent COF transactions.

For other card brands allow "Recurring" or "Recurring[00]".

- Should a recurring transaction decline, the merchant may perform a resubmission. Merchant may attempt to resubmit the transaction for up to 4 times within 16 calendar days from the date of the original decline response. The resubmission of authorization can only be performed when the original decline response code was one of the following:
 - 05 (Authorization declined)
 - 51 (Insufficient funds)
 - 61 (Exceeds approval amount limit)
 - 65 (Exceeds withdrawal frequency limit)
- When the authorization is approved, retain the full *RecurringDataResponse* to submit in subsequent authorization request.

Subsequent Recurring Transactions

- The transaction must be authorized.
- Submit valid formatted *RecurringData* field. The following are valid Recurring formats:

```
<blank>
Recurring
Recurring[00]
Recurring[00]314282069480098
```

Installment

An installment is a merchant initiated transaction in a series of transactions that uses a stored credential to make a payment for a fixed amount over a fixed period of time. Cardholder agreement/consent is required for the merchant to initiate installment transactions.

Examples of an Installment COF include furniture purchases or home shopping network purchase where the amount paid is a fixed amount over a fixed period of time.

Installment Cardholder Agreement

When entering into a cardholder agreement, the following information must be clearly displayed at the time the cardholder gives their consent. This information is to be displayed separately from the general purchase terms and conditions. Where required by applicable laws or regulations, the merchant or its agent must also provide to the cardholder a record of the cardholder's consent.

- A truncated version of the stored credential (e.g., last 4 digit of the account number)
- Method in which the cardholder will be notified of any changes in the agreement
- How the stored credential will be used
- Expiration date of the agreement when applicable

Additionally, before processing the transaction, the merchant must obtain the cardholder's express informed consent to an agreement that contains the following:

- Transaction amount (includes all taxes and charges) or how the transaction amount will be determined
- The amount may not include interest charges for the U.S. Region
- Transaction currency
- If surcharging is permitted and assesses, cardholder acknowledgement of any surcharges and associated disclosures
- Cancellation and refund policies
- Location of the merchant outlet
- The total purchase price
- Terms of future installment payments, including the dates, amounts, and currency
- The cardholder agreement must be retained for the duration of the agreement and provide it to the issuer upon request

Installment Cancellation Procedures

The merchant must provide a simple cancellation procedure to the cardholder. If the cardholder's order was initially accepted online, the merchant may provide an online cancellation procedure. The merchant cannot complete a transaction when:

- The transaction is beyond the agreed upon duration by the cardholder
- The cardholder request that the merchant change the payment method
- Cardholder cancels within the terms of the cancellation policy. Merchant must provide the cardholder a cancellation or refund confirmation in writing within 3 business days
- The merchant receives a decline response

First Time an Installment Credential is Stored

- Merchant to ask the cardholder if they'd like the merchant to store their payment credential (card) for future transactions prior to submitting the initial/first authorization request

- The transaction must be authorized either for the amount due or, if payment not required, by submitting a ZeroAuth request (\$0.00). Credentials may not be stored if the authorization declines.
- Submit correct `RecurringData = "Installment"`
- When the authorization is approved, retain the full *RecurringData* to submit in subsequent installment authorization request.

Note: A merchant may not process an initial installment transaction until the merchandise or services have been provided to the cardholder and must not process individual installment transaction at intervals less than either 7 calendar days or the monthly anniversary of the shipment date.

Subsequent Installment Transactions

- The transaction must be authorized.
- Submit valid formatted *RecurringData* field. The following are valid formats:

```
<blank>
Installment|00|314282069480098
Installment
Installment|00
```

Note on Response: The numbers following Installment |00|32104567893210 combines a piped delimited authorization code |00| and the network generated numeric transaction identifier. This identifier is card brand specific and, at this time, only Visa returns a network transaction identifier portion in this response. It is important to return the full *RecurringDataResponse* value in any subsequent COF transaction request.

For Visa, a new *RecurringData* network transaction ID—is passed back with each response. This should be saved and used in the subsequent COF transactions.

For other card brands allow "Installment" or "Installment|00|".

- Should a subsequent installment transaction decline, the merchant must notify the cardholder in writing and allow the cardholder at least 7 days to pay by other means
- Should an authorization decline the merchant may perform a resubmission. Merchant may attempt to resubmit the transaction for up to 4 times within 16 calendar days from the date of the original decline response. The resubmission of authorization can only be performed when the original decline response code was one of the following:
 - 05 (Authorization declined)
 - 51 (Insufficient funds)
 - 61 (Exceeds approval amount limit)
 - 65 (Exceeds withdrawal frequency limit)

Unscheduled (Visa Only)

An Unscheduled COF is defined as a merchant initiated transaction in a series that uses a stored credential, processed without a regularly scheduled transaction date and with a fixed or variable amount. Cardholder agreement/consent is required for the merchant to initiate unscheduled COF transactions.

Examples of Unscheduled COF payments include services such as snowplow service, window clearer services or vacation home cleaning services where there are agreed upon dependencies of time and conditions for the service.

Unscheduled Cardholder Agreement

When entering into a cardholder agreement, the following information must be clearly displayed at the time the cardholder gives their consent. This information is to be displayed separately from the general purchase terms

and conditions. Where required by applicable laws or regulations, the merchant or its agent must also provide to the cardholder a record of the cardholder's consent.

- A truncated version of the stored credential (e.g., last 4 digit of the account number)
- Method in which the cardholder will be notified of any changes in the agreement
- How the stored credential will be used
- Expiration date of the agreement when applicable

Additionally, before processing the transaction, the merchant must obtain the cardholder's express informed consent to an agreement that contains the following:

- Transaction amount (includes all taxes and charges) or how the transaction amount will be determined.
- The agreed upon event that will prompt the transaction (for example: to plow only when two inches of snow or more or the vacation home rental dates for cleaning.)
- Transaction currency
- If surcharging is permitted and assesses, cardholder acknowledgement of any surcharges and associated disclosures
- Cancellation and refund policies
- Location of the merchant outlet
- The fixed dates or intervals on which the transaction will be processed
- The cardholder agreement must be retained for the duration of the agreement and provide it to the issuer upon request

Unscheduled Cancellation Procedures

The merchant must provide a simple cancellation procedure to the cardholder. If the cardholder's order was initially accepted online, the merchant may provide an online cancellation procedure. The merchant cannot complete a transaction when:

- The transaction is beyond the agreed upon duration by the cardholder
- The cardholder request that the merchant change the payment method
- Cardholder cancels according to the agreed cancelation policy
- The merchant receives a decline response

First Time an Unscheduled Credential is Stored

- Merchant to ask the cardholder if they'd like the merchant to store their payment credential (card) for future transactions prior to submitting the initial/first authorization request
- The transaction must be authorized either for the amount due or, if payment not required, by submitting a ZeroAuth request (\$0.00). Credentials may not be stored if the authorization declines.
- Submit correct `RecurringData = "Unscheduled"`
- When the authorization is approved, retain the full *RecurringData* to submit in subsequent authorization request.

Subsequent Transactions -- Unscheduled

- The transaction must be authorized.
- Submit valid formatted *RecurringDataRequest* field. The following are valid formats:

<blank>

Installment|00|314282069480098

Installment

Installment|00|

Note on Response: The numbers following Unscheduled |00|32104567893210 combines a piped delimited authorization code |00| and the network generated numeric transaction identifier. This identifier is card brand specific and, at this time, only Visa returns a network transaction identifier. It is important to return the full *RecurringData* response value in any subsequent COF transaction request.

For Visa, a new *RecurringData* network transaction ID is passed back with each response. This should be saved and used in the subsequent COF transactions.

For other card brands allow "Unscheduled" or "Unscheduled|00|".

- Should an Unscheduled transaction decline, the merchant must notify the cardholder in writing and allow the cardholder at least 7 days to pay by other means
- Should an Unscheduled transaction decline the merchant may perform a resubmission. Merchant may attempt to resubmit the transaction for up to 4 times within 16 calendar days from the date of the original decline response. The resubmission of authorization can only be performed when the original decline response code was one of the following:
 - 05 (Authorization declined)
 - 51 (Insufficient funds)
 - 61 (Exceeds approval amount limit)
 - 65 (Exceeds withdrawal frequency limit)

Cardholder Initiated

A Cardholder Initiated payment is defined as a transaction in a series of transactions that uses a stored credential is neither determined by a fixed date or amount and is initiated by the cardholder. Cardholder agreement/consent is required for the merchant to store the credentials for future transactions.

Examples of Cardholder Initiated include online retail or restaurant merchant environments that store the cardholder's payment credentials, shipping and billing address is on file, such as Amazon, The U.S. Post Office or a Pizza delivery business.

Cardholder Initiated Agreement

When entering into a cardholder agreement, the merchant is establishing the terms and conditions of the consent to store the cardholder's payment credentials, shipping and billing address. The following information must be clearly displayed at the time the cardholder gives their consent. This information is to be displayed separately from the general purchase terms and conditions. Where required by applicable laws or regulations, the merchant or its agent must also provide to the cardholder a record of the cardholder's consent.

- A truncated version of the stored credential (e.g., last 4 digit of the account number)
- Method in which the cardholder will be notified of any changes in the agreement
- How the stored credential will be used
- Expiration date of the agreement when applicable
- Transaction currency
- If surcharging is permitted and assesses, cardholder acknowledgement of any surcharges and associated disclosures
- Cancellation and refund policies
- Location of the merchant outlet
- The cardholder agreement must be retained for the duration of the agreement and provide it to the issuer upon request

Cardholder Initiated Cancellation procedures

The merchant must provide a simple cancellation procedure to the cardholder. If the cardholder's order was initially accepted online, the merchant may provide an online cancellation procedure. The merchant cannot complete a transaction when:

- The transaction is beyond the agreed upon duration by the cardholder
- The cardholder request that the merchant change the payment method
- Cardholder cancels within the terms of the cancelation policy
- The merchant receives a decline response

First Time a Cardholder Initiated Payment Credential is Stored

- Merchant to ask the cardholder if they'd like the merchant to store their payment credential (card) for future transactions prior to submitting the initial/first authorization request
- The transaction must be authorized either for the amount due or, if payment not required, by submitting a ZeroAuth request (\$0.00). Credentials may not be stored if the authorization declines.
- Submit correct `RecurringData = "CardHolderInitiated"` in `InitPayment` call and capture response details including `RecurringDataResponse` in `VerifyPayment` call.
- When the authorization is approved, retain the full `RecurringDataResponse` to submit in subsequent authorization request.

Subsequent Transactions

- The merchant must validate the cardholder's identity (example: login ID and password) prior to processing each transaction
- The transaction must be authorized.
- Submit valid formatted `RecurringDataRequest` field. The following are valid formats:

```
<blank>
CardHolderInitiated
CardHolderInitiated|00|
CardHolderInitiated|00|314282069480098
```

Note on Response: The numbers following `CardHolderInitiated |00|32104567893210` combines a piped delimited authorization code `|00|` and the network generated numeric transaction identifier. This identifier is card brand specific and, at this time, only Visa returns a network transaction identifier. It is important to return the full `RecurringData` response value in any subsequent COF transaction request.

For Visa, a new `RecurringData` network transaction ID is passed back with each response. This should be saved and used in the subsequent COF transactions. For other card brands allow "CardHolderInitiated " or "CardHolderInitiated |00|".

PaymentsAPI: JSON Remote Procedure Call

Introduction

This section of the guide introduces the integration method direct to our Payments API endpoint implementing JSON/RPC formatted requests and responses. The JSON/RPC examples shown here will illustrate both a plain text (in-scope) example followed by and a combined encryption and tokenization example (reduced scope).

Setup

JSON formatted requests and responses use HTTPS for transaction processing.

Development and Production Servers/Hosts

Development/Cert Server	Primary	https://w1.mercurycert.net/PaymentsAPI/
Note: a primary and secondary server address are supplied for production		
Production Server	Primary	https://w1.mercurypay.com/PaymentsAPI/
Production Server	Secondary	https://w2.mercurypay.com/PaymentsAPI/

Authentication

Authentication occurs via HTTPS Basic Auth using an HTTPS authorization header.

1. Create string of [username]:[password]
 - **Username:** existing MerchantID
 - **Password:** created and stored by Mercury
2. BASE64 encoded string
3. Set Authorization header value to: "Basic [encoded string]"
4. Set Content-Type header value to: "application/json"

POSTS

Transactions are posted to the appropriate development/cert or production server address followed by the transaction type appended to the end of the URL. For example, a credit sale to our cert server would be: <https://w1.mercurycert.net/PaymentsAPI/Credit/Sale>

PaymentsAPI and P2PE Direct Device Integration Option

Because of the direct-to-platform nature of the PaymentsAPI interface, implementing a PCI PTS device which supports MercuryPay's encryption format for card data and PIN encryption requires the developer to complete a direct integration to the hardware manufacture's device. See IDTech, Ingenico, and Verifone developer portals for technical specifications. Accessing and parsing the encrypted output is specific to each device manufacturer. When developing to the PaymentsAPI endpoint the device development work is the responsibility of the developer to complete prior to Worldpay from FIS platform certification.

Once developers have completed the manufacturer's device integration and are able to successfully access and validate the encrypted output, four additional data elements are parsed and inserted into the JSON/RPC transaction request:

- **Encrypted Format:** the MercuryPay platform requires "*MagneSafe*" as the value.
- **AccountSource:** based on input value of *swiped*, *keyed*, and *contactless*
- **EncryptedBlock:** Encrypted *Track2* or manual-entry pseudo *Track* data.
- **EncryptedKey:** Device generated derived unique key.

PaymentsAPI and MercuryPay Tokenization: MToken™ Option

MTokens reduce the scope, risk, cost, and complexity of credit card processing. MTokens are a form of card data encryption generated dynamically upon request from MercuryPay's secure encryption service. This tokenization method is proprietary to the MercuryPay platform and cannot be transferred or otherwise decrypted outside of the secure transaction process. A token is generated (encrypted) by MercuryPay *per transaction* and is returned to the POS in a Base64 encryption format. Worldpay from FIS/MercuryPay is the sole custodian of the process that generates and decrypts a token.

- Tokens are initially generated "by request", using two tags: **RecordNo** and **Frequency**. *RecordNo* is used to request and once generated, pass a token record. *Frequency* refers to the usage intent of the token **OneTime** or **Recurring**.

- The life cycle of tokens is based on how its frequency was originally generated—a *OneTime* token cannot be submitted as a *Recurring* token. Tokens expire based on their frequency.
- *Recurring* tokens were designed to work with weekly, monthly or annual billing.
- *OneTime* tokens should be used for all unscheduled or customer initiated card on file token use.
- MToken was designed to generate dynamic, extended-life tokens with each use. Both token frequencies, when processed, generate a new token, extending the expiration life.
- Token frequency should be determined by usage, not by expiration. A *OneTime* token will expire after 6 months; a *Recurring* token will expire after 24 months.
- Additional Credentials on File/Card on File card brand requirements apply.

Initial Token Request and Subsequent Token Usage

On MercuryPay, token generation is initiated using the **RecordNo** tag. The initial token request value is, "**RecordNumberRequested**"; subsequent use token/**RecordNo** uses the actual value of the full token supplied. Subsequent use of tokens also requires an appended TranCode "**ByRecordNumber**".

- To obtain a token in the initial request, use "**RecordNo**":"**RecordNumberRequested**".
- To submit a token in a subsequent request use "**RecordNo**": **plus** the actual token returned from MercuryPay
- Append the TranCode when using a token to its corresponding **ByRecordNo** equivalent:

Credit TranCodes to Request a Token	Specific "ByRecordNo" TranCode for Subsequent Use of Token
Sale	SaleByRecordNo
Return	ReturnByRecordNo
VoidSale	VoidSaleByRecordNo
VoidReturn	VoidReturnByRecordNo
PreAuth	PreAuthByRecordNo
PreAuthCapture	PreAuthCaptureByRecordNo
Adjust	AdjustByRecordNo

PaymentsAPI Note If you receive an error message, "Token Invalid", check to make sure that the *RecordNo* is not being urlencoded in an unexpected manner.

JSON/RPC Credit Transactions Example Requests and Responses

Credit card information (card number and expiration date) can be sent by swiping the card through a standard or encrypted magnetic card reader (**swipe**) or manually entered (**keyed**). A JSON/RPC EMV option is unavailable. Quick key to the JSON/RPC examples below:

- The examples below include a clear-text example followed by an encryption and tokenization example.
- A subsequent use tokenization example is also included.
- Bold fields are required. Gray fields are optional.
- Values used in the examples below is intended as guidance for development only

/Credit/Sale – Clear-Text

A Credit **Sale** is used to process a credit card purchase (VISA, MasterCard, American Express, Discover, Carte Blanche, Diner's Club, JCB or Other) in one step.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/Sale>

Request: Swipe


```
{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "Track2Data",
  "Purchase": "1.00",
  "Name": "MPS TEST",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Request: Keyed (manual)

Substitute the **Track2** data with the hand-keyed **AcctNo** and **ExpDate**, and add **Address**, **Zip** and **CVVData**:

```
"AcctNo": "AccountNo",
"ExpDate": "0517",
"Address": "4 Corporate Square",
"Zip": "30329",
"CVVData": "880",
```

Response: Swiped

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "595901",
  "AcctNo": "400300XXXXXX6781",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "Sale",
  "AuthCode": "08704A",
  "CaptureStatus": "Captured",
  "RefNo": "1002",
  "InvoiceNo": "1",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "AcqRefData": "b8238953#20#20#20#20#20#20#20e00",
  "ProcessData": "|00|210100200000",
}
```

Response: Keyed (Manual)

The response will include two additional tags for the CVV and AVS results:

```
"CVVResult": "M",
"AVSResult": "A";
```

/Credit/Sale – Encryption and Tokenization**Initial Token Request**

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/Sale>

Request: Swiped

```
{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Frequency": "OneTime",
```

```

    "RecordNo": "RecordNumberRequested",
    "EncryptedFormat": "MagneSafe",
    "AccountSource": "Swiped",
    "EncryptedBlock": "2F8248964608156B2B1745287B44CA90A349905F905514ABE3979D7957F13804705684B1C9D5641C",
    "EncryptedKey": "9500030000040C200026",
    "TerminalName": "MPS Terminal",
    "ShiftID": "MPS Shift",
    "OperatorID": "MPS Operator"
  }

```

Request: Keyed (manual)

Substitute the **AccountSource** as follows, and add **Address**, **Zip**, and **CVVData**:

```

"AccountSource": "Keyed",
"Address": "4 Corporate Square",
"Zip": "30329",
"CVVData": "880",

```

Response: Swiped

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400555XXXXXX0480",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "Sale",
  "AuthCode": "VI0100",
  "CaptureStatus": "Captured",
  "RefNo": "0287",
  "InvoiceNo": "101",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "AcqRefData": "b8238953#20#20#20#20#20#20#20e00",
  "RecordNo": "/0/IFNXI7ISBDAPXxbM2K7h5uJ0uevtAwNXedIliMCUiEgUQBSIQa1Y5",
  "ProcessData": "|00|210100200000"
}

```

Response: Keyed (Manual)

The response will include two additional tags for the CVV and AVS results:

```

"CVVResult": "M",
"AVSResult": "A",

```

/Credit/SaleByRecordNo – Subsequent Use of the Token

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/SaleByRecordNo>
Request

```

{
  "InvoiceNo": "102",
  "RefNo": "102",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Frequency": "OneTime",
  "RecordNo": "cYiaGv9ZkPtXbq3uaLtxNoksqkz6DK+3KSBLJtuonxkiEgUQBCIQGqBR",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}

```

```

}
Response
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400555XXXXXX0480",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "Sale",
  "AuthCode": "VI0100",
  "CaptureStatus": "Captured",
  "RefNo": "0288",
  "InvoiceNo": "102",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "AcqRefData": "K b8238953#20#20#20#20#20#20e00",
  "RecordNo": "Qb9h1zE42TB7rPrvXj2TDZgdge+a0eGdZeTp+nshiwMiEgUQB SIQA1gx",
  "ProcessData": "|00|210100700000"
}

```

/Credit/ZeroAuth – Clear-Text

The **ZeroAuth** transaction enables merchants to validate a credit card without charging an amount to the cardholder's account. It can also be used to request a token.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/ZeroAuth>

Request: Swipe

```

{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "Track2Data",
  "Name": "MPS TEST",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}

```

Request: Keyed (manual)

Substitute the **Track2** data with the hand-keyed **AcctNo** and **ExpDate**, and add **Address**, **Zip** and **CVVData**:

```

  "AcctNo": "AccountNo",
  "ExpDate": "0517",
  "Address": "4 Corporate Square",
  "Zip": "30329",
  "CVVData": "880",

```

Response: Swiped

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "595901",
  "AcctNo": "400300XXXXXX6781",

```

```

"ExpDate": "XXXX",
"CardType": "VISA",
"TranCode": "ZeroAuth",
"AuthCode": "08704A",
"CaptureStatus": "Captured",
"RefNo": "1002",
"InvoiceNo": "1",
"OperatorID": "MPS Operator",
"Memo": "MPS Example JSON v1.0",
"Purchase": "1.00",
"Authorize": "1.00",
"AcqRefData": "b8238953#20#20#20#20#20#20#20#20e00",
"ProcessData": "|00|210100200000",
}

```

Response: Keyed (Manual)

The response will include two additional tags for the CVV and AVS results:

```

"CVVResult": "M",
"AVSResult": "A",

```

/Credit/ZeroAuth – Encryption and Tokenization

The **ZeroAuth** transaction enables merchants to validate a credit card without charging an amount to the cardholder's account. It can also be used to request a token.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/ZeroAuth>

Request: Swiped

```

{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Frequency": "OneTime",
  "RecordNo": "RecordNumberRequested",
  "EncryptedFormat": "MagneSafe",
  "AccountSource": "Swiped",
  "EncryptedBlock": "2F8248964608156B2B1745287B44CA90A349905F905514ABE3979D7957F13804705684B1C9D5641C",
  "EncryptedKey": "9500030000040C200026",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}

```

Request: Keyed (manual)

Substitute the **AccountSource** as follows, and add **Address**, **Zip**, and **CVVData**:

```

"AccountSource": "Keyed",
"Address": "4 Corporate Square",
"Zip": "30329",
"CVVData": "880",

```

Response: Swiped

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400555XXXXXX0480",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "ZeroAuth",

```

```

"AuthCode": "VI0100",
"CaptureStatus": "Captured",
"RefNo": "0287",
"InvoiceNo": "101",
"OperatorID": "MPS Operator",
"Memo": "MPS Example JSON v1.0",
"Purchase": "0.00",
"Authorize": "0.00",
"AcqRefData": "b8238953#20#20#20#20#20#20#20#20e00",
"RecordNo": "0|IFNXI7ISBDAPXxbM2K7h5uJ0uevtAwNXedIliMCUIEgUQB SIQA1Y5",
"ProcessData": "00|210100200000"
}

```

Response: Keyed (Manual)

The response will include two additional tags for the CVV and AVS results:

```

"CVVResult": "M",
"AVSResult": "A"

```

Credit Reversal – Clear-text

"Reversal" as it is used on MercuryPay, refers to an enhanced VoidSale transaction which *communicates directly with the card Issuing bank* in order to reverse the transaction and release the funds back to the cardholder's available balance. This is the preferred method for cancelling a transaction within the live batch. Reversals generally take less time (from 2 to 72 hours) to be removed from the cardholder's banking statement. Reversals may be used to reverse an existing *Credit Sale, VoiceAuth, PreAuthCapture* and *PreAuth*.

Follow the steps below to process a *Reversal* request:

1. The original transactions must "allow" the reversal.
 - This done by including the **PartialAuth=Allow** tag in the request.
 - *Transactions processed without the PartialAuth=Allow tag cannot be reversed.*
2. The request must include *TranCode* data elements of a **VoidSale** (*AcctNo, Exp Date, InvoiceNo, RefNo, AuthCode, Purchase* amount along with **AcqRefData** and **ProcessData**. This sends the request to the Issuing bank.
3. Finally, confirm that a *CmdStatus=Approved* and *TextResponse=REVERSED* message is returned; if it is not returned a standard *VoidSale* should be immediately sent without the *AcqRefData* and *ProcessData* elements.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/VoidSale>

Request: Swiped

```

{
  "InvoiceNo": "1",
  "RefNo": "1040",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "Track2Data",
  "Name": "MPS TEST",
  "Purchase": "1.00",
  "AuthCode": "06187A",
  "AcqRefData": "b8238953#20#20#20#20#20#20#20#20e00"
  "ProcessData": "00|201000200000"
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}

```

Request: Keyed (Manual)

Substitute the **Track2** data with the hand-keyed account number and expiration date as follows:

```

"AcctNo": "AccountNo",
"ExpDate": "0517",

```


Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "REVERSED",
  "UserTraceData": "",
  "MerchantID": "595901",
  "AcctNo": "400300XXXXXX6781",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "Void Sale",
  "AuthCode": "Voided",
  "RefNo": "0001",
  "InvoiceNo": "1",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "AcqRefData": "b8238953#20#20#20#20#20#20#20e00",
  "ProcessData": "|A4|210100200000",
}
```

Credit Reversal – Tokenization

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/VoidSaleByRecordNo>
Request

```
{
  "InvoiceNo": "102",
  "RefNo": "0288",
  "Memo": "MPS Example JSON v1.0",
  "Name": "MPS TEST",
  "Purchase": "1.00",
  "AuthCode": "VI0100",
  "Frequency": "OneTime",
  "RecordNo": "cYiaGv9ZkPtXbq3uaLtxNoksqkz6DK+3KSBLJtuonxkiEgUQBCIQGqBR",
  "AcqRefData": "b8238953#20#20#20#20#20#20#20e00",
  "ProcessData": "00|201000200000",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "REVERSED",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400555XXXXXX0480",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "Void Sale",
  "AuthCode": "VOIDED",
  "CaptureStatus": "Captured",
  "RefNo": "0288",
  "InvoiceNo": "102",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
}
```

```

    "Purchase": "1.00",
    "Authorize": "1.00",
    "AcqRefData": "K",
    "RecordNo": "dEkwbKVu/NtipLtDMzFKKxlzoVcqM/FTvCmofUR0MpliEgUQBISQA1hD",
    "ProcessData": "|A4|210100600000"
  }

```

/Credit/VoidSaleByRecordNo – Clear-Text (not a reversal)

A **VoidSale** cancels a previously completed *Credit Sale*, *VoiceAuth*, *PreAuthCapture* or *Return* transaction that is logged with a reference number in the merchant's current open batch. A *VoidSale* communicates with the **processor** and may take several business days before the sale is removed from the cardholder's bank account. *VoidSales* require the reference number (*RefNo*) and *AuthCode* from the original transaction, as well as *InvoiceNo*, the full card number, expiration date and authorized purchase amount.

Note Use the **RefNo** and **AuthCode** returned in the *Sale Response*

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/VoidSale>
Request: Swiped

```

{
  "InvoiceNo": "1",
  "RefNo": "1040",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "Track2Data",
  "Name": "MPS TEST",
  "Purchase": "1.00",
  "AuthCode": "06187A",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}

```

Request: Keyed (Manual)

Substitute the **Track2** data with the hand-keyed account number and expiration date as follows:

```

  "AcctNo": "AccountNo",
  "ExpDate": "0517",

```

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "595901",
  "AcctNo": "400300XXXXXX6781",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "Void Sale",
  "AuthCode": "Voided",
  "RefNo": "0001",
  "InvoiceNo": "1",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "AcqRefData": "aY",
  "ProcessData": "|A4|210100200000",
}

```

/Credit/VoidSaleByRecordNo – Tokenization

A **VoidSale** cancels a previously completed Credit Sale, *VoiceAuth*, *PreAuthCapture* or *Return* transaction that is logged with a reference number in the merchant's current open batch. A *VoidSale* communicates with the **processor** and may take several business days before the sale is removed from the cardholder's bank account. *VoidSales* require the reference number (*RefNo*) and *AuthCode* from the original transaction, as well as *InvoiceNo*, the full card number, expiration date and authorized purchase amount.

Note Use the **RefNo** and **AuthCode** returned in the *Sale Response*

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/VoidSaleByRecordNo>
Request

```
{
  "InvoiceNo": "102",
  "RefNo": "0288",
  "Memo": "MPS Example JSON v1.0",
  "Name": "MPS TEST",
  "Purchase": "1.00",
  "AuthCode": "VI0100",
  "Frequency": "OneTime",
  "RecordNo": "cYiaGv9ZkPtXbq3uaLtxNoksqkz6DK+3KSBLJtuonxkiEgUQBCIQGqBR",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400555XXXXXX0480",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "Void Sale",
  "AuthCode": "VOIDED",
  "CaptureStatus": "Captured",
  "RefNo": "0288",
  "InvoiceNo": "102",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "AcqRefData": "K",
  "RecordNo": "dEkwbKVu/NtipLtDMzFKKxlzoVcqM/FTvCmofUR0MpliEgUQBISQA1hD",
  "ProcessData": "|A4|210100600000"
}
```

/Credit/Return – Clear-Text

A **Return** (refund) is the opposite of a *Sale*. It is an independent, non-referenced transaction used to credit a cardholder's account. **DO NOT USE a Return to void a transaction in the current open batch.**

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/Return>
Request: Swiped

```
{
  "InvoiceNo": "1",
```

```

"RefNo": "1",
"Memo": "MPS Example JSON v1.0",
"Track2": "Track2Data",
"Name": "MPS TEST",
"Purchase": "1.00",
"TerminalName": "MPS Terminal",
"ShiftID": "MPS Shift",
"OperatorID": "MPS Operator"
}

```

Request: Keyed (manual)

Substitute the **Track2** data with the hand-keyed account number and expiration date as follows:

```

"AcctNo": "AccountNo",
"ExpDate": "0517",

```

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "595901",
  "AcctNo": "400300XXXXXX6781",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "Return",
  "AuthCode": "029720",
  "CaptureStatus": "Captured",
  "RefNo": "1001",
  "InvoiceNo": "1",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "ProcessData": "|20|210100200000",
}

```

/Credit/Return – Encryption and Tokenization

A **Return** (refund) is the opposite of a *Sale*. It is an independent, non-referenced transaction used to credit a cardholder's account. **DO NOT USE a Return to void a transaction in the current open batch.**

Initial Token Request

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/Return>

Request: Swiped

```

{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Frequency": "OneTime",
  "RecordNo": "RecordNumberRequested",
  "EncryptedFormat": "MagneSafe",
  "AccountSource": "Swiped",
  "EncryptedBlock": "2F8248964608156B2B1745287B44CA90A349905F905514ABE3979D7957F13804705684B1C9D5641C",
  "EncryptedKey": "9500030000040C200026",
  "Name": "MPS TEST",
  "Purchase": "1.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
}

```

```
"OperatorID": "MPS Operator"
}
```

Request: Keyed (manual)

Substitute the **AccountSource** as follows:

```
"AccountSource": "Keyed",
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400555XXXXXX0480",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "Return",
  "AuthCode": "000013",
  "CaptureStatus": "Captured",
  "RefNo": "0289",
  "InvoiceNo": "1",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "AcqRefData": "aN",
  "RecordNo": "CAYIFeqBAGHy8rHNsHg3lAdizYY3IT0Hq1Rj8tYD20QiEgUQBSIQA1hs",
  "ProcessData": "|20|210100200000"
}
```

/Credit/ReturnByRecordNo**Subsequent Use of the Token**

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/ReturnByRecordNo>

Request

```
{
  "InvoiceNo": "104",
  "RefNo": "104",
  "Memo": "MPS Example JSON v1.0",
  "Frequency": "OneTime",
  "RecordNo": "CAYIFeqBAGHy8rHNsHg3lAdizYY3IT0Hq1Rj8tYD20QiEgUQBSIQA1hs",
  "Name": "MPS TEST",
  "Purchase": "1.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400555XXXXXX0480",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "Return",

```

```

    "AuthCode": "000013",
    "CaptureStatus": "Captured",
    "RefNo": "0290",
    "InvoiceNo": "104",
    "OperatorID": "MPS Operator",
    "Memo": "MPS Example JSON v1.0",
    "Purchase": "1.00",
    "Authorize": "1.00",
    "AcqRefData": "KaN",
    "RecordNo": "CTBawIWcMie1j/tA9DRuabyvBgSPPCi87Wlcwj9VteAiEgUQBSIQA1il",
    "ProcessData": "|20|210100700000"
  }

```

/Credit/VoidReturn – Clear-Text

Once captured, use **VoidReturn** to void the original *Return* transaction.

Note Use the **RefNo** and **AuthCode** returned in the *Return* Response

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/VoidReturn>
Request: Swiped

```

{
  "InvoiceNo": "1",
  "RefNo": "1001",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "Track2Data",
  "Name": "MPS TEST",
  "Purchase": "1.00",
  "AuthCode": "029720",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}

```

Request: Keyed (Manual)

Substitute the **Track2** data with the hand-keyed account number and expiration date as follows:

```

    "AcctNo": "AccountNo",
    "ExpDate": "0517",

```

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturn Code": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "595901",
  "AcctNo": "400300XXXXXX6781",
  "ExpDate": "XXXX",
  "Card Type": "VISA",
  "TranCode": "VoidReturn",
  "AuthCode": "VOIDED",
  "CaptureStatus": "Captured",
  "RefNo": "1001",
  "InvoiceNo": "1",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "ProcessData": "|A4|210100200000",
}

```


/Credit/VoidReturnByRecordNo – Tokenization

Once captured, use **VoidReturn** to void the original *Return* transaction.

Note Use the **RefNo** and **AuthCode** returned in the *Return* Response

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/VoidReturnByRecordNo>
Request

```
{
  "InvoiceNo": "104",
  "RefNo": "0290",
  "Memo": "MPS Example JSON v1.0",
  "Name": "MPS TEST",
  "Purchase": "1.00",
  "AuthCode": "000013",
  "Frequency": "OneTime",
  "RecordNo": "CTBawIWcMie1j/tA9DRuabyvBgPPCi87WlCwj9VteAiEgUQBSiQA1il",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400555XXXXXX0480",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "VoidReturn",
  "AuthCode": "VOIDED",
  "CaptureStatus": "Captured",
  "RefNo": "0290",
  "InvoiceNo": "104",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "AcqRefData": "K",
  "RecordNo": "2vgM298NosWSZi4V5X+UDEJL5XtrtGbF/M/b5Z+qoI0iEgUQBSiQA1in",
  "ProcessData": "|A4|210100600000"
}
```

/Credit/PreAuth – Clear-Text

This **TranCode** is used when the credit purchase occurs in two steps, authorization (*PreAuth*) and capture (*PreAuthCapture*). Used when the exact final purchase is unknown or requires modification upon capture – such as in a restaurant, card-facing environment; also be implemented by eCommerce merchants when ship-date is more than 24 hours after the online order. The **PreAuth** is used to pre-authorize a "pending" amount on a credit card. The transaction must be finalized using a *PreAuthCapture*.

Note **Keyed** credit transactions should include two additional tags for AVS and CVV validation.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/PreAuth>
Request: Swiped

```
{
  "InvoiceNo": "1",
  "RefNo": "1",

```

```

    "Memo": "MPS Example JSON v1.0",
    "Track2": "Track2Data",
    "Name": "MPS TEST",
    "Purchase": "1.00",
    "Authorize": "1.00",
    "TerminalName": "MPS Terminal",
    "ShiftID": "MPS Shift",
    "OperatorID": "MPS Operator"
  }

```

Request: Keyed (Manual)

Substitute the **Track2** data with the hand-keyed **AcctNo** and **ExpDate** expiration date as follows, and add **Address**, **Zip** and **CVVData**:

```

    "AcctNo": "AccountNo",
    "ExpDate": "0517",
    "Address": "4 Corporate Square",
    "Zip": "30329",
    "CVVData": "880"

```

Response: Swiped

```

{
  ResponseOrigin: "Processor",
  DSIXReturnCode: "000000",
  CmdStatus: "Approved",
  TextResponse: "AP",
  UserTraceData: "",
  MerchantID: "595901",
  AcctNo: "400300XXXXXX6781",
  ExpDate: "XXXX",
  CardType: "VISA",
  TranCode: "PreAuth",
  AuthCode: "05764A",
  InvoiceNo: "1",
  OperatorID: "MPS Operator",
  Memo: "MPS Example JSON v1.0",
  Purchase: "1.00",
  Authorize: "1.00",
  AcqRefData: "b8238953#20#20#20#20#20#20#20e00",
  ProcessData: "|14|210100200000",
}

```

Response: Keyed (Manual)

The response will include two additional tags for the CVV and AVS results:

```

    "CVVResult": "M",
    "AVSResult": "A"

```

/Credit/PreAuth – Encryption and Tokenization

This TranCode is used when the credit purchase occurs in two steps, authorization (*PreAuth*) and capture (*PreAuthCapture*). Used when the exact final purchase is not known or requires modification upon capture – such as in a restaurant, card-facing environment. May also be implemented by eCommerce merchants when ship-date is more than 24 hours after the online order. The **PreAuth** is used to pre-authorize an amount on a credit card. The transaction is finalized using a *PreAuthCapture*.

Note **Keyed** credit transactions should include two additional tags for AVS and CVV validation.

Initial Token Request

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/PreAuth>

Request: Swiped

```

{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",

```

```

    "Frequency": "OneTime",
    "RecordNo": "RecordNumberRequested",
    "EncryptedFormat": "MagneSafe",
    "AccountSource": "Swiped",
    "EncryptedBlock": "2F8248964608156B2B1745287B44CA90A349905F905514ABE3979D7957F13804705684B1C9
D5641C",
    "EncryptedKey": "9500030000040C200026",
    "Name": "MPS TEST",
    "Purchase": "1.00",
    "Authorize": "1.00",
    "TerminalName": "MPS Terminal",
    "ShiftID": "MPS Shift",
    "OperatorID": "MPS Operator"
  }

```

Request: Keyed (Manual)

Substitute the **AccountSource** as follows, and add **Address**, **Zip**, and **CVVData**:

```

"AccountSource": "Keyed",
"Address": "4 Corporate Square",
"Zip": "30329",
"CVVData": "880",

```

Response: Swiped

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400555XXXXXX0480",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "PreAuth",
  "AuthCode": "VI0100",
  "InvoiceNo": "1",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "AcqRefData": " b8238953#20#20#20#20#20#20#20e00",
  "RecordNo": "6Lze0P0u3tRiVdUyGYbRIfv0AQ1p8T+J96YHzoG0kbUiEgUQBSIQA1oU",
  "ProcessData": "|14|210100200000"
}

```

Response: Keyed (manual)

The response will include two additional tags for the CVV and AVS results:

```

"CVVResult": "M",
"AVSResult": "A",

```

/Credit/PreAuthByRecordNo – Tokenization**Subsequent Use of the Token**

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/PreAuthByRecordNo>

Request

```

{
  "InvoiceNo": "101",
  "RefNo": "101",
  "Memo": "MPS Example JSON v1.0",
  "Frequency": "OneTime",
  "RecordNo": "6Lze0P0u3tRiVdUyGYbRIfv0AQ1p8T+J96YHzoG0kbUiEgUQBSIQA1oU",
  "Name": "MPS TEST",
  "Purchase": "1.00",

```

```
"Authorize": "1.00",
"TerminalName": "MPS Terminal",
"ShiftID": "MPS Shift",
"OperatorID": "MPS Operator"
}
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400555XXXXXX0480",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "PreAuth",
  "AuthCode": "VI0100",
  "InvoiceNo": "101",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "AcqRefData": "b8238953#20#20#20#20#20#20#20#20e00",
  "RecordNo": "sUmuOR2axsSufz3g+Wm4bHABNQ5gDRaZSGvce+6P4KAiEgUQB SIQA1om",
  "ProcessData": "|14|210100700000"
}
```

/Credit/PreAuthCapture – Clear-Text

PreAuthCapture is used to finalize a *PreAuth* transaction so that settlement can occur. Use in restaurant environment for tip modifications. The tip amount is specified with **Gratuity** tags and the original check amount is specified with **Purchase** tags.

Note Never store *Track2* data. *PreAuthCapture* uses **AcctNo** and **ExpDate**.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/PreAuthCapture>

Request

```
{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "AcctNo": "AccountNo",
  "ExpDate": "0517",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "Gratuity": "0.20",
  "AuthCode": "05764A",
  "AcqRefData": "b8238953#20#20#20#20#20#20#20#20e00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "595901",

```

```

"AcctNo": "400300XXXXXX6781",
"ExpDate": "XXXX",
"CardType": "VISA",
"TranCode": "PreAuthCapture",
"AuthCode": "05764A",
"CaptureStatus": "Captured",
"RefNo": "1002",
"InvoiceNo": "1",
"OperatorID": "MPS Operator",
"Memo": "MPS Example JSON v1.0",
"Purchase": "1.00",
"Authorize": "1.20",
"Gratuity": "0.20",
"AcqRefData": "b8238953#20#20#20#20#20#20#20#20e00",
"ProcessData": "|15|210100200000",
}

```

/Credit/PreAuthCaptureByRecordNo -- Tokenization

PreAuthCapture is used to finalize a *PreAuth* transaction so that settlement can occur. Use in restaurant environment for tip modifications. The tip amount is specified with **Gratuity** tags and the original check amount is specified with **Purchase** tags.

Note Never store *Track2* data. With Tokenization, *PreAuthCapture* uses **AcctNo** and **ExpDate** built within the **RecordNo**.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/PreAuthCaptureByRecordNo>
Request

```

{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Frequency": "OneTime",
  "RecordNo": "sUmuOR2axsSufz3g+Wm4bHABNQ5gDRaZSGvce+6P4KAiEgUQBISQA1om",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "Gratuity": "0.20",
  "AuthCode": "VI0100",
  "AcqRefData": "Kb8238953#20#20#20#20#20#20#20#20e00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}

```

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400555XXXXXX0480",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "PreAuthCapture",
  "AuthCode": "VI0100",
  "CaptureStatus": "Captured",
  "RefNo": "0291",
  "InvoiceNo": "1",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
}

```


/Credit/VoiceAuth – Encryption and Tokenization

A **VoiceAuth** (also referred to as a **Force**), is a credit purchase request sent through the POS when a merchant is directed to call the credit card company to obtain a verbal authorization code. This **AuthCode** must be included in the transaction request so the transaction can be settled and paid to the merchant.

Initial Token Request

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/VoiceAuth>

Request: Swiped

```
{
  "InvoiceNo": "105",
  "RefNo": "105",
  "Memo": "MPS Example JSON v1.0",
  "Frequency": "OneTime",
  "RecordNo": "RecordNumberRequested",
  "EncryptedFormat": "MagneSafe",
  "AccountSource": "Swiped",
  "EncryptedBlock": "2F8248964608156B2B1745287B44CA90A349905F905514ABE3979D7957F13804705684B1C9D5641C",
  "EncryptedKey": "9500030000040C200026",
  "Name": "MPS TEST",
  "Purchase": "1.00",
  "AuthCode": "123456",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Request: Keyed (Manual)

Substitute the **AccountSource** as follows:

"AccountSource": "Keyed",

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400555XXXXXX0480",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "VoiceAuth",
  "AuthCode": "123456",
  "CaptureStatus": "Captured",
  "RefNo": "0292",
  "InvoiceNo": "105",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "AcqRefData": "aN",
  "RecordNo": "DHSQSw2JhjGN4bly2yR5dr+bf9Ua0r0712EmoeDGpqAiEgUQBSIQA1ph",
  "ProcessData": "|18|210100200000"
}
```

/Credit/Adjust – Clear-Text

Adjust is used to change the *Amount* value for a *Sale* transaction already captured in the current open batch.

Note Use the **AuthCode** and **RefNo** from the Response in the original purchase transaction.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/Adjust>

Request: Swiped

```
{
  "InvoiceNo": "1",
  "RefNo": "1002",
  "Memo": "MPS Example JSON v1.0v",
  "Track2": "Track2Data",
  "Name": "MPS TEST",
  "Purchase": "1.00",
  "Gratuity": "0.20",
  "AuthCode": "08704A",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Request: Keyed (Manual)

Substitute the **Track2** data with the hand-keyed account number and expiration date as follows:

```
"AcctNo": "AccountNo",
"ExpDate": "0517",
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "595901",
  "AcctNo": "400300XXXXXX6781",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "Adjust",
  "AuthCode": "05764A",
  "CaptureStatus": "Captured",
  "RefNo": "1002",
  "InvoiceNo": "1",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.00",
  "ProcessData": "|02|210100200000",
}
```

/Credit/AdjustByRecordNo – Tokenization

Adjust is used to change the *Amount* value for a *Sale* transaction already captured in the current open batch.

Note Use the **AuthCode** and **RefNo** from the Response in the original purchase transaction.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Credit/AdjustByRecordNo>

Request

```
{
  "InvoiceNo": "105",
  "RefNo": "0292",
```

```

"Memo": "MPS Example JSON v1.0",
"Frequency": "OneTime",
"RecordNo": "DHSQSw2JhjGN4bly2yR5dr+bf9Ua0r0712EmoeDGpqAiEgUQBISQA1ph",
"Name": "MPS TEST",
"Purchase": "1.00",
"Gratuity": "0.20",
"AuthCode": "123456",
"TerminalName": "MPS Terminal",
"ShiftID": "MPS Shift",
"OperatorID": "MPS Operator"
}

```

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400555XXXXXX0480",
  "ExpDate": "XXXX",
  "CardType": "VISA",
  "TranCode": "Adjust",
  "AuthCode": "123456",
  "CaptureStatus": "Captured",
  "RefNo": "0292",
  "InvoiceNo": "105",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.20",
  "Gratuity": "0.20",
  "RecordNo": "UQwjFuzU0sJRRjdzLtdaS+4uYnR3otK1CdVZdg7M8ngiEgUQBISQA1ql",
  "ProcessData": "|02|210100600000"
}

```

JSON/RPC Debit Transactions

PIN Encryption Note: Because of the direct-to-platform nature of the PaymentsAPI interface, implementing a PCI PTS device which supports MercuryPay's encryption format for card data and PIN encryption requires the developer to complete a direct integration to the hardware manufacture's device. See IDTech, Ingenico, and Verifone developer portals for technical specifications. Accessing and parsing the encrypted output is specific to each device manufacturer. When developing to the PaymentsAPI endpoint the device development work is the responsibility of the developer to complete prior to Worldpay from FIS platform certification.

Debit transactions refer to the use of a Debit Card that is swiped through a magnetic card reader.

- Within the JSON/RPC PaymentsAPI format, Debit transactions are swiped only and require the input of a PIN by the cardholder into a PTS confirmed PINpad device. Where the PIN is always required to be encrypted by the PINPad, it is recommended to also implement a PTS confirmed Encryption device for card data.
- The PINPad must always encrypt the PIN number for secure transmission to the bank's computer for validation.
- The PIN is captured within the **PINBlock** tag along with the Key identifier, **DervdKey** used to encrypt the PIN.

/Debit/Sale – Clear-Text

A **Debit Sale** is a real-time debiting of the cardholder's account.

- Debit sales cannot be voided or reversed; they can only be returned.
- The *Sale* may also include cash-back amounts as a sub-amount of the total sale.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Debit/Sale>

Request

```
{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "Track2Data",
  "Name": "MPS TEST",
  "Purchase": "1.00",
  "CashBack": "0.50",
  "PINBlock": "254B0613851AFD36",
  "DervdKey": "3D01000007C00041",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturn Code": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "AP*",
  "UserTraceData": "",
  "MerchantID": "595901",
  "AcctNo": "400300XXXXXX6781",
  "Exp Date": "XXXX",
  "Card Type": "DEBIT",
  "Tran Code": "Sale",
  "CaptureStatus": "Captured",
  "RefNo": "0024",
  "InvoiceNo": "1",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.50",
  "CashBack": "0.50"
}
```

/Debit/Sale – Encryption

A **Debit Sale** is a real-time debiting of the cardholder's account.

- Debit sales cannot be voided or reversed; they can only be returned.
- The *Sale* may also include cash-back amounts as a sub-amount of the total sale.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Debit/Sale>

Request

```
{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Frequency": "OneTime",
  "RecordNo": "RecordNumberRequested",
  "EncryptedFormat": "MagneSafe",
  "AccountSource": "Swiped",
}
```

```

    "EncryptedBlock": "005234EAD7AD5A279B8FB947BC15A123FFD775D03386B54F6B37CB54AB33106FFC105588E28B0799",
    "EncryptedKey": "9500030000040C20001D",
    "Name": "MPS TEST",
    "Purchase": "1.00",
    "CashBack": "0.50",
    "PINBlock": "0A04E4C40F6A41B2",
    "DervdKey": "3D0100000B200007",
    "TerminalName": "MPS Terminal",
    "ShiftID": "MPS Shift",
    "OperatorID": "MPS Operator"
  }

```

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "APPROVED",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400300XXXXXX6781",
  "ExpDate": "XXXX",
  "CardType": "DEBIT",
  "TranCode": "Sale",
  "CaptureStatus": "Captured",
  "RefNo": "0293",
  "InvoiceNo": "1",
  "OperatorID": "MPS Operator",
  "Memo": "MPS Example JSON v1.0",
  "Purchase": "1.00",
  "Authorize": "1.50",
  "CashBack": "0.50"
}

```

/Debit/Return – Clear-Text

A Debit **Return** is a real-time crediting of the cardholder's account. It is an independent transaction, and is not dependent on referencing a previously processed Debit Sale.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Debit/Return>

Request

```

{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "Track2Data",
  "Name": "MPS TEST",
  "Purchase": "1.00",
  "CashBack": "0.50",
  "PINBlock": "760246866109281B",
  "DervdKey": "3D01000007C0003F",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}

```

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",

```

```

"TextResponse": "APPROVED      ",
"UserTraceData": "",
"MerchantID": "595901",
"AcctNo": "400300XXXXXX6781",
"ExpDate": "XXXX",
"CardType": "DEBIT",
"TranCode": "Return",
"CaptureStatus": "Captured",
"RefNo": "0025",
"InvoiceNo": "1",
"OperatorID": "MPS Operator",
"Memo": "MPS Example JSON v1.0",
"Purchase": "1.00",
"Authorize": "1.50",
"CashBack": "0.50"
}

```

/Debit/Return – Encryption

A Debit **Return** is a real-time crediting of the cardholder's account. It is an independent transaction, and is not dependent on referencing a previously processed Debit *Sale*.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Debit/Return>
Request

```

{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Frequency": "OneTime",
  "RecordNo": "aAb3122sd234sdrf1231dfsd123123sdfsdrf21341",
  "EncryptedFormat": "MagneSafe",
  "AccountSource": "Swiped",
  "EncryptedBlock": "005234EAD7AD5A279B8FB947BC15A123FFD775D03386B54F6B37CB54AB33106FFFC105588E28B0799",
  "EncryptedKey": "9500030000040C20001D ",
  "Name": "MPS TEST",
  "Purchase": "1.00",
  "CashBack": "0.50",
  "PINBlock": "0A04E4C40F6A41B2",
  "DervdKey": "3D0100000B200007",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}

```

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "APPROVED      ",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "AcctNo": "400300XXXXXX6781",
  "ExpDate": "XXXX",
  "CardType": "DEBIT",
  "TranCode": "Return",
  "CaptureStatus": "Captured",
  "RefNo": "0294",
  "InvoiceNo": "1",
  "OperatorID": "MPS Operator",
}

```



```

    "Memo": "MPS Example JSON v1.0",
    "Purchase": "1.00",
    "Authorize": "1.50",
    "CashBack": "0.50"
  }

```

PaymentsAPI Settlement Admin Functions

MercuryPay supports the ability for either a *Time Initiated* settlement option that closes the merchant batch file on behalf of the merchant automatically, or a *Merchant Initiated* settlement option that allows the merchant to close the batch locally at any time. Both are also supported using JSON/RPC formatted transactions:

/Admin/BatchSummary

BatchSummary is used to get a batch totals amount and count summary of the information in the batch.

- A *BatchSummary* request must be performed prior to a **BatchClose** request in order to obtain the host record.
- All relevant response fields in the *BatchSummary* should be supplied in the *BatchClose* as required.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Admin/BatchSummary>
Request

```

{
  "Memo": "MPS Example JSON v1.0",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}

```

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Success",
  "TextResponse": "OK",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "OperatorID": "MPS Operator",
  "BatchNo": "3247",
  "BatchItemCount": "253",
  "NetBatchTotal": "21008.19",
  "CreditPurchaseCount": "231",
  "CreditPurchaseAmount": "21017.64",
  "CreditReturnCount": "11",
  "CreditReturnAmount": "53.53",
  "DebitPurchaseCount": "9",
  "DebitPurchaseAmount": "53.81",
  "DebitReturnCount": "2",
  "DebitReturnAmount": "9.73"
}

```

/Admin/BatchClear

BatchClear is rarely used and was designed to be implemented in the event that there is a need to clear all transactions in the merchant's currently open batch.

- This function was initially designed for the Batch Capture close method payment applications.
- **CAUTION!** Clearing the batch erases all transaction information.

- Before ever processing a *BatchClear*, make sure that the batch is settled successfully.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Admin/BatchClear>

Request

```
{
  "Memo": "MPS Example JSON v1.0",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Success",
  "TextResponse": "OK TEST",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "OperatorID": "MPS Operator",
  "BatchItemCount": "11",
  "NetBatchTotal": "44.08",
  "ControlNo": "0000"
}
```

/Admin/BatchClose

BatchClose is used to close the currently open batch.

- All values in the *BatchClose* must match those returned from the *BatchSummary* response.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/Admin/BatchClose>

Request

```
{
  "Memo": "MPS Example JSON v1.0",
  "BatchNo": "0221",
  "BatchItemCount": "3",
  "NetBatchTotal": "3.20",
  "CreditPurchaseCount": "3",
  "CreditPurchaseAmount": "3.20",
  "CreditReturnCount": "0",
  "CreditReturnAmount": "0.00",
  "DebitPurchaseCount": "0",
  "DebitPurchaseAmount": "0.00",
  "DebitReturnCount": "0",
  "DebitReturnAmount": "0.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Success",
  "TextResponse": "OK",
  "UserTraceData": "",
  "MerchantID": "395347308",
  "OperatorID": "MPS Operator",
  "BatchNo": "0221",
  "BatchItemCount": "3",
  "NetBatchTotal": "3.20",
}
```

```

"CreditPurchaseCount": "3",
"CreditPurchaseAmount": "3.20",
"CreditReturnCount": "0",
"CreditReturnAmount": "0.00",
"DebitPurchaseCount": "0",
"DebitPurchaseAmount": "0.00",
"DebitReturnCount": "0",
"DebitReturnAmount": "0.00",
"ControlNo": "0221",
}

```

5. HostedCheckout Direct

Overview of Worldpay's Integrated Hosted Platforms

This section of the guide will supply details specific to **MercuryPay's HostedCheckout** platform. HostedCheckout is recommended for existing POS applications requiring a unified payments experience on the MercuryPay platform. This guide combines information on HostedCheckout related specifications previously found in separate guides: HostedCheckout for eCom, for POS, for P2PE and Cascading Style Sheets (CSS) Functionality.

HostedCheckout and HostedPayments

On an enterprise level, Worldpay supports multiple eCommerce solutions, as in Worldpay eCommerce, Express Hosted Payments and MercuryPay HostedCheckout. These other options were independently designed to be compatible with the platform on which they were originally developed and, because of this, helped maintained merchant operations on those platforms such as boarding, funding and shared tokenization.

- For a unified integration solution and a streamlined merchant experience between brick and mortar and online shopping carts, we wish to preserve this unified platform approach for eCommerce payments.
- Although Worldpay's eCommerce platforms share many common processing features, the SDKs and APIs will be different; additionally, access to broader enterprise implementation requirements and amenities such as account updating and fraud prevention tools will also vary.

MercuryPay HostedCheckout

MercuryPay HostedCheckout provides a secure payment API for eCommerce website developers by redirecting the transaction path or using an iFrame direct to the Worldpay Integrated Payments secure payment site. It was designed for use with Online and Mobile eCommerce payment solutions and is also applicable when integrated into card-facing POS applications (often for a secure manual entry option). The MercuryPay HostedCheckout solution targets the final checkout process by providing a customizable payment page for collection and submission of card data. Because the payment page is hosted by Worldpay, card data is kept off of the solution provider's servers, keeping them from qualifying as a "Service Provider" as defined by the PCI Council. HostedCheckout does not replace the checkout process entirely, just the handling of sensitive credit card data.

- HostedCheckout eliminates the need for web developers to handle cardholder data.
- HostedCheckout has been integrated with MToken™ from its inception. MToken is MercuryPay's tokenization process which returns a unique, dynamically generated token that may be stored "as is" for subsequent use. This helps merchants to implement features such as recurring billing and card-on-file programs. Subsequent transactions using a Token call can be processed using Transaction Web Service, outlined in this guide, or by either SOAP wrapped XML or REST/JSON remote procedure calls pointing to MercuryPay's Web Services platform.
- With recent mandates for identifying subsequent transactions, referred to by the card brands as the process of maintaining and processing with "Credentials on File," HostedCheckout has implemented new *RecurringData* fields in support of these new recurring transaction types.
- In 2020, Worldpay from FIS completed American with Disabilities Act (ADA) upgrades to MercuryPay Hosted Checkout following the Level A compliance standards of the Web Content Accessibility Guidelines (WCAG) 2.1.

HostedCheckout for eCommerce (or Adaptable for In-Store)

HostedCheckout for eCommerce

MercuryPay's HostedCheckout API was developed as a secure eCommerce, card-not-present payment page solution.

The payment page was designed purposely neutral to provide a simple, uncluttered user interface (UI) to collect the cardholder data for processing the transaction.

The payment page redirects to or uses an embedded iFrame directly to our hosted servers.

This page includes required fields for manually entering Card Number, Expiration Date, Card Security Code, and Name on Card. Here is an example of our iFrame payment page:

The payment page also includes

- Supported card brand Logos, required for eCommerce implementations
- Optional (yet highly recommended for fraud protection) fields for Address and Postal code verification (AVS). See details on how to manage AVS results [here](#).
- Optional Worldpay "Security Logo" which indicates the Terms of Use and that the site is hosted by Worldpay. (Note at this time, the logo uses our pre-Worldpay branded company name, Vantiv.)

Important to note: The eCommerce payment page only allows manual card entry. See details below for adapting HostedCheckout for in-store card present implementations.

Tested Browsers and Devices

The following browsers and devices have been tested for HostedCheckout eCommerce:

Tested Browsers	Tested Devices
Internet Explorer 10	iPhone 6 and 7 using iOS (Latest Version)
Internet Explorer 10	Samsung Galaxy S6 Edge
Firefox (Latest Version)	iPad Air 2
Google Chrome (Latest Version)	
Safari on latest iOS	

HostedCheckout for In-Store (HostedCheckout POS)

Although HostedCheckout was designed for eCommerce Shopping Cart payment pages, the API can be easily adapted for in-store use. Included here for reference are the details specific to HostedCheckout for POS. If integrating HostedCheckout for eCommerce only, please proceed to [Integration Three Step Process](#) in this guide.

HostedCheckout can be used in card-present, in-store environments implemented as a secure manual entry solution and, when used with a keyboard-emulated card reader, for swiped transactions. When implementing in this manner, please note:

- Always use the Checkout pages with "POS" in the URL. (for example: <https://hc.mercurycert.net/CheckoutPOS.aspx>)
- **Encrypted** and **Clear-text** MSR card readers are supported.

HostedCheckout POS Specific Parameters

Note there are several HostedCheckout POS specific parameters used in the **InitializePayment** command which will drive the payment page UI presented:

Parameter	Type	Length	Description	Required
Keypad	String	3	Valid values = On or Off . If <i>DefaultSwipe</i> = "Swipe" keypad will not display. Note: When using a tablet, value should be off.	No
DefaultSwipe	String	6	Determines the default state of the HostedCheckout page entry method radio buttons. User may still choose another entry method by selecting the other radio button. This value is ignored if <i>CardEntryMethod</i> is swipe or manual . Valid values = Manual or Swipe . NA for mobile POS .	No
CardEntryMethod	String	6	Valid values = Manual , Swipe , or Both . If manual , swipe option is unavailable. If swipe , manual option is unavailable. Default value is Both . If value is swipe or manual , the card entry method radio buttons will get hidden and user will be unable to switch entry methods. For mobile POS page, Both is not an option. Must be swipe or manual.	No

HostedCheckout POS UI

Basic examples of the payment page UI presented for a HostedCheckout POS implementation for both a swiped card and a key-entered card:

Credit Sale \$8.14

☐ Swipe Card
☒ Manually Enter Card Number

Card Number
 Exp Date
 CVV

Process

Credit Sale \$8.14

☒ **Swipe Card**
☐ Manually Enter Card Number

Card Number
 Exp Date
 CVV

Process

Supported Clear-Text MSRs

Any clear-text, keyboard-emulated MSR can be used as long as it can support the correct output formatting.

- Output must include both Track1 and Track2 with NO *carriage return* between the tracks.
- The MSR should output a *carriage return* at the end of the output.
- Tested Clear Text/Keyboard Emulated MSRs:

Desktop PC environment

- MagTek® USB Mini Magstripe Card Reader KB, part number 21040108
- IDTECH® card reader, model IDMB-334112B

Desktop PC environment with a touch screen monitor	<ul style="list-style-type: none"> Planar® PT 1585P touch screen monitor tested MagTek USB Mini Magstripe Card Reader KB, part number 21040108 IDTECH card reader, model IDMB-334112B
Tablet environment	<ul style="list-style-type: none"> Apple® iPad® Air 2 tested with a MagTek® Bullet Card Reader

Supported Point-to-Point Encrypted MSRs

Two MagTek keyboard-emulated encrypted MSRs are supported by the HostedCheckout API:

Manufacturer	Device	Remarks
MagTek	BulleT KB	with BlueTooth
	Dynamag KB	KB encrypted MSR

Note: The **MagTek IPAD-KB** PIN pad was supported in earlier API versions, but this device has been end-of-lifed by the manufacturer and **is no longer supported**.

When a P2PE-enabled MerchantID is used for the **InitializePayment** command, the payment page will display a single field for the swiped entry of the card, as in this example:

- When a P2PE-enabled MerchantID is used for InitializePayment, the **DefaultSwipe** parameter is ignored – the ‘Manually Enter Card Number’ option will not be displayed on the HostedCheckout page.

Encryption Keys

- Encrypted devices have unique encryption keys. This device key, often referred to as the **KSID**, is identified by the first seven non-masked digits, and varies by manufacturer, injection vendor and platform.
- Separate devices are distributed for testing and for production. Per PCI requirements, testing devices may only be used in testing environments, and production devices may only be used in production environments.

Note: P2PE Encryption and Key Entry using HostedCheckout Payment Page Requires two MIDs

Multiple MerchantIDs Required for Encrypted MSR and Key Entry

On the MercuryPay platform, MerchantIDs are configured specific to the transaction traffic passing through them—a P2PE enabled MerchantID cannot be used for non-encrypted key-entry through the HostedCheckout payment page. If you are implementing both HostedCheckout with an encrypted MSR and wish to also support key-entry using the HostedCheckout payment page, please note that a second MerchantID is required.

To solve for this, your integration will need to support multiple MerchantIDs and Password values – one with P2PE encryption enabled, and one without.

- The user will need to select whether they’re swiping or keying in the card.
- An InitializePayment call is made with the appropriate MerchantID depending on whether they are swiping or key-entering the card.
- This will result in two batch deposits for the merchant.

Common P2PE Error Responses

More details on encryption methods, specifications and error handling are included in the MercuryPay Platform Integration Guide. Here is a quick summary for HostedCheckout users.

Connectivity Errors

In the rare event that Worldpay's encryption service is unavailable, or if there is an issue accessing the secure server; a "Decryption Service Unavailable" error will be returned:

```
<ResponseOrigin>Server</ResponseOrigin>
<DSIReturnCode>004116</DSIReturnCode>
<CmdStatus>Error</CmdStatus>
<TextResponse>Decryption Service Unavailable.</TextResponse>
```

Decryption Service Unavailable errors vary: Corrupt, Malformed or Incorrect Data

In the event that one of the P2PE encryption data elements contains corrupted, malformed or in any way incorrect data, Worldpay's encryption service will return a text response containing Decryption Service Unavailable followed by the field detected to contain the failed data. These "Failure Messages" vary and are dynamically generated based on the reason for the failure, for example:

```
<CmdResponse>
  <ResponseOrigin>Server</ResponseOrigin>
  <DSIXReturnCode>004116</DSIXReturnCode>
  <CmdStatus>Error</CmdStatus>
  <TextResponse>Decryption Service Unavailable-200 Response with Status: Failure
  Message:Invalid EncryptedBlock</TextResponse>
```

HostedCheckout Integration: Three Step Transaction Process



1. Initiate the Payment Process. The web site application initiates a call to a Worldpay web service (server to server) to start the checkout process and obtain a unique *PaymentID*. Transaction data fields such as the *MerchantID* and the *TotalAmount* are passed to Worldpay at this time.
2. Display the Worldpay HostedCheckout page for Secure Processing. This redirects the user to the Worldpay HostedCheckout page, or the developer embeds the page in an iFrame on their site. The unique *PaymentID* is passed to Worldpay in hidden fields using an HTTP form post. The customer will enter their credit card information on Worldpay's hosted page and then press a **[Submit/Process Payment]** button. The payment is processed and control is relinquished back to the developer to continue processing as needed.
3. Verify Payment. The *VerifyPayment* method is used to confirm the successful status of a payment. This allows the developer to validate the transaction approval or decline status and includes additional information about the payment after Worldpay has processed it, including receipt data.

The details, required or optional parameter specifications on these three steps are described below.

Step 1: Initiate the Payment Process

To initiate the payment process, the **Initialize Payment** method in the HostedCheckout web service must be called to pass the information needed to setup the process.

HostedCheckout HCService

Development URL:	https://hc.mercurycert.net/hcws/hcservice.aspx
Production URL:	https://hc.mercurypay.com/hcws/hcservice.aspx

Web Service Definition: InitializePayment

The information passed is stored on Worldpay's server for a defined period of time. A unique **PaymentID** will be returned.

Input Parameters to InitializePayment Method

Note: Parameter names are case sensitive; please follow exactly as shown. Length is maximum length unless where specified. C# sample code follows.

Parameter	Type	Length	Description	Required
MerchantID	Numeric	24	The Merchant's account or <i>MerchantID</i> that payments will be processed under. The typical format for a HostedCheckout merchant ID is: 88430087469; Request a test <i>MerchantID</i> from your Implementation Consultant.	Yes
OperatorID	String	10	Employee associated with the transaction	No
Password	String	16	The HostedCheckout password generated by merchant or dealer/developer on the MercuryView portal to allow access to the HostedCheckout. Please contact your developer integrations analyst for the test password.	Yes
LaneID	String	3	This tag is required for card-present transactions sent to Worldpay when implementing HostedCheckout for POS with multiple workstations. Each terminal or input device must have a unique, alphanumeric identifier. No special characters, dashes or slashes.	Yes Condition- al
TranType	String	16	The type of payment Worldpay should process. Values are <i>PreAuth</i> , <i>Sale</i> , and <i>ZeroAuth</i>	Yes
PartialAuth	String	3	A card-present requirement for Prepaid cards; Values = on or off ; Default is off . Requires code to prompt for the remaining balance if the card has insufficient funds to cover the <i>TotalAmount</i> .	No
TotalAmount	Numeric	8	The total amount of the order to charge the card holder. Two decimal places are required. Format is 99999.99. <i>ZeroAuth</i> amount must be 0.00. If <i>TotalAmount</i> is left null, or is not a valid number, you will receive an error response back, such as "The remote server returned an error: (500) Internal Server Error."	Yes
TokenType	String	9	<i>MToken</i> or <i>OmniToken</i> . Defaults to <i>MToken</i> .	No
Frequency	String	9	<i>OneTime</i> or <i>Recurring</i>	Yes
RecurringDataRequest	String	20	Valid <i>RecurringDataRequest</i> content (other values passed will fail): <blank> Recurring Installment Unscheduled CardHolderInitiated	No
Invoice	Numeric	16	Invoice Number from the eCommerce site. It will be used for both the <i>Invoice</i> and the <i>RefNo</i> fields when the pre-auth is submitted.	Yes
Memo	String	40	The <i>Memo</i> tag is the product name and version number of the developer's software.	Yes
TaxAmount	Numeric	8	The tax amount of the transaction. Two decimal places are required. If there is no tax amount, pass in '0.00'.	Yes

			Format is 99999.99.	
CardHolderName	String	30	Customer Card Holder Name as it appears on card. This will pre-populate the Card Holder name on the checkout page and will appear in Worldpay reporting. This can be pre-populated with the Billing Name if you don't have the Cardholder's Name from the card.	No
AVSFields	String	4	Recommended for fraud protection. Valid values = Off , Zip , or Both . If provided will return Issuer Address Verification response code. Application must interpret and allow or reverse.	No
AVSAddress	String	8	Recommended for fraud protection. Street number portion of Card Holder Billing Address.	No
AVSZip	String	5 or 9	Recommended for fraud protection. U.S. Cardholder Zip. Canadian Postal Codes must use the following alpha numeric format "A1A1A1". Letters D, F, I, O, Q, and U are not permitted.	No
CVV	String	3	Card Security Code. Valid values = off or on . Determines whether CVV field is displayed. Default is on . If provided will return Issuer CVV response code. Application must interpret and either allow or reverse.	No
CustomerCode	String	16 max	Customer Code.	No
ProcessCompleteUrl	String	512	The URL on the Merchant's Ecommerce site to return to after the Worldpay HostedCheckout page finishes processing the transaction. For an iFrame: This page will display within the iFrame window. Must be a valid URL.	Yes
ReturnUrl	String	512	For a Redirect: The URL to return to in the event the user wants to cancel and go back to the eCommerce site. Typically this is the E-Commerce Order Page. Also used during a session timeout or if Worldpay goes into maintenance mode. For an iFrame: The [Cancel] button is not displayed. However, the <i>ReturnUrl</i> is used for a session timeout. In that case, the <i>ReturnUrl</i> page will display in the iFrame. This URL must not be the same page that the iFrame exists on or it will cause the iFrame to display duplicate or nested pages. Use a separate URL such as the <i>ProcessCompleteURL</i> or some other page. Must be a valid URL.	Yes
PageTimeoutDuration	Numeric	2	Values are 0 to 15 , and indicate the number of minutes until the page times out. Defaults to 0 . If <i>PageTimeoutDuration</i> is greater than 0, timeout is active. Upon timeout, user will be automatically redirected to the <i>ReturnUrl</i> and a <i>ReturnCode</i> of 105 will be sent from HostedCheckout indicating the reservation can be released.	No
PageTimeoutIndicator	String	3	Values are on and off . Default is off . The timeout indicator will count down to 0 and then redirect. iFrame size must be increased to 600 X 600 if FontSize = large or 550 X 500 if FontSize = medium.	No
DisplayStyle	String	8	Valid values are Mercury or Custom .	No

Default is **Mercury**.

Mercury: Displays a light blue banner with a white background, and black Arial text. On an iFrame, no banner is displayed.

Custom: Displays the supplied logo from LogoURL, background color, and font information. If no LogoURL is passed or if it is an iFrame, then no logo is displayed.

Required for CSS support. See Appendix for additional information.*

BackgroundColor	String	16	<p>Specifies the background color of the page. Default is white. Format is hexadecimal or a standard named color. The color should be consistent with the background displayed on the Merchant's web site to make the integration more seamless.</p> <p>Note If 'Mercury' is passed as DisplayStyle, the background color will be white regardless of the value passed.</p>	No
FontColor	String	16	<p>The color for the labels on the page. Default is black. Format is hexadecimal or HTML color name (i.e., #FF0000 or Red)</p> <p>Note If 'Mercury' is passed as DisplayStyle, font color will be black, regardless of the value passed.</p>	No
FontFamily	String	64	<p>The font family to be used for the labels on the page. Default is Arial. There are 3 pre-defined font family values that may be used:</p> <p>FontFamily1, FontFamily2, FontFamily3.</p> <p>The pre-defined font families are defined as follows:</p> <p>FontFamily1: Arial, Verdana, Geneva, Helvetica, sans-serif</p> <p>FontFamily2: Georgia, Times New Roman, Times, serif</p> <p>FontFamily3: Courier New, Courier, monospace</p> <p>A custom font family may also be specified using comma-delimited string of fonts. Example: 'Comic Sans MS, Arial, sans-serif'</p> <p>Note: If 'Mercury' is passed as DisplayStyle, font family will be Arial, regardless of the value passed.</p> <p>Not utilized in mobile integration.</p>	No
FontSize	String	8	<p>The font size. Default Medium. Options are Small, Medium, Large.</p> <p>Note: If 'Mercury' is passed as DisplayStyle, font size will be Medium, regardless of the value passed.</p> <p>Not utilized in mobile integration.</p>	No
LogoUrl	String	512	<p>For a Redirect: Specifies the URL of the logo to display at the top of the page. Default is no logo. The logo should be located on the Merchant's eCommerce web site at a public URL. The logo should be consistent with the Merchant's web site to make the integration more seamless.</p> <p>Note: If 'Mercury' is passed as DisplayStyle, this will not be displayed.</p> <p>This URL must use the HTTPS protocol. If the LogoURL is HTTP, security warnings will be displayed to the end user.</p> <p>Must be a valid image URL.</p>	No

			For an iFrame: The LogoURL is not used because the iFrame is already embedded on the developer's web page.	
PageTitle	String	64	For a Redirect: The html title of the page that the browser will display in the title bar. Typically this is the name of the merchant's web site. Use this to brand the checkout page similar to the Merchant's web site. If this is not specified, the default value is 'Mercury Secure Checkout'.	No
			For an iFrame: Not applicable.	
OrderTotal	String	3	Indicates whether the Order Total box will be displayed on the page. This only applies to iFrame. Valid values are on and off . Default is on .	No
TotalAmountBackgroundndColor	String	8	Background color of the Order Total box. Default is gray.	No
SubmitButtonText	String	32	Custom text for the [Submit] button. Default value is Submit Payment . Note that on iFrame, some 32 character strings will not fit due to physical space constraints, and the text will be cut off. DisplayStyle = Custom or Mercury.	No
CancelButtonText	String	32	Custom text for the [Cancel] button. Default value is Cancel . Only applies to Redirect. DisplayStyle = Custom or Mercury.	No
ButtonTextcolor	String	16	Custom button text color. Can be a HTML Web color, such as "Red", or a color hex value such as #FF0000	No
ButtonBackgroundColor	String	16	Custom button background color. Can be a HTML Web color, such as "Red", or a color hex value such as #FF0000	No
CancelButton	String	3	Displays a [Cancel] button on the iFrame. If selected it returns user to the <i>ReturnUrl</i> page. Values are on and off . Default is off . This only applies to iFrame pages.	No
JCB	String	3	Display JCB logo. Values are on and off . Default is on .	No
Diners	String	3	Display Diners logo. Values are on and off . Default is on .	No
CUP	String	3	Display UnionPay logo. Values are on and off . Default is on .	Yes
CanadianZip	String	3	Allow Canadian postal codes. Values are on and off . Default is on .	No
AlternateMCC	String	4	For applicable loan or cash advance indicator using a value of '6051' MCC. Visa, MasterCard only. Amex will decline. Note: although implemented to enable a second MCC, this value is no longer applicable.	No

Return Parameters from InitializePayment Method

Parameter Name	Type	Length	Description
ResponseCode	Integer		0 – Success 100 – AuthFail (bad password or <i>MerchantID</i>) 200 –Mercury Internal Error 300 –ValidationFail: General Validation Error (See Message for list of

			validation errors)
PaymentID	String	36	A unique identifier that identifies this particular payment process. Important Note: reusing a PaymentID will trigger an error!
Message	String	unlimited	Success if initialization was successful, a verbal description of one or more errors if initialization did not succeed.

Cascading Style Sheets (CSS) Functionality

CSS (Cascading Style Sheets) provide extensive styling capability for HostedCheckout pages. CSS utilizes a separate method to submit and have Worldpay store the merchants CSS. This process is outside the initialize payment process. See the Appendix for more information on CSS.

InitializePayment web method C# sample code

This sample code calls the **InitializePayment** web method. It is C# used in an asp.net button click event. This code references two variables on the markup page: **txtPaymentID** and **lblInitResponseMessage**

```
//Build request for the HCService initialize requests
HCService.InitPaymentRequest hcRequest = new HCService.InitPaymentRequest();

hcRequest.MerchantID = this.txtMerchantID.Text;
hcRequest.Password = this.txtPassword.Text;
hcRequest.TranType = this.rblTranType.SelectedItem.Text;
hcRequest.TotalAmount = dAmt1 + dAmt2 + dTaxAmt;
hcRequest.Frequency = this.rblFrequency.SelectedItem.Text;
//hcRequest.RecurringData = this.txtRecurringData.Text; //pending future use
hcRequest.Invoice = this.txtInvoice.Text;
hcRequest.TaxAmount = Convert.ToDouble(this.txtTaxAmt.Text);

hcRequest.CardHolderName = this.txtName.Text;
hcRequest.AVSAddress = this.txtAddress.Text;
hcRequest.AVSZip = this.txtZip.Text;
hcRequest.CustomerCode = this.txtCustomerCode.Text;
hcRequest.Memo = this.txtMemo.Text;
hcRequest.ReturnUrl =

System.Web.Configuration.WebConfigurationManager.AppSettings["returnURL_POS"].ToString();
hcRequest.ProcessCompleteUrl =
System.Web.Configuration.WebConfigurationManager.AppSettings["processCompleteURL_POS"].ToString();
hcRequest.BackgroundColor = this.txtBGColor.Text;
hcRequest.FontColor = this.ddlMultiColor.SelectedItem.Text;
hcRequest.FontSize = this.rblFontSize.SelectedItem.Text;
hcRequest.FontFamily = GetFontFamily();
hcRequest.PageTitle = this.txtPageTitle.Text;
hcRequest.DisplayStyle = this.rblDisplayStyle.SelectedItem.Text;
hcRequest.CancelButtonText = this.txtCancelButton.Text;
hcRequest.SubmitButtonText = this.txtPaymentButtonText.Text;
hcRequest.AVSFields = this.rblAVSFields.SelectedValue;
hcRequest.DefaultSwipe = this.rblDefaultSwipe.SelectedValue;
if (this.rblTranType.SelectedItem.Text == "VoiceAuth")
    hcRequest.VoiceAuthCode = this.txtVoiceAuthCode.Text;
hcRequest.Keypad = this.rblKeypad.SelectedItem.Text;
hcRequest.PartialAuth = this.rblPartialAuth.SelectedItem.Text;
hcRequest.OperatorID = this.txtOpID.Text;
hcRequest.TerminalName = this.txtTermName.Text;
hcRequest.ButtonBackgroundColor = this.txtBtnBgColor.Text;
hcRequest.ButtonTextColor = this.txtBtnTextColor.Text;
hcRequest.CardEntryMethod = this.txtCardEntryMethod.Text;
hcRequest.CancelButtonDefaultImageUrl = this.txtCancelBtnDefault.Text;
hcRequest.CancelButtonHoverImageUrl = this.txtCancelBtnHover.Text;
hcRequest.SubmitButtonDefaultImageUrl = this.txtSubmitBtnDefault.Text;
hcRequest.SubmitButtonHoverImageUrl = this.txtSubmitBtnHover.Text;
hcRequest.PageTimeoutDuration = this.ddlPageTimeoutDuration.SelectedValue;
hcRequest.PageTimeoutIndicator = this.rblPageTimeoutIndicator.SelectedValue;
hcRequest.TotalAmountBackgroundColor = this.txtTotalAmtBgColor.Text;
hcRequest.CW = this.rblCW.SelectedValue;
```



```

if (this.rblForce.SelectedIndex != -1)
    hcRequest.ForceManualTablet = this.rblForce.SelectedItem.Text;
hcRequest.LaneID = this.txtLaneID.Text;
hcRequest.CanadianZip = this.rblCanadianZip.SelectedValue;

//Call the InitializePayment Web Service Method
HCSERVICE.HCSERVICE hcWS = new HCSERVICE.HCSERVICE();
HCSERVICE.InitPaymentResponse response = new HCSERVICE.InitPaymentResponse();
response = hcWS.InitializePayment(hcRequest);

```

If you receive an error message: **"Your XML is invalid. Check your XML syntax and refer to the Integration Guide,"** a formatting error may have occurred. This can be caused by a misspelled tag, a tag name with an incorrect case, a bad namespace, or other malformed xml.

Note: Step 2 outlined below is divided into sub categories: **Redirect** option or embedded **iFrame**

Step 2 Redirect: Display the HostedCheckout Payment Page using the Redirect Option

The browser redirects the user to the HostedCheckout payment page. Upon completion, the user is redirected back to the original site.

Standard Redirect Page for **eCommerce**:

Development URL:	https://hc.mercurycert.net/Checkout.aspx
Production URL:	https://hc.mercurypay.com/Checkout.aspx

Standard Redirect Page for **POS**:

Development URL:	https://hc.mercurycert.net/CheckoutPOS.aspx
Production URL:	https://hc.mercurypay.com/CheckoutPOS.aspx

Mobile Optimized Redirect Page for **POS** and **eCommerce**:

Development URL:	https://hc.mercurycert.net/mobile/mCheckout.aspx
Production URL:	https://hc.mercurypay.com/mobile/mCheckout.aspx

Hidden Fields in HTML Form Post to HostedCheckout

Field Name	Description	Required
PaymentID	The unique identifier returned by the <i>InitializePayment</i> web service call.	Yes
ReturnMethod	<p>Returns users to your site using a GET or a POST. Default is POST.</p> <p>If ReturnMethod = POST, <i>PaymentID</i>, <i>ReturnCode</i>, and <i>ReturnMessage</i> will be passed back as hidden fields in an html form.</p> <p>If ReturnMethod = GET, the <i>PaymentID</i> and <i>ReturnCode</i> will be returned as Query String parameters in the URL.</p> <p>Note: HTTPS is required</p>	No

Redirect sample code

This sample code will redirect the browser to Worldpay's HostedCheckout page. It is server-side C# code used in an asp.net click event that creates an html response that will redirect to Worldpay.

```

<!--Set the necessary variables before building html.-->
String hostedCheckoutURL = ConfigurationManager.AppSettings["HostedCheckoutURL"];
string paymentID = this.txtPaymentID.Text;

<!--Build an html form post to be sent back to the browser -->
<!--It will redirect the browser to the HostedCheckout page-->

```

```

Response.Clear();
Response.Write("<html><head>");
Response.Write("</head><body onload=\"document.frmCheckout.submit()\">");
Response.Write("<form name=\"frmCheckout\" method=\"Post\" action=\"\" +
hostedCheckoutURL + \"\" >");
Response.Write("<input name=\"PaymentID\" type=\"hidden\" value=\"\" + paymentID +
\"\">");
Response.Write("</form>");
Response.Write("</body></html>");
Response.End();

```

Mobile Optimization and Tablets

In the event that a smartphone user ends up on a HostedCheckout page that is designed for a desktop browser, HostedCheckout will attempt to redirect the user to our mobile optimized page. If the user's device is a larger tablet such as an Apple iPad, HostedCheckout will not redirect to the mobile version.

- Larger tablets are best viewed with the desktop version of HostedCheckout pages.
- Some larger tablets may still redirect to the mobile version. Typically this happens when the tablet's "UserAgent" string in the browser indicates that it is mobile and may not indicate it is a tablet. The HostedCheckout mobile pages should render without issue on the larger tablets.

Step 2 iFrame: Embedded iFrame Option—Desktop and Mobile Optimizing

Embedding the HostedCheckout page in an iFrame

The HostedCheckout iFrame allows the user to stay on the integrated checkout page - the Hosted Payment page is displayed within an iFrame.

Standard iFrame for **eCommerce**:

Development URL:	https://hc.mercurycert.net/CheckoutIFrame.aspx
Production URL:	https://hc.mercurypay.com/CheckoutIFrame.aspx

Standard iFrame for **POS**:

Development URL:	https://hc.mercurycert.net/CheckoutPOSiFrame.aspx
Production URL:	https://hc.mercurypay.com/CheckoutPOSiFrame.aspx

Mobile Optimized iFrame for **POS** and **eCommerce**:

Development URL:	https://hc.mercurycert.net/mobile/mCheckoutIFrame.aspx
Production URL:	https://hc.mercurypay.com/mobile/mCheckoutIFrame.aspx

Desktop iFrame (CheckoutIFrame.aspx) Details

Create an iFrame HTML tag on your main shopping cart or order page where you need to collect the payment information.

- Set your iFrame height and width attributes to 450 pixels. The default payment information content has been optimized for display at 450 x 450 pixels. However, you may have a requirement to modify the height and width values or change the value from pixels to a percentage. If you set the height and width attributes to something other than 450 x 450 pixels, take care to ensure scrolling is not forced in your target resolution.
- Set the scrolling attribute to **auto**.
- Set the **src** attribute (used to specify the address of the document to embed) to an empty string to keep the iFrame from loading before you are ready.

Example of the markup for an iFrame

Combining the style setting of "**display: none**" with the **onload="this.style.display='block';"** will help avoid the flashing effect when the iFrame loads. Note that the iFrame will not be visible at this point because of the **src=""** setting. The 'Your browser does not support...' text only appears if the browser does not support iFrames.

Example HTML:

```
<div style="height: 450px; margin-left: 253px;">
  <iFrame id="ctl00_MainContent_ifrm"
    src="https://hc.mercurycert.net/Checkoutiframe.aspx?pid=e727bd2e-c1ae-43c9-aadb-e20fd604ae4e"
    height="450px" width="450px" scrolling="auto" frameborder="0" style="text-align: center;
    display: none;"> onload="this.style.display='block';"> Your browser does not support
    iFrames. To view this content, please download and use the latest version of one of the
    following browsers: Internet Explorer, Firefox, Google Chrome or Safari.
  </iFrame>
</div>
```

Mobile iFrame (mCheckoutiFrame.aspx) Details

1. Create an iFrame HTML tag on your main shopping cart or order page where you need to collect the payment information.
 - Enclose the iFrame in the <div> tag, and set the height of this div to approximately 500px. Do not set the width of the div. -webkit-overflow-scrolling: touch; may help with scrolling on some devices.
 - Set your iFrame width to 100%, and the height attribute the same as the enclosing div.
 - Set the scrolling attribute to **yes**.
 - Set the **src** to an empty string to keep the iFrame from loading before you are ready.

Below is an example of the markup for an iFrame for a mobile device. Note that the iFrame will not be visible at this point because of the **src=""** setting. The 'your browser does not support...' text only appears if the browser does not support iFrames.

Example HTML:

```
<div style="height: 500px; -webkit-overflow-scrolling: touch; padding-bottom: 10px;">
  <iFrame id='ifrm' src="" width="100%" height="500px"
    scrolling="yes" frameborder="0" runat="server" style="text-align: center;
    border: 1px solid #A3221C;">Sorry, your browser doesn't support iframes.</iFrame>
  &nbsp;
</div>
```

2. After you have called *InitializePayment*, and when you are ready to display and load the iFrame page, set the **src** property of the iFrame from either the client side script or server side code:

src= <https://hc.mercurycert.net/mobile/mCheckoutiFrame.aspx?pid=e727bd2e-c1ae-43c9-aadb-e20fd604ae4e>.

- The URL must contain a **pid** query parameter that is the *PaymentID* you received when calling *InitializePayment*.
- If you require the user to be returned to your site using **GET** instead of a **Form POST**, include **ReturnMethod=get** in the URL. The default is **post**.
- If your site is using HTTP instead of HTTPS (HTTPS is highly recommended), you should use a **GET** to avoid browser warnings.

src=<https://hc.mercurycert.net/CheckoutiFrame.aspx?pid=e727bd2e-c1ae-43c9-aadb-e20fd604ae4e&ReturnMethod=get>

- Once the user has pressed **[Submit Payment]**, on the HostedCheckout page within the iFrame, the iFrame will process the payment and then display the page specified in the *ProcessCompleteURL*.

Additional Notes on Use of iFrame and returnUrl:

- The **ReturnUrl** parameter passed in *InitializePayment* should not be the same page that contains the iFrame. Currently the only reason the *ReturnUrl* is used for an iFrame is when a session timeout occurs. If the *ReturnUrl* is the same as the page containing the iFrame, and a timeout occurs, it will cause a repetitive nested display in the iFrame.
- To handle this, use the page specified in the *ProcessCompleteURL*, or create an additional page for session timeouts. The page content size should display in a 450 x 450 pixel iFrame. You can display whatever you want on this page based on the *ReturnCode* in the form post as identified in section 2c below. (**Note: If you modify the suggested iFrame size**, take care to ensure your content displays properly in its defined size based on your target resolution.)
- Similarly, the page specified in the *ProcessCompleteURL* parameter passed in *InitializePayment* is going to appear within the iFrame window. The page content size should display in a 450 x 450 pixel iFrame. (**Note: If you modify the suggested iFrame size**, take care to ensure your content displays properly in its defined size based on your target resolution.) The page will receive form post parameters as identified in section 2c below. You can display whatever information they want because the developer controls this page. It is also possible to communicate between the iFrame and the parent page by navigating the DOM iFrame object.

Parameters Passed Back to ProcessCompleteURL or returnUrl Page via GET or POST

Once the user enters the credit card data and presses **[Submit Payment]**, the payment is processed and control is returned back to the web site developer via the **ProcessCompleteURL**. The fields passed to the developer are listed in the table below.

- When returning to the eCommerce web site, if the *ReturnMethod* is a **GET**, the fields below will appear as query string parameters in the URL.
- For a *ReturnMethod* of **POST**, they will appear as hidden fields in an html form.
- For a redirect, the user is redirected back to the *ProcessCompleteURL*.
- For an iFrame, the iFrame will redirect to the page specified by the *ProcessCompleteURL* and display it within the iFrame.

Important Note: It is very important to reference the ResponseCode/Message returned in the *ReturnUrl* and *ProcessCompleteURL*. Using them will allow you to continue the transaction flow in a logical manner.

Fields Returned in Form Post to the ProcessCompleteURL

Parameter Name	Type	Length	Description
PaymentID	String	36	The <i>PaymentID</i> that started the payment process is returned back to the eCommerce developer.
ReturnCode	Int		0 – Success 100 – AuthFail (bad password or id) 101 – CardDeclined – the card was declined for the transaction. Status=Decline 200 – Worldpay Internal Error 202 – LoadPaymentInfoFail – data saved from Initialize Payment invalid/missing 203 – ProcessPaymentFail – unable to process. Payment Status=Error. 204 – PreAuthFailDBErr – internal database error for PreAuth transaction 205 – SalesNotCompletedDBErr – internal database error for Sale transaction

Parameter Name	Type	Length	Description
			301 – ValidationCCFail – Credit Card failed Mod10 check multiple times 302 – ValidationServerSideFailure – can occur when scripting is off in user's browser so client side validation did not occur. 303 – ValidateNameFail —Invalid data entered in cardholder name field.
ReturnMessage	String	unlimited	Success: Your Transaction has been approved.* AuthFail, ValidationFail, ValidateCCFail, and ValidationServerSideFail: We are unable to process your order at this time. Please contact our customer service. MercuryInternalFail, FormPostFail, GetInitDBFail, ProcessPaymentFail, SavePreAuthDBFail, and SaveSaleDBFail: We are unable to process your order at this time. Please place your order again. * ReturnMessage is not returned if ReturnMethod = GET

The user may be sent back to the designated *ReturnUrl* for several different reasons as listed below. The *ReturnMessage* is provided, but does not necessarily need to be displayed.

Fields Returned in Form Post to the *ReturnUrl*

Parameter Name	Type	Length	Description
PaymentID	String	36	The PaymentID that started the payment process is returned back to the eCommerce developer.
ReturnCode	Int		102 – UserCancelled – the user pressed cancel. 103 – SessionTimeout – the session timed out. 104 – MaintenanceMode – the site is in maintenance mode. 105 – PageTimeout – duration time exceeded.
ReturnMessage	String	unlimited	102 – "The user pressed cancel." 103 – "The user session timed out." 104 – "Payment processing is temporarily unavailable at this time." 105 – "The transaction was cancelled because the time to complete the transaction was exceeded." *ReturnMessage is not returned if ReturnMethod = "GET".

Step 3: Verify the Payment

The **VerifyPayment** method in the HostedCheckout web service **must** be called to determine the authorization status of a payment. This allows the developer to get the status **PLUS** additional information about the payment after Worldpay has processed it.

The HostedCheckout web service is available through the URL:

HostedCheckout HCSERVICE

Development URL:	https://hc.mercurycert.net/hcws/hcservice.asmx
Production URL:	https://hc.mercurypay.com/hcws/hcservice.asmx

Web Service Definition: VerifyPayment

Important Note: *VerifyPayment* is a polling service and should be implemented based on the recommendations below.

General recommendations when calling *VerifyPayment*:

1. Call *VerifyPayment* for **no longer than 5 minutes per transaction**.

When you initiate the payment process (Step 1 above), set **PageTimeoutDuration** to **5**, which sets the timeout for 5 *minutes*. You can optionally have the timeout indicator display by using the **TimeoutIndicator** API call.

2. Call *VerifyPayment* at an interval of once every 5 seconds at most, preferably once every 10 seconds.
3. **Stop** calling *VerifyPayment* as soon as a status (Approved, Declined, Error, etc.) is received.
4. **Stop** calling *VerifyPayment* if the **PostBack** to your page from HostedCheckout succeeds.

Input Parameters

Parameter Name	Type	Length	Description
MerchantID	Numeric	24	The Merchant's account or <i>MerchantID</i> that payments will be processed under.
PaymentID	String	36	The unique identifier returned by the <i>InitializePayment</i> web service.
Password	String	16	The web services password assigned by Worldpay to the Merchant's account (<i>MerchantID</i>) to allow access to the web

Return Parameters

Field Name	Type	Length	Description
ResponseCode	Int		A Code indicating the result of the transaction (See associated Status): 0 – Success 100 – AuthFail (bad password or id) 200 – Worldpay Internal Error. Specific error will be logged in Worldpay's internal error log. 300 – Validation failure – one of the request parameters was either missing or invalid.
Status	Char	16	The Status of the transaction. Values: Approved, Declined, Error, Invalid, AuthFail, MPSError. <i>If ResponseCode = 0</i> Approved: The transaction was approved. Declined: The transaction was declined. Blank: The payment has not been processed at the time the method was called. Error: Indicates an error from the transaction processing host. Invalid: Indicates that the user entered invalid card data too many times and was therefore redirected back to the Merchants eCommerce site. <i>If ResponseCode = 100</i> AuthFail: The authorization failed for the given MerchantID and Password <i>If ResponseCode = 200</i> MPSError: An internal error has occurred.
StatusMessage	Char	64	The textual description of the status. This should not be displayed to the end user. Use <i>DisplayMessage</i> instead.
AcqRefData	Char	200	Required for <i>PreAuthCapture</i> and <i>PreAuth Reversal</i> .
Amount	Numeric	8	The amount of the transaction
AuthCode	Char	16	The Authorization Code. Required for <i>PreAuthCapture</i> .
AVSAddress	Char	8	Address used for AVS verification. Note it is truncated to 8 characters.

Field Name	Type	Length	Description
AVSResult	Char	4	The result of the AVS check. <i>AVSResult</i> codes are listed here . AVS mismatch may indicate fraud but will not decline the authorization. Your application will need to interpret the AVS result code and determine appropriate follow up action based on merchant business need, i.e. to allow or reverse the transaction and prompt for retry.
AVSZip	Char	9	Postal code used for AVS verification.
CardholderName	Char	32	The name on the card
CardType	Char	40	The type of card used to make the payment.
CustomerCode	Char	16	The customer code passed in <i>InitializePayment</i> .
CvvResult	Char	4	Result of the CVV check. <i>CVVResult</i> codes are listed here . CVV mismatch may indicate fraud but will not decline the authorization. Your application will need to interpret the CVV result code and determine appropriate follow up action based on merchant business need, i.e. reverse the transaction and prompt for retry.
DisplayMessage	Char	512	Worldpay's Recommended message to display to the end user. The eCommerce web site developer can use this message or display own message.
ExpDate	Char	4	Expiration date of card used. Can be stored in instances where card-on-file or recurring billing is being utilized and merchant desires to remind cardholder to update card information when expiration date approaches. Note: Expiration date cannot be printed on receipts or used in correspondence with card holder. Expiration date on receipts should be masked with X's, such as XXXX or XX/XX.
Invoice	Char	16	Worldpay responds back with the Invoice passed to us on the HostedCheckout request.
MaskedAccount	Char	12	Masked credit card number. (e.g.,: xxxxxxxx6781)
Memo	Char	40	The <i>Memo</i> tag is the product name and version number of the developer's software.
PaymentIDExpired	Boolean		If the <i>PaymentID</i> has been used to make a payment (even if they payment was declined or had an error), or if the session timed out while the user was on the HostedCheckout page, then this will be true . Otherwise, this will be false . Once the payment ID is set to <i>Expired</i> , it may not be used again to make a payment. The payment process must be started again with <i>InitializePayment</i> .
ProcessData	Char	200	Required for <i>PreAuthReversal</i> .
RefNo	Char	16	Transaction reference number. Required for <i>VoidSale</i> .
TaxAmount	Numeric	8	The tax amount of the transaction, passed in <i>InitializePayment</i> .
Token	Char	100	The token generated by the transaction that replaces the credit card #. Can be used in subsequent transactions such as <i>CreditPreAuthCaptureToken</i> .
TransPostTime	DateTime		The date/time that the transaction occurred in EST. If it is 1/1/0001 this is equivalent to blank or Null.
TranType	Char	10	<i>PreAuth</i> or <i>Sale</i>
RecurringDataRequest	String	20	(Pending) Confirms the original <i>RecurringData</i> requested
RecurringDataResponse	String	40	(Pending) Returns a pipe-delimited combined field response of <i>RecurringData</i> originally passed and network confirmation. Visa only. Note: If <i>RecurringData</i> not passed in <i>Submit</i> , it will not be returned in <i>VerifyPayment</i> of <i>VeriFyCardInfo</i> response

VerifyPayment Web Method Sample Code

This example calls the **VerifyPayment** web method and displays the response information.

```
<!--create the request-->
    HCService.PaymentInfoRequest verifyPaymentRequest = new HCService.PaymentInfoRequest();

verifyPaymentRequest.MerchantID = System.Configuration.ConfigurationManager.AppSettings["MerchantID"];
verifyPaymentRequest.Password = System.Configuration.ConfigurationManager.AppSettings["HCPassword"];
verifyPaymentRequest.PaymentID = txtPaymentID.Text;

<!--Call the VerifyPayment web method-->
HCService.HCService hcService = new HCService.HCService();
HCService.PaymentInfoResponse response = hcService.VerifyPayment(verifyPaymentRequest);
<!--get the information from the response-->
this.lblVerifyInfo.Text = "ResponseCode=" + response.Status + "<BR/>";
this.lblVerifyInfo.Text += "Status=" + response.Status + "<BR/>";
this.lblVerifyInfo.Text += "StatusMessage=" + response.Status + "<BR/>";
this.lblVerifyInfo.Text += "DisplayMessage=" + response.Status + "<BR/>";
this.lblVerifyInfo.Text += "Token=" + response.Token + "<BR/>";
this.lblVerifyInfo.Text += "AuthCode=" + response.AuthCode + "<BR/>";
this.lblVerifyInfo.Text += "AvsResult: " + response.AvsResult + "<BR/>";
this.lblVerifyInfo.Text += "CvvResult: " + response.CvvResult + "<BR/>";
this.lblVerifyInfo.Text += "CardType: " + response.CardType + "<BR/>";
this.lblVerifyInfo.Text += "RefNo: " + response.RefNo + "<BR/>";
this.lblVerifyInfo.Text += "Invoice: " + response.Invoice + "<BR/>";
this.lblVerifyInfo.Text += "AcqRefData: " + response.AcqRefData + "<BR/>";
this.lblVerifyInfo.Text += "TaxAmount: " + response.TaxAmount + "<BR/>";
this.lblVerifyInfo.Text += "TransPostTime: " + response.TransPostTime + "<BR/>";
this.lblVerifyInfo.Text += "Masked Acct: " + response.MaskedAccount + "<BR/>";
this.lblVerifyInfo.Text += "Amount: " + response.Amount + "<BR/>";
this.lblVerifyInfo.Text += "Cardholder Name: " + response.CardholderName + "<BR/>";
this.lblVerifyInfo.Text += "AVS Address: " + response.AVSAddress + "<BR/>";
this.lblVerifyInfo.Text += "AVS Zip: " + response.AVSZip + "<BR/>";
this.lblVerifyInfo.Text += "Tran Type: " + response.TranType + "<BR/>";
this.lblVerifyInfo.Text += "Payment ID Expired: " + response.PaymentIDExpired.ToString() + "<BR/>";
this.lblVerifyInfo.Text += "Customer Code: " + response.CustomerCode + "<BR/>";
this.lblVerifyInfo.Text += "Memo: " + response.Memo + "<BR/>";
<!--pending future development -->
this.lblVerifyInfo.Text += "RecurringDataRequest: " + response.RecurringDataRequest + "<BR/>";
this.lblVerifyInfo.Text += "RecurringDataResponse: " + response.RecurringDataResponse + "<BR/>";
```

Using MToken for Subsequent Transactions with Transaction Web Service

MToken is Worldpay Integrated Payment's proprietary technology use exclusively on the MercuryPay platform that replaces sensitive card data with a non-sensitive, undecipherable token. Only MercuryPay's token service can unscramble and decrypt the token, rendering it useful for payment processing.

- By calling the TransactionWebService (TWS) the token is used in place of card data to perform subsequent transactions for the same card such as capturing a *PreAuth*, voiding or reversing a transaction in the current batch, returning a previous transaction, card-on-file, or recurring billing.
- Alternately, the Token functions outlined here can also be supported using a direct SOAP or REST/JSON Web Service API's.

The actual card number (PAN) is used only when submitted by the user on the HostedCheckout web page. After the transaction is processed a token is retrieved using the *VerifyPayment* web method. For greater security and to extend their expiration date, tokens are dynamically generated and change with each use. It is important to store, manage and dispose of tokens correctly.

Important details when storing tokens:

- Only store a token if there is a business need and if this information supports your targeted business/merchants.

Note that although AVS data may be stored as non-sensitive card data, CVV data *may never be stored*.

- Build in retention and expiration timelines for any stored token record. Note that because of truncated card expiration data returned in the token response, developers should, if needed, obtain the expiration date prior to processing the request.
- Always dispose of expired tokens.
- A new token will always be generated and returned with each subsequent transaction associated with a particular card number. Therefore, if storing token data, always store only the most recent token returned and dispose of all previous tokens.

MercuryPay Tokenization Basics for use with HostedCheckout

Element	Req	Min	Max	Type	Description
Transaction: Token	Y	1	100	AN	Used in subsequent transaction as the actual token returned from Worldpay
Transaction: Frequency	Y	1	20	AN	Type of Token requested: OneTime or Recurring
TransResponse: Token	Y	1	100	AN	Token record returned from Worldpay

Initial token request: <Token> and <Frequency>

- Tokens** are unique reference records generated by Worldpay for credit transactions. A token is returned in the *VerifyPayment* response. In the event of a decline/error response, no token is generated.
- The token is used in subsequent transactions against the Transaction Web Service. With each transaction call, a new token will be sent back in the response.

The <Frequency> element: OneTime or Recurring

- OneTime** – token used in subsequent transactions and card-on-file when the amount and time in between usage is driven by cardholder input (not recurring). A *OneTime* token expires 6 months from the date the token is generated. With every use, a new token is returned and the token expiry is extended another 6 months.
- Recurring** – tokens used for weekly, monthly or annual recurring billing when amount is either fixed (as in an installment) or based on a merchant initiated schedule. A *Recurring* token expires 24 months from the date the token record is generated. With every use, a new token is returned and the token expiry is extended another 24 months.

Note: See information on additional card brand requirements for processing transactions using "Credentials on File" in this guide.

- Frequency** values must stay consistent in all subsequent MToken requests. For example, a *OneTime* Sale request must be followed with a *OneTime VoidSaleByRecordNo* Request or a *Recurring PreAuth* request can only be followed by a *Recurring PreAuthCaptureByRecordNo* request.
- Inconsistent usage of Frequency will cause the subsequent transaction request to fail with the following error: Error # 004119 - Token Service Unavailable. - 200 Response With Status: Failure Message: Parse token failure.

Multiple-Merchant Tokenization (MMT)—Using Tokens Across Multiple Locations

MMT enables a single business entity with multiple locations to use Worldpay's MToken platform across all of their locations *if*:

- Locations all share common ownership attributes (legal name, Tax ID and owner)
- Locations have a centrally housed, database infrastructure in place to support it.
- Tokens are tied to merchant locations and can only be used in MMT situations if they are "grouped". Upon request, Worldpay's Underwriting account specialists can authorize the creation of an MMT grouping that allows token sharing from one location to another.

Subsequent Token Transaction Calls using Transaction Web Service (TWS)

The TransactionService web service is available through the following URL's:

HostedCheckout TransactionService

Development URL:	https://hc.mercurycert.net/tws/transactionservice.aspx
Production URL:	https://hc.mercurypay.com/tws/transactionservice.aspx

There are eight token calls available to a developer in the Transaction Web Service (examples of each are provided on the following pages):

Token Call	Description
1. CreditPreAuthToken	For Recurring billing and card-on-file transactions
2. CreditPreAuthCaptureToken	For finalizing a PreAuth
3. CreditSaleToken	For Recurring billing and card-on-file transactions
4. CreditAdjustToken	For adjusting the amount of the original Sale or PreAuth Capture.
5. CreditVoidSaleToken	For voiding a Sale or PreAuthCapture transaction in the current batch
6. CreditReversalToken	For reversing PreAuth, PreAuthCapture, and Sale transactions
7. CreditReturnToken	For returning a previously ran transaction
8. CreditVoidReturnToken	For voiding a Return transaction in the current batch

Notes: HostedCheckout supports **PreAuth Reversals** (*CreditReversalToken*). This transaction allows a *PreAuth* to be reversed in cases where the *PreAuth* won't be captured. This keeps the funds from being held on the card holder's account. (This transaction requires the addition of the *AcqRefData* and *ProcessData* from the *PreAuth* response.)

Each of these transactions requires a token and the merchant's HostedCheckout password. See the examples of each call shown at the bottom of each section.

- Unlike the Initialize payment, note that the password is outside of the request tag and must be lower case.

For each of these transactions, a Credit Response structure is returned (see the Credit Response Returned Values for a list of the returned values). The structure is the same for each transaction, although some values will be blank if they are not applicable.

Token Functions also available using MercuryPay SOAP or REST/JSON Web Service

- Alternately, the Token functions outlined here can be supported using a direct SOAP or REST/JSON Web Service API's. The use of these APIs for subsequent token transactions does require maintenance of a second Web Service password, in addition to the HostedCheckout password.
- Using a standard Datacap TranType/TranCode message payload format, the SOAP wrapped XML or JSON RPC calls use the Datacap "ByRecodNo" token naming convention added to the TranCode for and passed to our web service endpoints for processing. Details are included in the MercuryPay Platform, SOAP and RESTful Integration guides available from your Integrations Consultant.

1. CreditPreAuthToken

CreditPreAuthToken Input Parameters

Parameter	Type	Length	Description	Required
-----------	------	--------	-------------	----------

Parameter	Type	Length	Description	Required
Address	String	8	CardHolder Billing Address. If provided will be used for Address Verification.	No
CardholderName	String	30	Customer Card Holder Name as it appears on card.	No
CVV	Numeric	3 or 4	Card Verification Value from back of credit card. Used for card verification	No
Frequency	String	9	OneTime or Recurring	Yes
Invoice	Numeric	16	Invoice Number from the eCommerce site. It will be used for both the <i>Invoice</i> and the <i>RefNo</i> fields when the pre-auth is submitted. Invoice Number must be unique.	Yes
Memo	String	40	The <i>Memo</i> tag is the product name and version number of the developer's software.	Yes
MerchantID	Numeric	24	The Merchant's account or <i>MerchantID</i> that payments will be processed under.	Yes
OperatorID	String	16	Operator ID	No
Amount	Double		The initial amount of the order to charge the card holder. Two decimal places are required. Format is 99999.99.	Yes
TerminalName	String	20	Terminal Name	No
Token	Numeric	100	The token generated by a <i>PreAuth</i> transaction that replaces the credit card #. Can be used in subsequent transactions.	Yes
Zip	String	5 or 9	Cardholder Zip. If provided, will be used for address verification.	No

CreditPreAuthToken Example

```

TransactionService.TransactionService ts = new TransactionService.TransactionService();
TransactionService.CreditPreAuthToken preAuthRequest = new TransactionService.CreditPreAuthToken()
{
    Token = "J9yQ0+wyBWS7G1A7TarjDtqzxk7DEOHDMhAF EAASNB0uYA==",
    Frequency = "OneTime",
    Amount = 9.00,
    MerchantID = "778825001",
    Invoice = "12345",
    TerminalName = "",
    OperatorID = "",
    Memo = "MPS WebServer v1.0.0",
    Address = "4 Corporate Square",
    Zip = "30329",
    CardHolderName = "Jen Brown",
    CVV = "123",
};

string password = "vNxE:r_CZNaHmkb^";
TransactionService.CreditResponse response = ts.CreditPreAuthToken(preAuthRequest, password);

```

2. CreditPreAuthCaptureToken

CreditPreAuthCaptureToken Input Parameters

Parameter	Type	Length	Description	Required
AcqRefData	String	200	Acquirer Reference Data	Yes
AuthCode	String	16	The Authorization code.	Yes
AuthAmount	Double		The amount authorized for the transaction. Two decimal places are required. Must match the <i>AuthorizeAmount</i> that was sent in the <i>PreAuth</i> . Format is 99999.99.	Yes
CardHolderName	String	30	The name on the card	No

CustomerCode	String	16	The customer code	No
Frequency	String	9	OneTime or Recurring	Yes
GratuityAmount	Double		The gratuity amount of the order. Two decimal places are required. If there is no gratuity amount, pass '0.00'. Format is 99999.99.	Yes
Invoice	Numeric	16	Invoice Number from the eCommerce site.	Yes
Memo	String	40	The <i>Memo</i> tag is the product name and version number of the developer's software.	Yes
MerchantID	Numeric	24	The Merchant's account or <i>MerchantID</i> that payments will be processed under.	Yes
OperatorID	String	16	Operator ID	No
PurchaseAmount	Double		The amount authorized for the transaction. Two decimal places are required. Must match the Amount that was sent in the <i>PreAuth</i> . Format is 99999.99.	Yes
TaxAmount	Double		The tax amount of the order. Two decimal places are required. If there is no tax amount, pass '0.00'. Format is 99999.99.	No
TerminalName	String	20	Terminal Name	No
Token	String	100	The token generated by a <i>PreAuth</i> transaction that replaces the credit card #. Can be used in subsequent transactions.	Yes

CreditPreAuthCaptureToken Example

```

TransactionService.TransactionService ts = new TransactionService.TransactionService();
TransactionService.CreditPreAuthCaptureToken preAuthCaptureRequest = new
TransactionService.CreditPreAuthCaptureToken()
{
    AcqRefData = "Kb8238953#20#20#20#20#20#20#20e00",
    AuthCode = "12345",
    AuthAmount = 9.00,
    CardHolderName = "Jen Brown",
    CustomerCode = "CUSTCODE1",
    Frequency = "OneTime",
    GratuityAmount = 1.00,
    Invoice = "12345",
    Memo = " MPS WebServer v1.0.0",
    MerchantID = "778825001",
    OperatorID = "",
    PurchaseAmount = 9.00,
    TaxAmount = ".62",
    TerminalName = "",
    Token = "J9yQ0+wyBWS7G1A7TarjDtqzxk7DEOHDMhAF EAASNB0uYA==",
};
string password = "vNxE:r_CZNaHmkb^";
TransactionService.CreditResponse response = ts.CreditPreAuthCaptureToken(preAuthCaptureRequest,
password);

```

3. CreditSaleToken

CreditSaleToken Input Parameters

Parameter Name	Type	Length	Description	Required
Address	String	8	Card Holder Billing Address. If provided will be used for Address Verification.	No
Cardholder Name	String	30	Customer Card Holder Name as it appears on card.	No
CustomerCode	String	16	The customer code.	No
CVV	Numeric	3 or 4	Card Verification Value from back of credit card. Used for card	No

			verification	
Frequency	String	9	OneTime or Recurring	Yes
Invoice	Numeric	16	Invoice Number from the eCommerce site.	Yes
Memo	String	40	The <i>Memo</i> tag is the product name and version number of the developer's software.	Yes
MerchantID	Numeric	24	The Merchant's account or <i>MerchantID</i> that payments will be processed under.	Yes
OperatorID	String	16	Operator ID	No
PartialAuth	Boolean	5	Set to true to allow partial authorization. Default is false .	No
PurchaseAmount	Double	8	The total amount of the order to charge the card holder. Two decimal places are required. Format is 99999.99.	Yes
TaxAmount	Double	8	The tax amount of the order. If there is no tax for the transaction, set to 0.00. Format is 99999.99.	Yes
TerminalName	String	20	Terminal Name	No
Token	String	100	The token generated by a <i>PreAuth</i> transaction that replaces the credit card #. Can be used in subsequent transactions.	Yes
Zip	String	5 or 9	Cardholder Zip. If provided, will be used for address verification.	No

CreditSaleToken Example

```

TransactionService.TransactionService ts = new TransactionService.TransactionService();
TransactionService.CreditSaleToken saleRequest = new TransactionService.CreditSaleToken()
{
    Token = "J9yQ0+wyBWS7G1A7TarjDtqzxk7DEOHDMhAF EAASNB0uYA==",
    Frequency = "Recurring",
    PurchaseAmount = 9.00,
    TaxAmount = .72,
    MerchantID = "778825001",
    Invoice = "12345",
    TerminalName = "",
    OperatorID = "",
    Memo = "Sale Test",
    Address = "4 Corporate Square",
    Zip = "30329",
    CardHolderName = "Jen Brown",
    CustomerCode = "cust1042",
    CVV = "123",
};
string password = "vNxER_CZNaHmkb^";
TransactionService.CreditResponse response = ts.CreditSaleToken(saleRequest, password);

```

4. CreditAdjustToken

CreditAdjustToken Input Parameters

Parameter	Type	Length	Description	Required
AuthCode	String	16	The Authorization code.	Yes
CardHolderName	String	30	The name on the card	No
CustomerCode	String	16	The customer code	No
Frequency	String	9	OneTime or Recurring	Yes
GratuityAmount	Double		The gratuity amount of the order. Two decimal places are required. If there is no gratuity amount, pass '0.00'. Format is 99999.99.	Yes
Invoice	Numeric	16	Invoice Number from the POS.	Yes
Memo	String	40	The <i>Memo</i> tag is the product name and version number of the developer's software.	Yes

MerchantID	Numeric	24	The Merchant's account or <i>MerchantID</i> that payments will be processed under.	Yes
OperatorID	String	16	Operator ID. Alphanumeric.	Yes
PurchaseAmount	Double		The amount of the order to charge the cardholder not including gratuity. Two decimal places are required. Format is 99999.99.	Yes
RefNo	String	16	The reference number returned to the developer in the response of the <i>Sale</i> or <i>PreAuthCapture</i> transaction.	Yes
TaxAmount	Double		The tax amount of the order. Two decimal places are required. If there is no tax amount, pass '0.00'. Format is 99999.99.	No
TerminalName	String	20	Terminal Name	No
Token	String	100	The token generated by a <i>Sale</i> or <i>PreAuthCapture</i> transaction that replaces the credit card #. Can be used in subsequent transactions.	Yes

CreditAdjustToken Example

```

TransactionService.TransactionService ts = new TransactionService.TransactionService();
TransactionService.CreditadjustToken AdjustRequest = new TransactionService.CreditAdjustToken()
{
    AuthCode = "12345",
    AuthorizeAmount = 9.00,
    CardHolderName = "Jen Brown",
    CustomerCode = "CUSTCODE1",
    Frequency = "OneTime",
    GratuityAmount = 1.00,
    Invoice = "12345",
    Memo = " Worldpay WebServer v1.0.0",
    MerchantID = "778825001",
    OperatorID = "Bob67",
    PurchaseAmount = 9.00,
    TaxAmount = ".62",
    TerminalName = "",
    Token = "J9yQ0+wyBWS7G1A7TarjDtqzxk7DEOHDMhAF EAASNB0uYA==",
};
string password = "vNxE:r_CZNaHmkb^";
TransactionService.CreditResponse response = ts.CreditAdjustToken(AdjustRequest, password);

```

5. CreditVoidSaleToken

CreditVoidSale Token Input Parameters

Parameter	Type	Length	Description	Required
AuthCode	String	16	The Authorization code	Yes
CardHolderName	String	30	The name on the card	No
Frequency	String	9	OneTime or Recurring	Yes
Invoice	Numeric	16	Invoice Number from the eCommerce site.	Yes
Memo	String	40	The <i>Memo</i> tag is the product name and version number of the developer's software.	Yes
MerchantID	Numeric	24	The Merchant's account or <i>MerchantID</i> that payments will be processed under.	Yes
OperatorID	String	16	Operator ID. Alphanumeric.	No
PurchaseAmount	Double		The total amount of the order to charge the card holder. Two decimal places are required. Format is 99999.99.	Yes
RefNo	String	16	The reference number returned to the developer in the response of the <i>Sale</i> or <i>PreAuthCapture</i> transaction or the <i>InvoiceNo</i> from	Yes

			a <i>PreAuth</i> for reversal.	
TerminalName	String	20	Terminal Name	No
Token	String	100	The token generated by a <i>PreAuth</i> , <i>PreAuthCapture</i> , or <i>Sale</i> transaction that replaces the credit card #. Can be used in subsequent transactions.	Yes

CreditVoidSaleToken Example

```
TransactionService.TransactionService ts = new TransactionService.TransactionService();
TransactionService.CreditVoidSaleToken saleRequest = new TransactionService.CreditVoidSaleToken()
{
    Token = "J9yQ0+wyBWS7G1A7TarjDtqzxk7DEOHDMhAF EAASNB0uYA==",
    Frequency = "OneTime",
    PurchaseAmount = 9.00,
    MerchantID = "778825001",
    Invoice = "12345",
    RefNo = "004",
    TerminalName = "",
    OperatorID = "Bob67",
    AuthCode = "36092B",
    Memo = "Void Sale Test";
    CardHolderName = "Jen Brown",
};
string password = "vNxE:r_CZNaHmkb^";
TransactionService.CreditResponse response = ts.CreditVoidSaleToken(saleRequest, password);
```

6. CreditReversalToken

CreditReversalToken Input Parameters

Parameter	Type	Length	Description	Required
AuthCode	String	16	The Authorization code	Yes
CardHolderName	String	30	The name on the card	No
Frequency	String	9	OneTime or Recurring	Yes
Invoice	Numeric	16	Invoice Number from the POS.	Yes
Memo	String	40	The <i>Memo</i> tag is the product name and version number of the developer's software.	Yes
MerchantID	Numeric	24	The Merchant's account or <i>MerchantID</i> that payments will be processed under.	Yes
OperatorID	String	16	Operator ID. Alphanumeric.	No
PurchaseAmount	Double		The total amount of the order to charge the card holder. Two decimal places are required. Format is 99999.99.	Yes
RefNo	String	16	The reference number returned to the developer in the response of the <i>Sale</i> or <i>PreAuthCapture</i> transaction or the <i>RefNo</i> filler from the <i>PreAuth</i> .	Yes
TerminalName	String	20	Terminal Name	No
Token	String	100	The token generated by a <i>PreAuth</i> transaction that replaces the credit card #. Can be used in subsequent transactions.	Yes
AcqRefData	String	200	Acquirer Reference Data that was received in the response of <i>VerifyPayment</i> , <i>CreditSaleToken</i> , <i>CreditPreAuthToken</i> , or <i>CreditPreAuthCaptureToken</i> transaction.	Yes
ProcessData	String	200	Process Data that was received in the response of <i>VerifyPayment</i> , <i>CreditSaleToken</i> , <i>CreditPreAuthToken</i> , or <i>CreditPreAuthCaptureToken</i> transaction.	Yes

CreditReversalToken Example

```
TransactionService.TransactionService ts = new TransactionService.TransactionService();
```

```
TransactionService.CreditReversalToken saleRequest = new TransactionService.CreditReversalToken()
{
    Token = "J9yQ0+wyBWS7G1A7TarjDtqzXk7DE0HDMhAF EAASNB0uYA==",
    Frequency = "OneTime",
    PurchaseAmount = 9.00,
    MerchantID = "778825001",
    Invoice = "12345",
    RefNo = "004",
    TerminalName = "",
    OperatorID = "Bob67",
    AuthCode = "36092B",
    Memo = "Void Sale Test";
    CardHolderName = "Jen Brown",
    AcqRefData = "Kb8238953#20#20#20#20#20#20#20e00",
    ProcessData = "|17|210100700000",
};
string password = "vNxE:r_CZNaHmkb^";
TransactionService.CreditResponse response = ts.CreditReversalToken(saleRequest, password);
```

7. CreditReturnToken

CreditReturnToken Input Parameters

Parameter	Type	Length	Description	Required
CardHolderName	String	30	The name on the card	No
Frequency	String	9	OneTime or Recurring	Yes
Invoice	Numeric	16	Invoice Number from the eCommerce site.	Yes
Memo	String	40	The <i>Memo</i> tag is the product name and version number of the developer's software.	Yes
MerchantID	Numeric	24	The Merchant's account or <i>MerchantID</i> that payments will be processed under.	Yes
OperatorID	String	16	Operator ID. Alphanumeric	No
PurchaseAmount	Double		The total amount of the order to charge the card holder. Two decimal places are required. Format is 99999.99.	Yes
TerminalName	String	20	Terminal Name	No
Token	String	100	The token generated by a <i>PreAuth</i> transaction that replaces the credit card number. Can be used in subsequent transactions.	Yes

CreditReturnToken Example

```
TransactionService.TransactionService ts = new TransactionService();
TransactionService.CreditReturnToken request = new TransactionService.CreditReturnToken()
{
    Token = "J9yQ0+wyBWS7G1A7TarjDtqzxk7DEOHDmHAF EAASNB0uYA==",
    Frequency = "OneTime",
    PurchaseAmount = 9.00,
    MerchantID = "778825001",
    Invoice = "12345",
    TerminalName = "",
    OperatorID = "Bob67",
    Memo = "Void Sale Test";
    CardHolderName = "Jen Brown",
};
string password = "vNxE:r_CZNaHmkb^";
TransactionService.CreditResponse response = ts.CreditReturnToken(request, password);
```

8. CreditVoidReturnToken

CreditVoidReturnToken Input Parameters

Parameter	Type	Length	Description	Required
AuthCode	String	16	The Authorization code	Yes
CardHolderName	String	30	The name on the card	No
Frequency	String	9	OneTime or Recurring	Yes
Invoice	Numeric	16	Invoice Number from the eCommerce site.	Yes
Memo	String	40	The <i>Memo</i> tag is the product name and version number of the developer's software.	Yes
MerchantID	Numeric	24	The Merchant's account or <i>MerchantID</i> that payments will be processed under.	Yes
OperatorID	String	16	Operator ID. Alphanumeric.	No
PurchaseAmount	Double		The total amount of the order to charge the card holder. Two decimal places are required. Format is 99999.99.	Yes
RefNo	String	16	The reference number of the transaction returned in the original PreAuthCapture or Sale response.	Yes
TerminalName	String	20	Terminal Name	No
Token	String	100	The token generated by a <i>PreAuth</i> transaction that replaces the credit card number. Can be used in subsequent transactions.	Yes

CreditVoidReturnToken Example

```
TransactionService.TransactionService target = new TransactionService.TransactionService();
```

```

CreditVoidReturnToken voidReturnToken = new CreditVoidReturnToken()
{
    Memo = "CreditVoidReturnToken Test",
    Token = "J9yQ0+wyBWS7G1A7TarjDtqzxk7DEOHDMhAFE AASNB0uYA==",
    Frequency = "OneTime",
    PurchaseAmount = 9,
    MerchantID = "778825001",
    Invoice = "12345",
    TerminalName = "",
    OperatorID = "Bob67",
    AuthCode = "123456",
    CardHolderName = "Jen Brown",
    RefNo = "004",
};
string password = "vNxE:r_CZNaHmkb^";

CreditResponse response = target.CreditVoidReturnToken(voidReturnToken, password);

```

HostedCheckout CreditResponse - Returned Values

These are the standard values returned for all of the above token transactions:

CreditPreAuthToken; CreditPreAuthCaptureToken; CreditSaleToken; CreditReturnToken; CreditVoidSaleToken, CreditVoidReturnToken; and CreditReversalToken

Parameter	Type	Length	Description
Account	String	12	The masked account number used for the transaction.
AcqRefData	String	200	Acquirer Reference Data
AuthCode	String	16	Authorization code.
AuthorizeAmount	Numeric	8	The amount that the transaction was authorized for. If this amount is less than the <i>PurchaseAmount</i> field, then payment in full has not been received and additional tender will need to be requested.
AVSResult	String	40	The result of the AVS check. See details later in this guide.
BatchNo	String	6	Batch # of transaction if applicable
CardType	String	40	Type of card used to make the payment

CvvResult	String	40	Result of the CVV check. Additional details later in this guide. M =match N =no match P =not processed S =CVV should be on card but merchant indicated it is not present (Visa/Discover only) U =Issuer is Not Certified, CID not checked (AMEX only)
GratuityAmount	Numeric	8	Tip (gratuity) amount (with 2-place decimal – e.g., 5.00)
Invoice	Numeric	16	Invoice # of the transaction
Message	String	unlimited	Transaction status message or concatenated validation error messages.
PurchaseAmount	Numeric	8	Purchase price (with 2-place decimal – e.g., 29.25)
RefNo	String	16	Transaction Reference Number
Status	String	19	The status of the transaction. The Message field contains additional information. Approved. The transaction was approved. Declined. The transaction was declined. Error. A transaction processing error occurred. AuthFail. Authentication failed for MerchantID/password. MercuryInternalFail. An error occurred internal to Worldpay. ValidateFail. Validation of the request failed. See Message for validation errors.
Token	String	100	The token generated by the transaction that replaces the credit card #. Can be used in subsequent transactions.
ProcessData	String	200	ProcessData
RecurringData	String	40	<blank> Recurring Recurring 00 Recurring 00 Recurring 00 314282069480098 Installment 00 314282069480098 Unscheduled 00 314282069480098 CardHolderInitiated 00 314282069480098

Gift/StoreCard transactions also use Transaction Web Services

Gift and StoreCard transaction and sent via a call to Transaction Web Service.

Production and Development-Test Gift Servers

HostedCheckout Gift URL's	
Development URL:	https://hc.mercurycert.net/tws/transactionservicegift.asmx
Production URL:	https://hc.mercurypay.com/tws/transactionservicegift.asmx

Notes

- Gift server names are different from those used for credit and must be explicitly specified.
- The links above will take you to the libraries of supported transactions.

TransactionServiceGift

The following operations are supported. For a formal definition, please review the [Service Description](#).

- [GiftBalance](#)
 - [GiftIssue](#)
 - [GiftReload](#)
 - [GiftReturn](#)
 - [GiftSale](#)
 - [GiftVoidIssue](#)
 - [GiftVoidReload](#)
 - [GiftVoidReturn](#)
 - [GiftVoidSale](#)
- For a Gift sale transaction in Transaction Web Services to emulate a standard **NoNSFSale** TranCode, the transaction request must be sent with the addition of a `<PartialAuth>true</PartialAuth>`.

Regulatory Compliance and Recommended Best Practices for eCommerce Solutions

Card Data Storage Using MToken

Ecommerce merchants that ship merchandise beyond a 24-hour period must support *PreAuth/PreAuthCapture* transaction types and are required to implement tokenization support for secure storage and subsequent usage of card data. These integrations need to store and maintain token data (*RecordNo*, *Frequency*) and *AuthCode*, *AcqRefData* appropriately for subsequent use. Accordingly, *CVV Data* used for the *PreAuth* must not be retained for any length of time.

Required Transactions

HostedCheckout integrations must support *PreAuth*, *PreAuthCapture*, and *Return* transaction types (although in integrations where there is an immediate exchange of merchandise within 24 hours of purchase, *Sales* and *Returns* transaction types can be substituted).

<Memo Tag> - Software Product and Version

The *Memo* data element should be coded dynamically in all transaction requests; it is used to identify the software model and version. This use of the *Memo* tag is a certification requirement.

Fraud Prevention of CVV and AVS data

For fraud protection in a card-not-present environment, and for the best qualifying processing rates, eCommerce credit integrations are highly recommended to include AVS (Address/Zip Verification) and CVV2/CVC2 data in the transaction request. Though a failed AVS or CVV match will not always result in a declined response, it is important to code in options for proceeding with the transaction in the event of an AVS/CVV mismatch.

Website Requirements

Before a business can be approved to process eCommerce transactions, it must have a valid website with the requirements listed below. Should the products or services offered through an eCommerce merchant account change, Worldpay must be notified immediately. A merchant that currently has a retail or MOTO account with Worldpay must submit an entirely separate application for eCommerce. Worldpay issues eCommerce merchants unique Merchant ID numbers upon underwriting approval.

The following eCommerce policies and procedures have been established to comply with Visa[®], MasterCard[®], Discover[®] and Worldpay regulations:

1. **Product(s) Sold.** Ecommerce merchants must maintain a valid website with complete descriptions of the products and services sold. All federal and state laws apply in addition to the card association regulations regarding all products and services sold.

2. **Security Policy.** Ecommerce merchants must submit transactions for authorization and settlement through a secure online gateway. The security protocols used to protect a customer's information must be disclosed.
3. **Privacy & Return Policy.** The website must contain a return, refund and privacy policy. Terms of Use or Terms and Conditions must also be properly disclosed on the website.
4. **Delivery Policy.** Ecommerce merchants set their own policies and restrictions regarding delivery of goods. Any restrictions on delivery must be clearly stated on the website.
5. **Customer Service Contact.** Contact information for your business must be easily accessible to customers. It must show a physical address along with an electronic mail address and telephone number.
6. **Card Acceptance Brand Marks.** Full-color brand marks are required by Visa®, MasterCard® and Discover®. Discover partners, JCB and UnionPay, must also be displayed. The American Express® logo must be displayed if accepted.

General eCommerce Best Practices

1. **Shipping and Delivery.** The shipping method, time frame and delivery procedures should be clearly stated. After the payment has been accepted, the product must ship immediately. The customer cannot be charged unless the product is in inventory and ready to ship.
2. **Transaction Receipt.** A transaction receipt via electronic mail should be provided to the customer and must include your business name, web address, contact information and the terms and conditions of the sale. Always keep the transaction receipt, shipping record (UPS, Airborne, FedEx or USPS) and proof-of-delivery on file. This information will need to be presented in the event of a chargeback.
3. **Transaction Verification.** AVS (the Address Verification System) should be used to validate the billing address of the cardholder. Use the Card Verification Value (CVV2) to further validate the transaction.
4. **Suspicious Transactions.** It is highly recommended to develop in-house policies and procedures for handling irregular or suspicious transactions and provide appropriate training for sales staff. Any order that seems suspicious should be further investigated. Contact the cardholder for verification. Orders that fail AVS or CVV2 validation should be flagged for research. If an order is high-risk or possibly fraudulent, void the transaction immediately. Shipping orders that are not validated by AVS or CVV2 could result in chargebacks and loss of merchandise. All eCommerce merchants should be extremely suspicious of international orders. Some geographic areas have a higher frequency of fraud and black market stolen credit card information.
5. Additional policies and procedures should be put into place to monitor the following types of orders:
 - Rush orders,
 - Random orders (orders not specifying color, size, etc.),
 - Shipping address not the same as billing address,
 - Shipments to post office boxes or business addresses,
 - Transactions associated with anonymous email addresses or
 - Orders from internet addresses at free email services.
6. **Acknowledgement of Policies.** Policies should require a cardholder to click to accept them. Not doing so leaves merchants open to claims by the cardholder that they were not aware of the merchant's policies at the time of the transaction.

Handling Errors in eCommerce Transactions

General Response Categories	TextResponse	Suggested Displayed Response	Remarks
--------------------------------	--------------	---------------------------------	---------

General Response Categories	TextResponse	Suggested Displayed Response	Remarks
Invalid Credentials/MID Merchant ID not Found Response due to configuration error	INVLD MERCH ID, INVLD CREDENTIALS MERCHANT ID NOT FOUND, MerchantID not found (Gift) Invalid TranType for Prepaid Server	"Web Site Application Error. Please Contact ###-###- ####"	Send corresponding email alert notification to Gateway admin/merchant.
Communication, Connectivity and Socket Response Errors	EDC UNAVAILABLE, DB UNAVAIL, TIMEOUT ON RESPONSE, TIMEOUT WAITING FOR RESPONSE, MAX CONNECTION ERROR, SOCKET CONNECTION FAILED, NO CONNECTION TO ANY SERVER, HOST UNAVAIL, ISSUER UNAVAIL	"We're sorry. We are experiencing a temporary connectivity error. Please try again."	
Transaction Level: Approvals	AP, AP*, AP NEW INFO APPROVED STANDIN AP-NOT CAPTURED	"Approved, Successful!"	
Transaction Level: Declines	DECLINED, OTHER NOT EXCEPTED, PIC UP, EXCEEDS MAX AMT, DECLINE TRY LATER	"We're sorry, your transaction has declined."	
Transaction Level Declines: Invalid Data Errors	INVLD ACCT, INVALID EX DATE, INVALID REFERENCE NUMBER, INVLD TRAN CODE, INVALID FIELD: ____.	"Your account information has been entered incorrectly. Please try again".	Maximum 3 retries.
Transaction Level Declines: Duplicate Transaction	(AP*-- signifies duplicate logic detected duplicate and did not charge the card twice) AP DUPE	"Duplicate transaction data has been entered. Transaction declined."	
Transaction Level Declines: Call Referrals	CALL AE, CALL DISCOVER, CALL ND, CALL XXXX	"Your transaction has declined: please contact us at ###-###-#### to complete your transaction."	
Verification Result codes: CVV	M =CVV Match N =CVV no Match P =Not Processed DECLINED-CV2 FAIL	"Your information has been entered incorrectly. Please verify and try again."	Maximum 3 retries, or Auto-void upon CVV fail response of N, or base void on established criteria as in >\$300.00 Note Worldpay recommends coding provisions for Merchant level configurable

General Response Categories	TextResponse	Suggested Displayed Response	Remarks
			attributes.
Verification Result codes: AVS	(AVS Varies by card brand, all single digit alpha characters)	"Your information has been entered incorrectly. Please verify your retry."	Maximum 3 retries
Settlement Response Codes	OK, INV BAL/SETL, MUST BAL NOW		Merchant/Shopping Cart/Gateway level only
Gift Card (PrePaid) Error Responses	Invalid Account Number Account not Issued Account Expired Account Already Issued Issuance not Authorized Insufficient Account Balance Mag Stripe Error-Hand Key Amount Exceeds Maximum Return Not Allowed Account Deactivated Number of Returns on Card over Limit Account in Use Could not Void Issue	"We're Sorry, your transaction has declined: xxx."	All gift transaction error verbiage may be passed as-is, replacing the xxx.

Additional HostedCheckout/eCommerce Details

Protection against Duplicate Transactions

Worldpay's duplicate logic screens for unintended duplicates within a current open merchant batch. This logic incorporates the following variables:

- Card number
- Amount
- Transaction Type
- InvoiceNo

Avoiding duplicate transactions requires implementing an always-incrementing invoice logic. Use the Worldpay *InvoiceNo* field as outlined below.

The one exception to this rule is in the event of a time-out response where the authorization response is unknown. In this case, the merchant site should resubmit the exact same transaction request with the same amount, same card number, and same invoice number.

- If the transaction was not previously authorized, the response data will include the standard approved or declined processor's response.
- If the transaction was previously authorized, the response data will include a *CmdStatus* of APPROVED with the additional *TextResponse* designation of AP* (reported as Approved/AP DUPE). The second transaction will not be charged.

When Worldpay's duplicate checking recognizes the same card number, same amount, same transaction type and same *InvoiceNo* within the same open batch, the duplicate checking assumes that the second transaction is an unintended duplicate. The original authorization holds and the card is not charged twice. Worldpay responds by replacing the duplicated request with the original transaction information—*AuthCode*, *AcqRefData*, *ProcessData*, *RefNo*, CVV and AVS Result data are all passed from the original response. Worldpay passes the approval on to the second duplicate transaction without charging the card a second time.

Important InvoiceNo and Duplicate Checking Specifications

- **InvoiceNo specification:** Numeric only, incrementing 1-16 char in length BUT the right-ten characters must be unique. The right ten are used for duplicate logic checking. No alpha characters, special characters or dashes.
 - Although there may be additional local uses for this field, developers are encouraged to strictly adhere to these parameters and to use this field as an always-incrementing local transaction number with the one rare exception of a time out.
- **Duplicate logic will fail** if the application never sends an invoice number, always sends the same invoice such as "NOINVOICENO", or sends a zero-value: "0000."
- **If it is required to charge the same card again for the same amount**, the application must submit a second transaction with a new, unique invoice number.
- **The Duplicate Override tag is not required for U.S. processing.** Duplicate Override is used in Canadian processing as the duplicate logic only checks card number and amount in any current batch.

PrePaid Partial Authorization and Real-Time Reversal Support

Real-time reversal support has been added to the HostedCheckout platform for better transaction handling in an eCommerce environment. Though not required for card-not-present environments, this support allows *PreAuth*, *PreAuthCapture*, and *Sale* transactions to be reversed, returning funds in real-time to the cardholder's account as long as their issuing bank supports it.

The **<AcqRefData>** and **<ProcessData>** values returned in the response will need to be stored for use with real-time reversal functionality.

Real-Time Reversal Support using *CreditReversalToken* + **<AcqRefData> and **<ProcessData>****

In the event a *PreAuth* needs to be reversed, a real-time reversal will be sent.

TranCode is **CreditReversalToken**

- *AcqRefData* and *ProcessData* from the original transaction response needs to be provided in the Reversal request. The *AcqRefData* and *ProcessData* fields are what trigger the *Reversal* to work in real time.
- If reversing a *PreAuth*, *RefNo* value should equal the *Invoice* value from the original *PreAuth*.
- If successful, the *CMDStatus* will be **AP** and the *TextResponse* will be **REVERSED**.
- If an AP/REVERSED Status message is not returned in the reversal request, a *CreditVoidSaleToken* should be immediately sent without the *AcqRefData* and *ProcessData* tags.

Corporate Card Support: Level II data – Tax and CustomerCode

For the lowest rates when processing business/commercial (corporate) cards it is necessary to supply **Tax** and **CustomerCode** fields with all credit transactions. The use of this data lowers rates for retail and supermarket merchants, but **not restaurants**. Level II Data also has nothing to do with Track2 Data.

- **Tax:** Put the **<Tax>9.99</Tax>** tag in the *Amount* section of the XML request. The value format is numeric with 2 decimal places. Since the application already calculates the correct tax amount for each transaction, simply parse the actual tax amount from the application into the Tax tag. The Purchase amount tag will be the total charge, including tax. The tax field breaks out the tax amount for Level II reporting purposes only.
- **CustomerCode:** Submit **<CustomerCode>99999</CustomerCode>** in the *TranInfo* section. The value should be the actual PO number, the merchant's zip code, or the transaction invoice number. This is sufficient to meet the Level II reporting requirements.

Using Card Security Codes (CVV) and Address Verification Data (AVS)

HostedCheckout provides the ability to enable or disable CVV and AVS (address and zip code fields) on the HostedCheckout page. **It is highly recommended to implement these fraud protection measures!**

- **For CVV:** values are **on** and **off**. Default is **ON**.
- **For AVS:** Values are **on**, **zip** and **off**. Default is **OFF**.

For all eCommerce transactions, Card Security Codes (CVV/CVV2/CID) and Address Verification (AVS) data should be supplied in the transaction request. Issuing bank result codes are returned in the transaction response along with the card issuer authorization. See **Appendix 9** for fine-tuning fraud prevention screening based on these result codes. In the event of unauthorized credit card activity, it is critical to implement settings that enable merchants to automatically respond to mis-matched or questionable card security codes and address verification results.

- It is important to note that CVV and AVS results are returned *independently* from the authorization process. It is common for an issuer to **approve** a transaction but also send a **"no match"** CVV and/or AVS response. Regardless if the issuing bank sends back an approval, if the CVV or AVS data indicate a no-match or an otherwise questionable, potentially fraudulent result, POS applications should communicate the result along with the authorization response AND any applicable corrective measures.
- The appropriate corrective measure is determined by the merchant's environment, amount of the transaction, how often the card is being used and, if applicable, any shipping-billing address non-match. In other words, merchants can only prevent fraud and protect themselves from chargebacks if the POS application they are using supplies them the features to do so.
- General "rules of thumb" actions are provided in **Appendix 9**, but per card brand regulations, the variables that determine the action are relative to the merchant environment. As such, AVS and Card Security Codes are important tools used to help make business decisions on whether to proceed with a sale or delivery of goods and services. In all cases of questionable or partial CVV or AVS results where the transaction is APPROVED by the issuing bank, your POS application should build in configurable merchant settings to:
 1. Prompt the non-match to the end user with "Your information has been entered incorrectly. Please verify your retry."
 2. Allow for the option to reject the transaction immediately by running a Reversal-VoidSale
 3. Allow the option to both process a Reversal-VoidSale and to allow for re-entry of the non-matched data
 4. Allow the option to accept the transaction authorization as-is and by doing so, accepting the risk of the potential fraudulent transaction.

Card Verification Data (CVV/CVC2/CID)

Card verification data is referred to by several different acronyms: **CVV2/CVC2** (Cardholder Verification Value)/**CID** (Amex and Discover). It is a three- or four-digit code found on the back of a credit card (for Amex it is on the front of the card) that is used in eCommerce transactions to determine that the card is physically with the customer who is placing the order, since the code can only be found on the card itself.

Below are some rules of thumb to follow for the CVV Code received in the transaction response:

CVV Code	Definition	Explanation	Recommended Action
M	Match	The card security code supplied matches the value on the card.	Complete Transaction
N	No Match	The card security code supplied does not match the value on the card.	Reverse/Void and retry with updated information
P	Not Processed	The request was not processed.	Reverse/Void then retry
U	Not Supported	The verification service is unavailable or issuer does not support this service.	Evaluate, then Reverse/Void and ask for another form of payment or complete transaction

Note: To avoid phishing threats for CVV data it is recommended to allow no more than three attempts that return a non-matching response.

Address Verification Service (AVS)

Address Verification (AVS) results codes are determined by the card's BILLING five or nine -digit postal code/ZIP and Street Address. The address verification service compares billing address information with what is on record at the issuing bank. The *AVSResult* Code is returned to the POS in the transaction response but is independent of the transaction authorization. Use of AVS impacts processing rates and aids in the prevention of chargeback retrieval requests—AVS is not, however, usable as compelling evidence in the event of a chargeback. AVS is highly recommended to be implemented any time there is a business need for manual/keyed input in card present merchant locations and especially in card-not-present (MOTO and eCom) environments.

The AVS request data elements include *AVSAddress* and *AVSZip*. Example:

```
<mer:AVSAddress>150 Mercury Village</mer:AVSAddress>
<mer:AVSZip>81301</mer:AVSZip>
```

The example above shows how this information is captured by HostedCheckout Payment page.

- **Format:** Address: Alphanumeric. Length: min 1 max 20. Zip: for the U.S: numeric, for Canada: alphanumeric. Length: min 5 max 9
- For all card-not-present environments, BOTH postal code/ZIP and ADDRESS are recommended for the greatest fraud protection.
- Addresses that start with alpha characters, such as "PO Box", or that include "Suite" or "Apt No" may or may not be allowable, depending on the Issuer's records.

It is possible for the transaction to be approved and at the same time respond with a "No Match" AVS response. **A transaction will NEVER decline based on an AVS mismatch.** Therefore, it is critical for the POS to be able to interpret the response codes and enable the merchant to respond to mismatched or questionable AVS results. Interpreting the response is driven by merchant's business need, limited by such things as transaction amount and the merchant's risk thresholds.

Below outlines basic guidance when an AVS Result Code is received in the transaction response. The recommended actions are based on two levels of matching: postal code/ZIP + address match exactly, or there is a partial match of postal code/ZIP or address. See **Appendix 9** for additional details on AVS.

AVS Code	Definition	Result Explanation or System Message	Recommended Actions
Y	Match	Street address and ZIP Code match	Exact match: complete transaction.
N	No Matches	Street address and ZIP Code do not match	No Match: Reverse and process a second authorization with updated information; do not allow multiple "phishing" attempts—use max of three tries.

Z	Zip Match	ZIP Code matches, but Street address does not match	Partial match: Evaluate based on business criteria; Reverse and retry with updated information or complete transaction
A	Address Match	Street address matches, but ZIP Code does not match	Partial match: Evaluate based on business, confirm cardholder ID; Reverse and retry with updated information or complete transaction
R	Retry	Retry / system unavailable	Issuer system message. Reverse the transaction and then Retry; if a second Retry is received, reverse and consider a second form of payment
U	Unavailable	Street and ZIP Code information unavailable or unverifiable	System message indicating AVS data is unavailable or un-verifiable. Evaluate based on business criteria; Reverse, check cardholder ID, ask for another form of payment or complete transaction
E	Error	AVS not supported or there was an error retrieving AVS data	System message indicating an error in retrieving AVS data or the data is un-verifiable. Evaluate based on business criteria; Reverse, check cardholder ID, ask for another form of payment or complete transaction

HostedCheckout URL Quick Reference – HostedCheckout Service: HCSservice

Development URL:	https://hc.mercurycert.net/hcws/hcservice.asmx
Production URL:	https://hc.mercurypay.com/hcws/hcservice.asmx

Transaction Functions:

- InitializePayment
Step 1 – Used to specify the parameters of a transaction session. Returns the PaymentID used for the session.
- VerifyPayment
Step 3 – After the user has submitted the payment, the VerifyPayment call is used to determine the final status of the transaction session and also returns the full transaction response data.

Cascading Style Sheet Functions:

- UploadCSS
Uploads a CSS file to the MerchantID being used.
- DownloadCSS
Returns the CSS file currently loaded to the MerchantID being used.
- RemoveCSS
Removes the CSS file currently loaded to the MerchantID being used.

HostedCheckout Payment Page URLs

Redirect

The browser redirects the user from the checkout page to the HostedCheckout payment page. Upon completion, the user is redirected back to the original site.

Standard Redirect Page for **eCommerce**:

Development URL:	https://hc.mercurycert.net/Checkout.aspx
Production URL:	https://hc.mercurypay.com/Checkout.aspx

Standard Redirect Page for **POS**:

Development URL:	https://hc.mercurycert.net/CheckoutPOS.aspx
Production URL:	https://hc.mercurypay.com/CheckoutPOS.aspx

Mobile Optimized Redirect Page for **eCommerce**:

Development URL:	https://hc.mercurycert.net/mobile/mCheckout.aspx
Production URL:	https://hc.mercurypay.com/mobile/mCheckout.aspx

Mobile Optimized Redirect Page for **POS**:

Development URL:	https://hc.mercurycert.net/mobile/mCheckoutPOS.aspx
Production URL:	https://hc.mercurypay.com/mobile/mCheckoutPOS.aspx

iFrame

The HostedCheckout iFrame allows the user to stay on the integrated checkout page - the Hosted Payment page is displayed within an iFrame.

Standard iFrame for **eCommerce**:

Development URL:	https://hc.mercurycert.net/CheckoutiFrame.aspx
Production URL:	https://hc.mercurypay.com/CheckoutiFrame.aspx

Standard iFrame for **POS**:

Development URL:	https://hc.mercurycert.net/CheckoutPOSiFrame.aspx
Production URL:	https://hc.mercurypay.com/CheckoutPOSiFrame.aspx

Mobile Optimized iFrame for **eCommerce**:

Development URL:	https://hc.mercurycert.net/mobile/mCheckoutiFrame.aspx
Production URL:	https://hc.mercurypay.com/mobile/mCheckoutiFrame.aspx

HostedCheckout TransactionService – Admin and Token calls

Development URL:	https://hc.mercurycert.net/tws/transactionsservice.asmx
Production URL:	https://hc.mercurypay.com/tws/transactionsservice.asmx

Administrative Functions

- BatchSummary
- BatchClose
- BatchClear

MToken Transactions using TransactionWebService

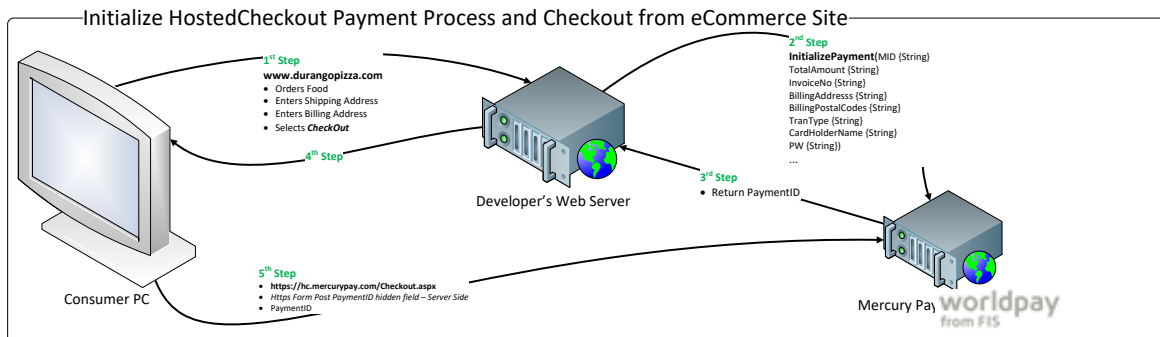
A Token representing credit card PAN data is returned in the VerifyPayment response for a transaction processed using the HostedCheckout payment page. Once your software has a token, follow-up transactions on that card can be processed using the token through the HostedCheckout TransactionService without the need for the hosted payment page to be displayed.

- CreditReversalToken
- CreditAdjustToken
- CreditReturnToken
- CreditVoidReturnToken
- CreditSaleToken

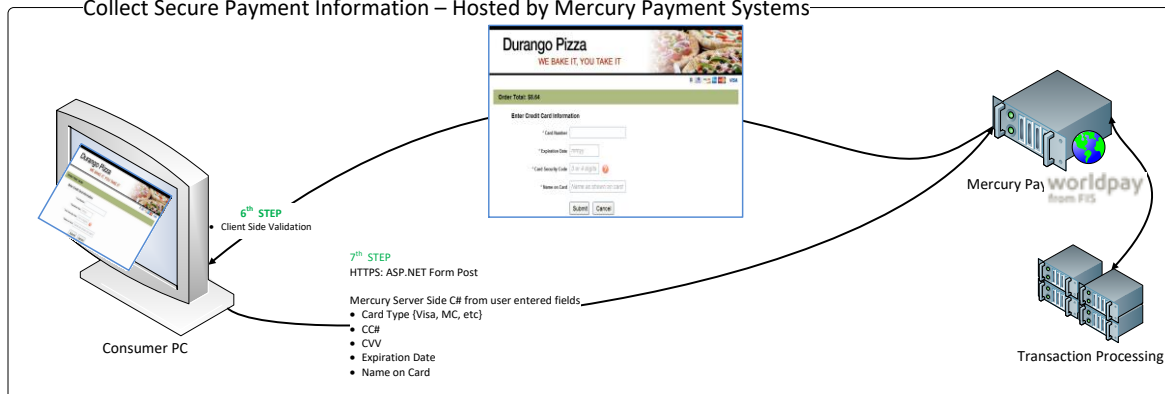
- CreditPreAuthToken
- CreditPreAuthCaptureToken

Alternately, these Token functions can be supported using our direct SOAP or REST/JSON Web Service API's. The use of either of these APIs for token transactions will required the use of a Web Service password, in addition to the HostedCheckout password.

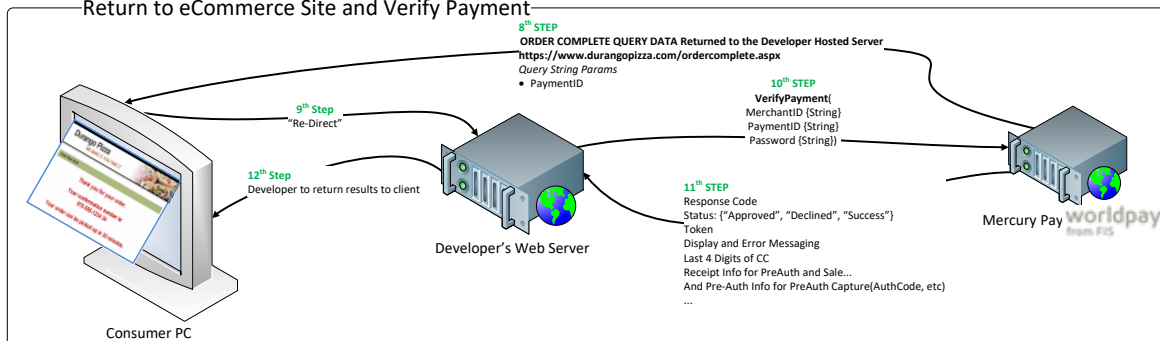
HostedCheckout Payment Process and Checkout from eCommerce Site



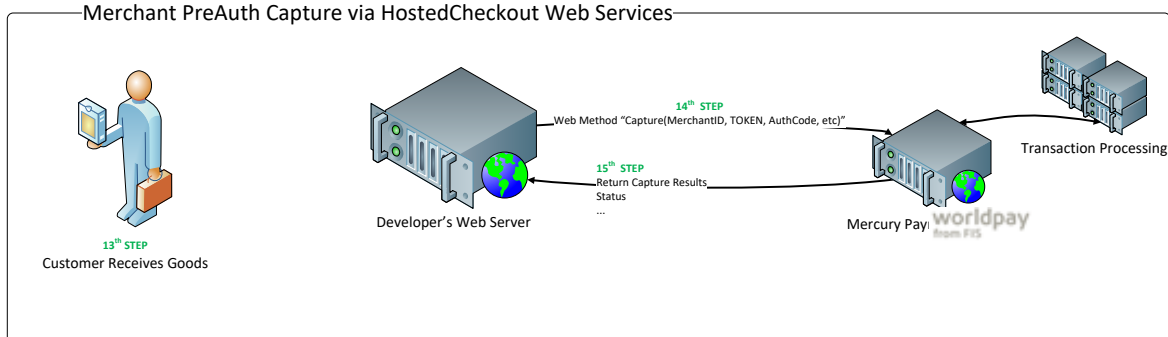
Collect Secure Payment Information – Hosted by Mercury Payment Systems



Return to eCommerce Site and Verify Payment

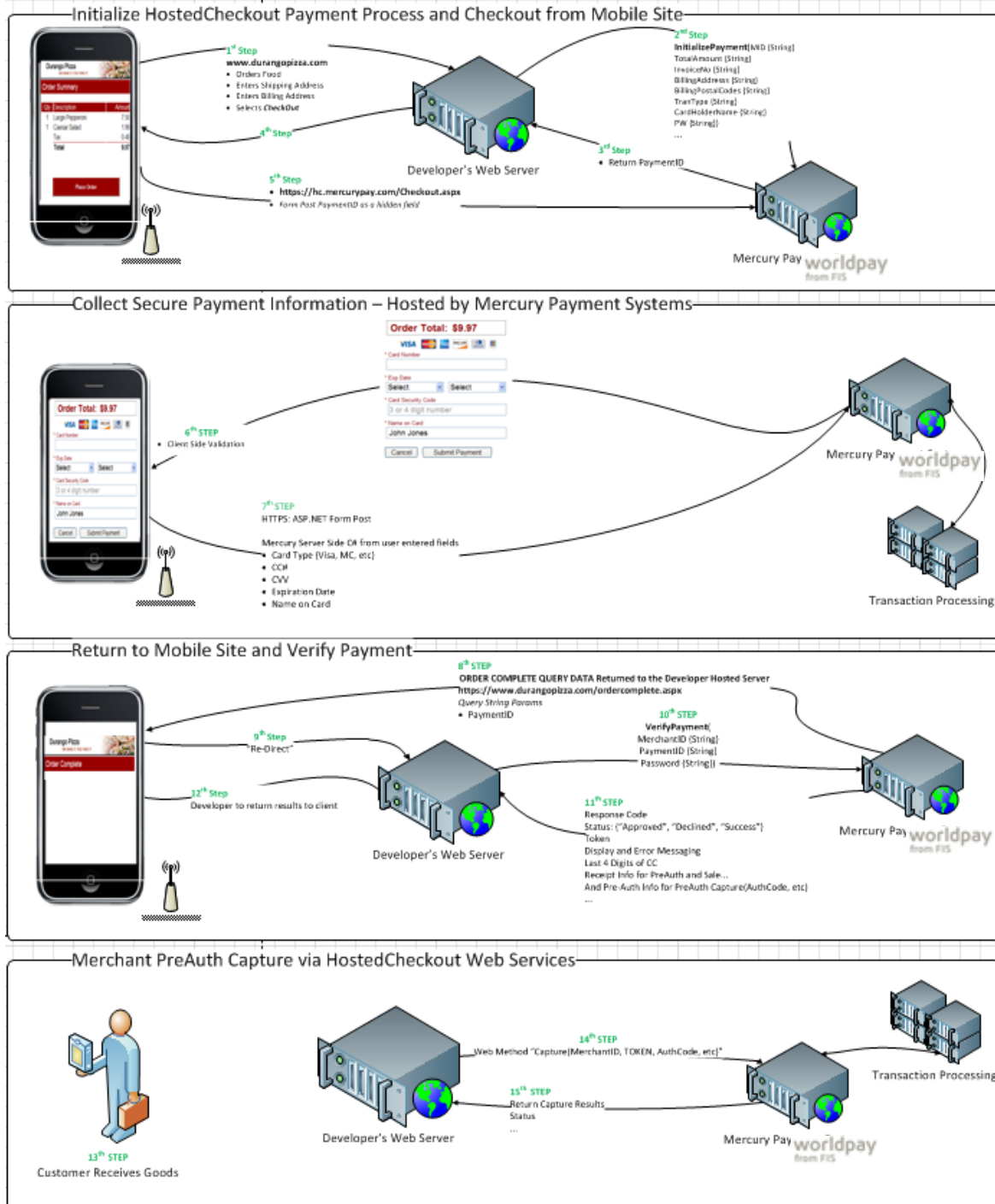


Merchant PreAuth Capture via HostedCheckout Web Services



HostedCheckout Mobile Data Flow

HostedCheckout Mobile - Logical Data Flow Version 3.0 2011



HostedCheckout Card-on-File/Credentials-on-File (COF)

What is a Credential and when is it a "Credential on File" (COF)?

Per card brand regulations, "credentials" are defined as Primary Account Number (PAN) or a token created by the Network (e.g., Visa, Mastercard) or processor (MToken). When credentials are stored by a merchant to process future transactions for the cardholder, additional details identifying the COF transaction are now required. For the MercuryPay HostedCheckout application, the new credentials on file requirements come into play whenever an MToken is stored for subsequent use.

Subsequent use was previously defined more generally and driven by the token frequency used in the transaction, OneTime or Recurring. With the card brand's new interpretation of COF, transactions using a stored token will need to be defined as an installment, recurring, cardholder initiated or unscheduled COF transaction. Details on the definition and use of each of these new COF types of payments are explained below.

Examples of COF include:

1. Shopping Cart checkout where online retailer stores the cardholder info.
2. Merchants that use a staged Digital Wallet, for example PayPal, for full or partial prepayment.
3. Wallet application with an agreed upon "replenish amount" when balance amount goes below a specific amount.
4. Hotel- cardholder has a membership profile with the hotel and provides their card number for future reservations.
5. Drug store in-person sale and cardholder uses QR code to link consumer to their profile with the merchant and uses stored credential.
6. Any Recurring, Installment, Unscheduled COF is always COF.

What is NOT considered a Stored Credential on File?

The COF requirements do not apply when the credential is used to complete a single transaction or purchase. This also includes purchases that encounter shipping delays and backorders, as for example, when multiple items are purchased from an e-commerce retailer and one item is backordered. The merchant will submit subsequent transactions to fulfill the entire order. Although a returned MToken is used, credentials are not stored as required for installment, recurring, cardholder initiated or unscheduled transaction; if the Cardholder returns to this site the cardholder would need to enter their payment credentials to place the order.

Other examples which are not considered COF include:

1. Apple Pay Samsung Pay, Android Pay Contactless
2. Apple Pay Samsung Pay, Android Pay In-app or e-Commerce website. Pass-through wallets such as Apple Pay, Samsung Pay, and Android Pay are not considered credential on file whether accepted as contactless or through a merchant website or merchant app.

Note: This does not apply if the cardholder instructs the merchant to store their payment credential during an in-app or e-commerce transaction using the payment credential from their Apple Pay or Android Pay. This would be considered initial storage of a payment credential.)

3. Hotel- cardholder provides the payment credential to cover charges (e.g., hotel stay, incidentals such as food) associated to that specific reservation only

Recurring, Installment, Unscheduled and Cardholder Initiated

Credentials on File Initialize Payment Input Parameter

Parameter	Type	Length	Description	Required
RecurringDataRequest	String	0-20	Valid RecurringDataRequest content <blank> Recurring Installment Unscheduled CardHolderInitiated	No

Credentials on File Verify Payment Return Parameters

Field Name	Type	Length	Description
RecurringDataRequest	String	20	Confirms the original RecurringData requested
RecurringDataResponse	String	40	Returns a pipe-delimited combined field response of RecurringData originally passed and network confirmation. Always return this full value of RecurringDataResponse in the subsequent transactions. Note: If RecurringData not passed in Submit, it will not be returned in VerifyPayment response.

Credentials on File TransactionWebService Input Parameters and Response values

Parameter	Type	Length	Description	Required
RecurringData	String	0-40	<blank> Recurring Recurring 00 Recurring 00 123456789012345 Installment Installment 00 Installment 00 123456789012345 Unscheduled Unscheduled 00 Unscheduled 00 123456789012345 CardHolderInitiated CardHolderInitiated 00 CardHolderInitiated 00 123456789012345 Case Sensitive. (Installment, Unscheduled and CardHolderInitiated are specific to Visa only)	No

Definitions, Cardholder Agreements and Technical Details

Recurring Payment

A Recurring payment is defined as a merchant initiated transaction in a series of transactions which uses a stored credential, processed with a variable amount over regular intervals. Intervals may not exceed one year between transactions. Cardholder agreement/consent is required for the merchant to initiate recurring transactions.

Examples of Recurring payments include subscription payments and bill payments such as an electric bill, gym membership, pest control, monthly car insurance payment, mobile phone bill, etc.

Recurring Cardholder Agreement

When entering into a Recurring cardholder agreement, the following information must be clearly displayed at the time the cardholder gives their consent. This information is to be displayed separately from the general purchase terms and conditions. Where required by applicable laws or regulations, the merchant or its agent must also provide to the cardholder a record of the cardholder's consent.

- A truncated version of the stored credential (e.g., last 4 digit of the account number)
- Method in which the cardholder will be notified of any changes in the agreement
- How the stored credential will be used
- Expiration date of the agreement when applicable

Additionally, before processing the transaction, the merchant must obtain the cardholder's consent to an agreement that contains the following:

- Transaction amount (includes all taxes and charges) or how the transaction amount will be determined.
- Transaction amount cannot contain finance charges, which must be listed separately.
- Transaction currency
- If surcharging is permitted and assesses, cardholder acknowledgement of any surcharges and associated disclosures
- Cancellation and refund policies
- Location of the merchant outlet
- The fixed dates or intervals on which the transaction will be processed
- The cardholder agreement must be retained for the duration of the agreement and provide it to the issuer upon request

Recurring Payment Cancellation Procedures

The merchant must provide a simple cancellation procedure to the cardholder. If the cardholder's order was initially accepted online, the merchant may provide an online cancellation procedure. The merchant cannot complete a transaction when:

- The transaction is beyond the agreed upon duration by the cardholder
- The cardholder request that the merchant change the payment method
- Cardholder cancels according to the agreed cancelation policy
- The merchant receives a decline response

First Time a Recurring Payment Credential is Stored

- Merchant to ask the cardholder if they'd like the merchant to store their payment credential (card) for future transactions prior to submitting the initial/first authorization request
- The transaction must be authorized either for the amount due or, if payment not required, by submitting a ZeroAuth request (\$0.00). Credentials may not be stored if the authorization declines.
- Submit correct `RecurringData = "Recurring"` in `InitPayment` call and capture response details including `RecurringDataResponse` in `VerifyPayment` call.

Code Example InitPayment

```
HCSERVICE.InitPaymentRequest hcRequest = new HCSERVICE.InitPaymentRequest();  
hcRequest.MerchantID = "778825001";  
    . . . (Other fields) . . .  
hcRequest.RecurringDataRequest = "Recurring";
```

```
response = hcWS.InitializePayment(hcRequest);
PaymentID = response.PaymentID
```

Code Example VerifyPayment Response

```
Token = "12345abcdeasdkfhaskdfha12314515",
AuthorizeAmount = 10.01,
. . . (Other fields) . . .
RecurringDataRequest = "Recurring"
RecurringDataResponse = "Recurring|00|01234567891234"
```

- When the authorization is approved, retain the full *RecurringDataResponse* to submit in subsequent authorization request.

Subsequent Recurring Transactions/TransactionWebService (TWS)

- The transaction must be authorized.
- Submit valid formatted *RecurringDataRequest* field. The following are valid Recurring formats:

```
<blank>
Recurring
Recurring|00|
Recurring|00| 314282069480098
```

- Example TWS Request and Response:

```
CreditSaleToken saleToken = new CreditSaleToken()

MerchantID = "778825001",
. . . (Other fields) . . .
RecurringDataRequest = "Recurring|00|01234567890123"

Response
AuthorizeAmount = 10.01,
. . . (Other fields) . . .
RecurringDataResponse = "Recurring|00|32104567893210"
```

Note on Response: The numbers following Recurring|00|32104567893210 combines a piped delimited response code |00| and the network generated numeric transaction identifier. This identifier is card brand specific and, at this time, Visa, MasterCard and Discover return a network transaction identifier. It is important to return the full *RecurringData* response value in any subsequent COF transaction request.

For Visa, MasterCard and Discover, a new *RecurringData* network transaction ID is passed back with each TWS response. This should be saved and used in the subsequent COF TWS transactions.

For other card brands allow "Recurring" or "Recurring|00|".

- Should a recurring transaction decline, the merchant may perform a resubmission. Merchant may attempt to resubmit the transaction for up to 4 times within 16 calendar days from the date of the original decline response. The resubmission of authorization can only be performed when the original decline response code was one of the following:
 - 05 (Authorization declined)
 - 51 (Insufficient funds)
 - 61 (Exceeds approval amount limit)
 - 65 (Exceeds withdrawal frequency limit)

Installment

An installment is a merchant initiated transaction in a series of transactions that uses a stored credential to make a payment for a fixed amount over a fixed period of time. Cardholder agreement/consent is required for the merchant to initiate installment transactions.

Examples of an Installment COF include furniture purchases or home shopping network purchase where the amount paid is a fixed amount over a fixed period of time.

Installment Cardholder Agreement

When entering into a cardholder agreement, the following information must be clearly displayed at the time the cardholder gives their consent. This information is to be displayed separately from the general purchase terms and conditions. Where required by applicable laws or regulations, the merchant or its agent must also provide to the cardholder a record of the cardholder's consent.

- A truncated version of the stored credential (e.g., last 4 digit of the account number)
- Method in which the cardholder will be notified of any changes in the agreement
- How the stored credential will be used
- Expiration date of the agreement when applicable

Additionally, before processing the transaction, the merchant must obtain the cardholder's express informed consent to an agreement that contains the following:

- Transaction amount (includes all taxes and charges) or how the transaction amount will be determined
- The amount may not include interest charges for the U.S. Region
- Transaction currency
- If surcharging is permitted and assesses, cardholder acknowledgement of any surcharges and associated disclosures
- Cancellation and refund policies
- Location of the merchant outlet
- The total purchase price
- Terms of future installment payments, including the dates, amounts, and currency
- The cardholder agreement must be retained for the duration of the agreement and provide it to the issuer upon request

Installment Cancellation Procedures

The merchant must provide a simple cancellation procedure to the cardholder. If the cardholder's order was initially accepted online, the merchant may provide an online cancellation procedure. The merchant cannot complete a transaction when:

- The transaction is beyond the agreed upon duration by the cardholder
- The cardholder request that the merchant change the payment method
- Cardholder cancels within the terms of the cancellation policy. Merchant must provide the cardholder a cancellation or refund confirmation in writing within 3 business days
- The merchant receives a decline response

First Time an Installment Credential is Stored

- Merchant to ask the cardholder if they'd like the merchant to store their payment credential (card) for future transactions prior to submitting the initial/first authorization request
- The transaction must be authorized either for the amount due or, if payment not required, by submitting a ZeroAuth request (\$0.00). Credentials may not be stored if the authorization declines.

- Submit correct `RecurringData = "Installment"` in `InitPayment` call and capture response details including `RecurringDataResponse` in `VerifyPayment` call.

Code Example InitPayment

```
HCSvc.InitPaymentRequest hcRequest = new HCSvc.InitPaymentRequest();
hcRequest.MerchantID = "778825001";
    . . . (Other fields) . . .
hcRequest.RecurringDataRequest = "Installment";

response = hcWS.InitializePayment(hcRequest);
PaymentID = response.PaymentID
```

Code Example VerifyPayment Response

```
Token = "12345abcdeasdkfhaskdfha12314515",
AuthorizeAmount = 10.01,
    . . . (Other fields) . . .
RecurringDataRequest = "Installment"
RecurringDataResponse = "Installment|00|01234567891234"
```

- When the authorization is approved, retain the full `RecurringDataResponse` to submit in subsequent installment authorization request.

Note: A merchant may not process an initial installment transaction until the merchandise or services have been provided to the cardholder and must not process individual installment transaction at intervals less than either 7 calendar days or the monthly anniversary of the shipment date.

Subsequent Installment Transactions/TransactionWebService (TWS)

- The transaction must be authorized.
- Submit valid formatted `RecurringDataRequest` field. The following are valid formats:

```
<blank>
Installment|00|314282069480098
Installment
Installment|00
```

- Example TWS Request and Response:

```
CreditSaleToken saleToken = new CreditSaleToken()

    MerchantID = "778825001",
    . . . (Other fields) . . .
    RecurringDataRequest = "Installment|00|01234567890123"

Response
    AuthorizeAmount = 10.01,
    . . . (Other fields) . . .
    RecurringDataResponse = "Installment|00|32104567893210"
```

Note on Response: The numbers following `Installment|00|32104567893210` combines a piped delimited authorization code `|00|` and the network generated numeric transaction identifier. This identifier is card brand specific and, at this time, only Visa returns a network transaction identifier portion in this response. It is important to return the full `RecurringDataResponse` value in any subsequent COF transaction request.

For Visa, a new `RecurringData` network transaction ID—is passed back with each TWS response. This should be saved and used in the subsequent COF TWS transactions.

For other card brands allow "Installment" or "Installment|00|".

- Should a subsequent installment transaction decline, the merchant must notify the cardholder in writing and allow the cardholder at least 7 days to pay by other means
- Should an authorization decline the merchant may perform a resubmission. Merchant may attempt to resubmit the transaction for up to 4 times within 16 calendar days from the date of the original decline response. The resubmission of authorization can only be performed when the original decline response code was one of the following:
 - 05 (Authorization declined)
 - 51 (Insufficient funds)
 - 61 (Exceeds approval amount limit)
 - 65 (Exceeds withdrawal frequency limit)

Unscheduled (Visa Only)

An Unscheduled COF is defined as a merchant initiated transaction in a series that uses a stored credential, processed without a regularly scheduled transaction date and with a fixed or variable amount. Cardholder agreement/consent is required for the merchant to initiate unscheduled COF transactions.

Examples of Unscheduled COF payments include services such as snowplow service, window clearer services or vacation home cleaning services where there are agreed upon dependencies of time and conditions for the service.

Unscheduled Cardholder Agreement

When entering into a cardholder agreement, the following information must be clearly displayed at the time the cardholder gives their consent. This information is to be displayed separately from the general purchase terms and conditions. Where required by applicable laws or regulations, the merchant or its agent must also provide to the cardholder a record of the cardholder's consent.

- A truncated version of the stored credential (e.g., last 4 digit of the account number)
- Method in which the cardholder will be notified of any changes in the agreement
- How the stored credential will be used
- Expiration date of the agreement when applicable

Additionally, before processing the transaction, the merchant must obtain the cardholder's express informed consent to an agreement that contains the following:

- Transaction amount (includes all taxes and charges) or how the transaction amount will be determined.
- The agreed upon event that will prompt the transaction (for example: to plow only when two inches of snow or more or the vacation home rental dates for cleaning.)
- Transaction currency
- If surcharging is permitted and assesses, cardholder acknowledgement of any surcharges and associated disclosures
- Cancellation and refund policies
- Location of the merchant outlet
- The fixed dates or intervals on which the transaction will be processed
- The cardholder agreement must be retained for the duration of the agreement and provide it to the issuer upon request

Unscheduled Cancellation Procedures

The merchant must provide a simple cancellation procedure to the cardholder. If the cardholder's order was initially accepted online, the merchant may provide an online cancellation procedure. The merchant cannot complete a transaction when:

- The transaction is beyond the agreed upon duration by the cardholder
- The cardholder request that the merchant change the payment method
- Cardholder cancels according to the agreed cancelation policy
- The merchant receives a decline response

First Time an Unscheduled Credential is Stored

- Merchant to ask the cardholder if they'd like the merchant to store their payment credential (card) for future transactions prior to submitting the initial/first authorization request
- The transaction must be authorized either for the amount due or, if payment not required, by submitting a ZeroAuth request (\$0.00). Credentials may not be stored if the authorization declines.
- Submit correct `RecurringData = "Unscheduled"` in `InitPayment` call and capture response details including `RecurringDataResponse` in `VerifyPayment` call.

Code Example InitPayment

```
HCSERVICE.InitPaymentRequest hcRequest = new HCSERVICE.InitPaymentRequest();

hcRequest.MerchantID = "778825001";
    . . . (Other fields) . . .
hcRequest.RecurringDataRequest = "Unscheduled";

response = hcWS.InitializePayment(hcRequest);
PaymentID = response.PaymentID
```

Code Example VerifyPayment Response

```
Token = "12345abcdeasdkfhaskdfha12314515",
AuthorizeAmount = 10.01,
    . . . (Other fields) . . .
RecurringDataRequest = "Unscheduled"
RecurringDataResponse = "Unscheduled|00|01234567891234"
```

- When the authorization is approved, retain the full `RecurringDataResponse` to submit in subsequent authorization request.

Subsequent Transactions -- Unscheduled

- The transaction must be authorized.
- Submit valid formatted `RecurringDataRequest` field. The following are valid formats:

```
<blank>
Installment|00|314282069480098
Installment
Installment|00|
```

Example TWS Request and Response:

```
CreditSaleToken saleToken = new CreditSaleToken()

MerchantID = "778825001",
    . . . (Other fields) . . .
RecurringDataRequest = "Unscheduled|00|01234567890123"
```

```
Response
  AuthorizeAmount = 10.01,
  . . . (Other fields) . . .
  RecurringDataResponse = "Unscheduled|00|32104567893210"
```

Note on Response: The numbers following `Unscheduled|00|32104567893210` combines a piped delimited authorization code `|00|` and the network generated numeric transaction identifier. This identifier is card brand specific and, at this time, only Visa returns a network transaction identifier. It is important to return the full *RecurringData* response value in any subsequent COF transaction request.

For Visa, a new *RecurringData* network transaction ID is passed back with each TWS response. This should be saved and used in the subsequent COF TWS transactions.

For other card brands allow "Unscheduled" or "Unscheduled|00|".

- Should an Unscheduled transaction decline, the merchant must notify the cardholder in writing and allow the cardholder at least 7 days to pay by other means
- Should an Unscheduled transaction decline the merchant may perform a resubmission. Merchant may attempt to resubmit the transaction for up to 4 times within 16 calendar days from the date of the original decline response. The resubmission of authorization can only be performed when the original decline response code was one of the following:
 - 05 (Authorization declined)
 - 51 (Insufficient funds)
 - 61 (Exceeds approval amount limit)
 - 65 (Exceeds withdrawal frequency limit)

Cardholder Initiated

A Cardholder Initiated payment is defined as a transaction in a series of transactions that uses a stored credential is neither determined by a fixed date or amount and is initiated by the cardholder. Cardholder agreement/consent is required for the merchant to store the credentials for future transactions.

Examples of Cardholder Initiated include online retail or restaurant merchant environments that store the cardholder's payment credentials, shipping and billing address is on file, such as Amazon, The U.S. Post Office or a Pizza delivery business.

Cardholder Initiated Agreement

When entering into a cardholder agreement, the merchant is establishing the terms and conditions of the consent to store the cardholder's payment credentials, shipping and billing address. The following information must be clearly displayed at the time the cardholder gives their consent. This information is to be displayed separately from the general purchase terms and conditions. Where required by applicable laws or regulations, the merchant or its agent must also provide to the cardholder a record of the cardholder's consent.

- A truncated version of the stored credential (e.g., last 4 digit of the account number)
- Method in which the cardholder will be notified of any changes in the agreement
- How the stored credential will be used
- Expiration date of the agreement when applicable
- Transaction currency
- If surcharging is permitted and assesses, cardholder acknowledgement of any surcharges and associated disclosures
- Cancellation and refund policies

- Location of the merchant outlet
- The cardholder agreement must be retained for the duration of the agreement and provide it to the issuer upon request

Cardholder Initiated Cancellation procedures

The merchant must provide a simple cancellation procedure to the cardholder. If the cardholder's order was initially accepted online, the merchant may provide an online cancellation procedure. The merchant cannot complete a transaction when:

- The transaction is beyond the agreed upon duration by the cardholder
- The cardholder request that the merchant change the payment method
- Cardholder cancels within the terms of the cancelation policy
- The merchant receives a decline response

First Time a Cardholder Initiated Payment Credential is Stored

- Merchant to ask the cardholder if they'd like the merchant to store their payment credential (card) for future transactions prior to submitting the initial/first authorization request
- The transaction must be authorized either for the amount due or, if payment not required, by submitting a ZeroAuth request (\$0.00). Credentials may not be stored if the authorization declines.
- Submit correct `RecurringData = "CardHolderInitiated"` in `InitPayment` call and capture response details including `RecurringDataResponse` in `VerifyPayment` call.

Code Example InitPayment

```
HCService.InitPaymentRequest hcRequest = new HCService.InitPaymentRequest();

hcRequest.MerchantID = "778825001";
    . . . (Other fields) . . .
hcRequest.RecurringDataRequest = "CardHolderInitiated";

response = hcWS.InitializePayment(hcRequest);
PaymentID = response.PaymentID
```

Code Example VerifyPayment Response

```
Token = "12345abcdeasdkfhaskdfha12314515",
AuthorizeAmount = 10.01,
    . . . (Other fields) . . .
RecurringDataRequest = "CardHolderInitiated"
RecurringDataResponse = "CardHolderInitiated|00|01234567891234"
```

- When the authorization is approved, retain the full `RecurringDataResponse` to submit in subsequent authorization request.

Subsequent Transactions/TransactionWebService (TWS)

- The merchant must validate the cardholder's identity (example: login ID and password) prior to processing each transaction
- The transaction must be authorized.
- Submit valid formatted `RecurringDataRequest` field. The following are valid formats:
 - <blank>
 - CardHolderInitiated
 - CardHolderInitiated|00|

```
CardHolderInitiated|00|314282069480098
```

Example TWS Request and Response:

```
CreditSaleToken saleToken = new CreditSaleToken()
```

```
    MerchantID = "778825001",
    . . . (Other fields) . . .
    RecurringDataRequest = "CardHolderInitiated|00|01234567890123"
```

```
Response
```

```
    AuthorizeAmount = 10.01,
    . . . (Other fields) . . .
    RecurringDataResponse = "CardHolderInitiated|00|32104567893210"
```

Note on Response: The numbers following CardHolderInitiated|00|32104567893210 combines a piped delimited authorization code |00| and the network generated numeric transaction identifier. This identifier is card brand specific and, at this time, only Visa returns a network transaction identifier. It is important to return the full *RecurringData* response value in any subsequent COF transaction request.

For Visa, a new *RecurringData* network transaction ID is passed back with each TWS response. This should be saved and used in the subsequent COF TWS transactions.

For other card brands allow "CardHolderInitiated " or "CardHolderInitiated |00|".

6. Gift and StoreCard

Overview

StoreCard™ is a Worldpay Integrated Payments-hosted gift and stored value processing platform designed to provide your POS system an integrated gift solution that can increase revenue, build merchant retention, and enhance the cardholder's experience.

Note: at the time of this writing, StoreCard is merging with the more robust features and functionality of the Valutec Gift platform. Please stay-tuned for more information on this exciting and backwards compatible upgrade to Valutec. Contact your integration consultant for further details coming in 2021.

StoreCard

StoreCard transforms the legacy *MercuryGift* program into a complete stored value service—it maintains all the benefits of a Mercury gift card, with the enhanced capabilities of a rewards card and mobile payment card, all wrapped into one. StoreCard is built on the MercuryPay GIFTePay™ platform and is integrated in the exact same way as MercuryGift. Merchants who already use MercuryGift can use the StoreCard with minimal changes to their POS integration, and their customers will then have easy access to all of the value-added features.

Merchants have the following options

StoreCard Manager	A website providing access to all StoreCard features, letting consumers manage their cards. This site is customizable to match the merchant's brand.
Cash Back Rewards	Designed to create customer loyalty, Cash Back Rewards encourage consumers to load and reload funds onto their cards. Every time a consumer uses their StoreCard to make a purchase, the consumer receives a merchant-configured cash back reward added directly onto the card. Merchants determine the percentage consumers will receive.
Go Mobile	Consumers have the option to convert their physical StoreCard into a digital QR code representation of the card. They can then pay using their digital card from their smartphone. This requires a merchant to use a QR code scanner.
Reload	Using StoreCard Manager, consumers can automatically or manually reload their StoreCard at any time, by linking their card to their credit card.
Protect Your Card	Cardholders can register their cards online to guard against loss or theft. If their card is lost or stolen, cardholders can cancel their card and Worldpay will send a new one loaded with the remaining balance.
Transactions	Cardholders can view their transaction history from the StoreCard Manager website.
Social Sharing	This will enable cardholders to share, tweet or pin a virtual gift onto social media.
DigiCard	Merchants have the option to promote their business with digital StoreCards from within their stores. Merchants determine the amount of the promotional offer and a text is sent to the customer with a link to open the digital StoreCard.

Processing Basics—the Gift Card/StoreCard Life Cycle

MercuryPay gift cards/StoreCards are shipped without a specified value, which allows the merchant to issue the cards per the request of the customer, or to use as a credit for returned merchandise.

- MercuryPay gift/StoreCard transactions are built around simple and consistent transaction **payload** using common XML request and response data elements.
- All gift transactions use the **transaction type of Prepaid** (<TranType>PrePaid</TranType>) accompanied by the corresponding transaction code (<TranCode>Issue</TranCode>). Other TranCodes include *Sale*, *NoNSFSale*, *Reload*, *Return*, *VoidIssue*, *VoidSale*, *VoidReload*, *VoidReturn* and *Balance*. Details are provided later in this guide.
- **The first gift transaction processed on a new gift card is Issue.** The *Issue* activates the card, sets the initial card balance, and associates the card with the usage parameters on the MercuryPay Gift/StoreCard server. Once a card is activated and put into use, it remains “issued” for the life of the card.
- **Gift cards/StoreCards can be used at the issuing merchant location or any associated locations as a form of tender.** Use *Sale* or *NoNSFSale* to reduce the available balance; use *Reload* or *Return* to add value to the available balance.
- **Worldpay manages the available balance of all Issued gift card accounts.** A balance value is always returned in the transaction response. A balance inquiry can also be processed at any time locally by the merchant, online by the cardholder or programmatically by the POS software.
- A comprehensive reporting tool is available on the *MercuryView* portal.

Supported Transactions

The following gift transactions are supported:

- Issue
- VirtualIssue
- VoidIssue
- Sale
- NoNSFSale
- VoidSale
- Reload
- VoidReload
- Return
- VoidReturn
- Balance

Although most developers choose to support the full range of gift transactions, we recommend supporting the following transactions at minimum for the best merchant and consumer experience possible:

- **Issue.** Required to perform the initial activation and load of the card. A card may be issued with a zero balance.
- **NoNSFSale.** This transaction allows the merchant to deduct the remaining balance from a gift card even if the purchase amount is greater than the card balance. Using this transaction avoids the “insufficient funds” error that would normally be received.
- **Reload.** Reload is used for monetary gift card loads. *Return* should be used for merchandise returns where the merchandise value is loaded onto a gift card.
- **Balance.** Used to determine a gift card’s balance even if the card is deactivated or expired.
- **Void Issue.** Reverses card activation.
- **Void Sale.** Reverses a redemption without triggering daily reload limits.

Standard Issue Gift Card Ranges

- Worldpay’s gift card ranges are 19 digits long, starting with **6050110XXXXXXXXXX**.
- Encoding standard is **Track2** format. Allow **Track1** formatting for future StoreCard enhancements. Though not used for gift card processing, there are value added mobile features planned for using Track1 data.
- If validating gift cards, it is recommended to use the complete “6050110” so as not to be mistakenly compared to Discover Card credit ranges.

Pre-Existing Cards and Third-Party Gift Cards

When a merchant has participated in another provider’s gift card program prior to processing with Worldpay, they may be able to use these cards with Worldpay. Consider the following:

- **If the encoded magnetic stripe data contains more than Track2 data**, it is important to parse out the Track2 string for processing to Worldpay. If the encoded magnetic stripe data contains less than Track2 data, it may be necessary to have Worldpay re-encode the data to a compatible format.
- **Worldpay’s gift servers may be able to support the previously issued cards** without having to physically re-encode the magnetic stripes and/or reprint the card numbers. In order to determine this, please contact the Worldpay Gift department and arrange to provide three to five sample gift cards for evaluation.
- **If the pre-existing cards ARE compatible with Worldpay’s platform**, Worldpay can import these gift card values. The merchant only needs to supply a CSV file or Excel spreadsheet listing the card numbers and current balances.
- In the event that the pre-existing cards are NOT compatible with Worldpay’s platform, Worldpay may be able to re-encode the Track2 portion of the merchant’s cards.

Chain or Franchise Merchant Gift Card Acceptance

If merchants are part of a larger chain or franchise, Worldpay can group accounts together so they will be able to perform *Sale* and *Reload* transactions on cards that are issued by other members of their group. Reports are also easily accessed that summarize these cross-merchant transactions, showing how much each store may owe to, and be owed by, the other stores in the group.

Gift/StoreCard XML Transactions Examples

There is a common set of data elements used to build gift transaction payloads, regardless of how your code is implemented. This section describes these common features as standard XML-based pairs. In other sections of this guide, details are provided on how to implement these same payloads using other integration methods via, for example RESTful API using JSON RPC and Web Services using SOAP.

Example of a Gift/StoreCard Transaction XML REQUEST

The example below illustrates the XML data elements required by Worldpay regardless of integration method.

Note There are three data elements that are used to differentiate a gift transaction from a credit transaction: *IpPort*, *TranType* and *TranCode*.

Each numbered tag is described below the figure:

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    ❶ <IpPort>9100</IpPort>
    ❷ <MerchantID>337234001</MerchantID>
      <OperatorID>test</OperatorID>
    ❸ <TranType>PrePaid</TranType>
      <TranCode>Issue</TranCode>
    ❹ <InvoiceNo>123456</InvoiceNo>
      <RefNo>123456</RefNo>
    ❺ <Memo>POS.v123</Memo>
      <Account>
    ❻ <Track2>60501100XXXXXX25120=250110117</Track2>
      </Account>
    ❼ <Amount>
      <Purchase>250.00</Purchase>
    </Amount>
  </Transaction>
</TStream>

```

1. **<IpPort> 9100** All gift transactions require that port 9100 be specified in the *IpPort* tag. If a gift transaction is sent without an *IpPort* specified, a client error will result with the message, "No Connection to any Server!"
2. **<MerchantID> (MID)** Worldpay merchant accounts are set up with a primary *MerchantID* (MID) used for routing daily transaction deposits to the merchant's bank, and identifying the merchant's monthly statement. This MID is also used for gift card processing.

- The format combines an 11-digit numeric value, separated by an "=" and a 6-digit alpha value, as in: **884300XXXXX=ABCDEF**. For coding purposes, allow a maximum value of 24 alphanumeric characters. Alpha characters are not case sensitive.
- Under the primary MID account, it is also possible to have additional, parallel processing sub-IDs, referred to as **Terminal IDs (TID)**. Used typically for credit and debit processing, these TIDs are linked to, and route funds to the primary Merchant account. **It is not required to use a separate TID for gift processing.** Gift accounts will be set up using the merchant's primary MID.

MerchantID and **TerminalIDs** can be independently configured to enable duplicate checking logic, tokenization and/or E2E support.

3. **<TranType>** (Transaction Type) and **<TranCode>** (Transaction Codes)

<TranType> is the transaction or tender category. For gift, use **Prepaid**.

<TranCode> is the specific type of transaction. Options include: *Issue*, *Sale*, *NoNSFSale*, *Return*, *Reload*, *Balance*, and all voids.)

4. **<InvoiceNo>** and **<RefNo>**

InvoiceNo, a required field, is the **local POS transaction number**. If it is missing, the request will error and return "Invoice Number Missing" in the response. The *InvoiceNo* should be kept distinct from other POS numbering methods (also referred to as order, ticket, check, or receipt number) because **each gift transaction that is processed requires a unique InvoiceNo, especially in the case of split tenders**. For example, if the POS presents ticket number #1234 for \$100 and this ticket is split among two gift cards, then

the POS would need to generate two separate receipts, each with its own unique *InvoiceNo* specific to each of the two cards presented.

- **InvoiceNo format:** Length: MIN: 1 digit, MAX: 16 digit, Right 10 unique; Data Type: Numeric only.

RefNo requires a “filler” value for all *Sale*, *NoNSFSale*, *Issue*, *Return*, *Reload* and *Balance* transaction requests, and is the actual response value for all Voids. Without a *RefNo* the request will error and return “Reference Number Missing” in the response.

- **RefNo format:** length: MIN 6 digits, MAX: 9 – always use full response value; Data Type: Numeric only.

Note: The *AuthCode* field in the response will match the *RefNo*. The value returned is 6 numeric digits. **Use the actual server-assigned response for RefNo and AuthCode** whenever submitting a *VoidIssue*, *VoidSale*, *Void Reload* or *Void Return*.

5. **<Memo>** Placed in all transaction requests, the **<Memo>** tag is used to identify your POS name and version. Example: **<Memo>POS NAME v1.10A-08</Memo>**.

Memo format: Length: 1-40 maximum; Data Type: Alpha-Numeric.

Note: In 2019, another tag was introduced by Datacap Systems for name and version: **<POSPackageID>** String, 1-99 chars).

6. **<Account> (parsed Track2 data)** This is the gift card account number. Track 2 data is obtained from a clear-text Card reader (MSR), a supported peripheral device, or an encryption E2E device. The card may either be swiped or manually entered.

Account level Examples:

In a **standard, non-encrypted swiped request, always parse Track2**, removing the start and end sentinels (; and ?). For example, here is a typical StoreCard formatted gift card Track encoding:
;6050110010021825120=250110117?

```
<Account>
  <Track2>605011001XXXXXX5120=250110117</Track2>
</Account>
```

In a **non-encrypted manual request**, only the *AcctNo* is required:

```
<Account>
  <AcctNo>605011001XXXXXX5120</AcctNo>
  <ExpDate>0513</ExpDate>
</Account>
```

The example below is for developers who also support **E2E encryption** and use a supported E2E peripheral for their credit integration. In this case, the gift card swipe data is encrypted and the *Account* data elements are sent using four additional, encryption-specific tags: **<EncryptedFormat>**, **<AccountSource>**, **<EncryptedBlock>** and **<EncryptedKey>**, as shown below:

```
<Account>
  <EncryptedFormat>MagneSafe</EncryptedFormat>
  <AccountSource>Swiped</AccountSource>
  <EncryptedBlock>F40DDBA1F645CC8DB85A6459D45AFF8002C244A0F4402B479ABC991
5EC9567C81BE99CE4483AF3D</EncryptedBlock>
  <EncryptedKey>9012090B01C4F200002B</EncryptedKey>
</Account>
```

For manual encryption transactions the *AccountSource* must be changed to **Keyed** and the accompanying *EncryptedBlock* is generated using a manual-entry format specific to each encryption device.

7. **<Amount>** For all gift transaction requests except *Balance*, the *Amount* element uses the **<Purchase>** tag. Amounts are dollar values to 2 decimal points—e.g., 25.30

```
<Amount>
```

```
<Purchase>25.30</Purchase>
</Amount>
```

Note: For developers supporting Tokenization: **Frequency** and **RecordNo** tags are used in Worldpay's tokenization procedures and if supplied, are ignored in gift processing; a token is never generated or returned.

Example of a Gift/StoreCard Transaction XML RESPONSE

All gift transactions return a common set of response data, whether originating in a locally installed *DSIClientX* or sent to a Worldpay/MercuryPay Web Services endpoint. This data contains authorization information and the available card balance and may be used to build receipts, build records for internal POS reporting, and can be stored for subsequent use.

The response data is divided into two sections:

- **<CmdResponse>** summarizes the authorization. It shows the response origin, status of the authorization response (**Approved, Declined, Success, Error**) and clarification verbiage in the **<TextResponse>**.
- **<TranResponse>** is the Transaction Response in detail.

Each numbered tag is described below this figure:

```
<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    ❶ <ResponseOrigin>Processor</ResponseOrigin>
    ❷ <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    ❸ <TextResponse>Approved</TextResponse>
  </CmdResponse>
  <TranResponse>
    ❹ <MerchantID>337234001</MerchantID>
    <TranType>PrePaid</TranType>
    ❺ <TranCode>Issue</TranCode>
    ❻ <InvoiceNo>123456</InvoiceNo>
    <OperatorID>test</OperatorID>
    ❼ <AcctNo>605011001XXXXXX5120</AcctNo>
    <RefNo>531747</RefNo>
    ❽ <AuthCode>531747</AuthCode>
    ❾ <Amount>
      <Authorize>250.00</Authorize>
      <Purchase>250.00</Purchase>
      <Balance>250.00</Balance></Amount>
    </TranResponse>
  </RStream>
```

<CmdResponse> summarizes the authorization:

1. **<Response Origin>** indicates the source of the response:

- **Client** indicates that the response is from a locally installed *DSIClientX.ocx* control or, for developers who point to Worldpay/MercuryPay Web Services endpoint, the *DSIClientX* maintained on Worldpay/MercuryPay Web Services platform.
- **Server** though rarely used, indicates is the response originating from MercuryPay's front-end gift server.
- **Processor** indicates that the response originates from Worldpay's processing platform.

2. **<DSIXReturnCode>** is **000000** on all *Approved* or *Declined* responses; on **ERROR** responses it is a six-digit numeric code that corresponds to dynamically-generated text responses. [Appendix 7](#) contains details on this text response verbiage.
3. **<CmdStatus>** and **<TextResponse>**

<CmdStatus> indicates the outcome of the transaction. For gift, the value returned will be either **Approved**, **Declined**, or **Error**.

<TextResponse> is clarifying verbiage used to explain the **<CmdStatus>**.

 - An *Approved* CmdStatus returns a TextResponse of "Approved".
 - A *Declined* or *Error* CmdStatus will return a TextResponse based on the origin of the message.

<TranResponse> contains all the transaction details as follows:
4. **<MerchantID>** is returned with the numeric value only.
5. **<TranType>** and **<TranCode>** mirror the transaction request.
6. **<InvoiceNo>** is passed back as generated by the POS.
7. **<AcctNo>** is returned in place of Track2 Data, returned in clear-text and may be stored as such.
8. **<RefNo>** and **<AuthCode>** are the same 6-digit numeric values generated at the time of the transaction authorization and are returned in the transaction response. (The *MercuryView* merchant portal report shows the Reference Number slightly differently—as a nine-digit value. It is the right-six digits of this identifier that are passed back in the transaction response for use in the *RefNo* and *AuthCode*.) These values may be stored for use with void transactions if supported.
9. **<Amount>** always returns the authorized amount, the original purchase amount and the remaining available balance amount.

The examples below illustrate **<Amount>** response tags:

- Response for **Sale, NoNSFSale** (where sufficient balance is remaining), *Return and Reload*:


```
<Amount>
  <Authorize>3.00</Authorize>
  <Purchase>3.00</Purchase>
  <Balance>109.76</Balance>
</Amount>
```
- Response for **Balance**:


```
<Amount>
  <Balance>200.00</Balance>
</Amount>
```
- Response for a **NoNSFSale** when the remaining balance is less than the purchase amount. In this case, a \$27.00 purchase amount was requested on a card that had a remaining balance of \$25.00:


```
<Amount>
  <Authorize>25.00</Authorize>
  <Purchase>25.00</Purchase>      <Balance>0.00</Balance>
</Amount>
```

Note: For *NoNSFSale* approved transactions, it is important to always compare the POS requested *Purchase* amount with the actual *Authorize* amount in order to submit a **balance due** for the remaining amount.

StoreCard Attributes

Each gift card issue range has specific features that control the behavior of the cards. These features are determined by the merchant during the gift card application process. The table below provides a list of these setup attributes and their default values.

Set up Feature	Remarks	Default Value
Card Expiration	There are five parameters pertaining to the expiration of gift cards. If "enforce expiration" is NOT activated, none of the other expiration attributes are active:	
Enforce Expiration	Controls whether transactions will be permitted after a card has expired.	NO expiration date
Monthly Debit on Expiration	Allows assessment of monthly inactivity fees after card has expired.	\$0.00
Months Expire	Each card will expire after a specified number of months, relative to the issue date.	<Null>
Absolute Expiration Date	All cards in the range will expire on specified date, regardless of issue/activation date.	<Null>
Expire Type	Two choices: Absolute or Relative (see items 3 and 4 above.)	Relative
Maximum Issue	Limits the maximum balance allowed on the card. An issue or return that results in exceeding the Maximum will decline.	\$500.00
Allow Return	Controls whether return transactions will be accepted.	Allow Returns
Max # of Returns per Day	If Returns ARE allowed, limits how many return transactions are allowed per day.	2 per card, per day
Deactivate Card at Zero	If true, once the card's balance reaches zero, it will be deactivated and no further transactions will be accepted for that card.	Do NOT deactivate
Use CRC	On swiped transactions, compare the full track read to the track on file. Provides a higher level of security against fraudulent cards. Cannot be used on third-party gift cards not originating from Worldpay.	Yes
Date Effective	Date after which previously issued cards can be redeemed (for holiday promotions, etc.)	Auto-populates with the date on which the card range was created
Dormancy	There are three fields related to dormancy, or non-use of an issued gift card:	
Enforce Dormancy	Allows assessment of monthly inactivity fees after a period of non-use.	NO dormancy
Dormancy Fee	Specifies the dollar amount to be deducted on a monthly basis after dormancy is met.	\$0.00
Months to Dormancy Activation	Specifies the number of months of inactivity before dormancy fees are applied.	12 months
Cashback Percentage	Controls the percentage at which value is added to the card based on redemption or reload amount Note only available on StoreCard	<Null>

Note Gift card legislation varies by State. For example, some states are now mandating the ability to "cash out" prepaid gift cards when amounts are \$10 or less. Therefore, it is important that developers, resellers and merchants consult with the appropriate State agencies in order to determine how to set merchant gift card attributes.

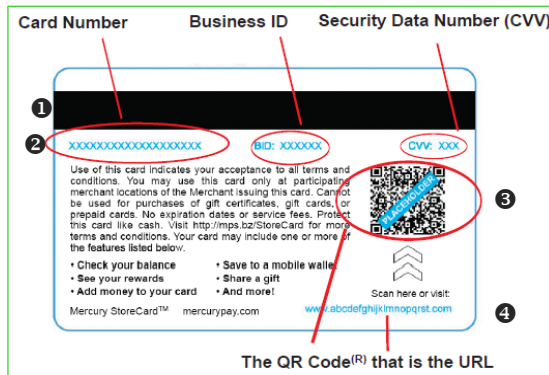
Card Encoding and Security Features – CRC and CVVData

For fraud protection on StoreCard cards, there is a printed set of numbers and an encoded set of numbers on the magnetic stripe. For manual entry, fraud protection is available using a three-digit card verification value (CVV). The three-digit security code is the standard encoding on StoreCard cards. A two-digit code on our legacy cards is also supported, which is referred to as Cyclic Redundancy Check (CRC).

Note Developers may code to support *both* 2 and 3 digit security data values. Two-digit CRC values are considered legacy but still in circulation. However, the CRC validation method should no longer be used.

The front of the card is designed by the merchant and the data or graphics vary.

Printed on the back of the card:



1. The encoding on the magnetic stripe contains **Track2** data, which is the encoding format used on StoreCard cards. Worldpay and many third parties use a non-ISO financial format when encoding gift cards. The following is a typical Track2 read found on StoreCard cards:

The sentinels (; and ?) are used to delineate the track. Sentinels must be parsed before processing. The data after the “=” is discretionary data only. **2501** is a filler value. Expiration dates are not imposed from the card track data. Expiration is a usage parameter associated with the card at the time the range is generated.

2. Directly under the magnetic stripe are the **card number**, **Business ID**, and three-digit **card verification value (CVV)**.
3. The QR code represents the Gift URL shown in Item 4 below.
4. The Gift URL is **unique to the merchant** and can be used to look up the card balance using the card number and Business ID.

Legacy Two-Digit Card Verification Value

Although no longer recommended, some older Mercury gift cards include a legacy two-digit card verification value referred to in this discussion as **CRC** (but typically also referred to as **CVV**). This two-digit value is printed on the back of the card.

In order to build a request that uses the CRC, the card number is combined with the two-digit security data and inserted into a modified, “pseudo” *Track2* format.

Example of a Manual Gift Transaction using the 2-digit Card Verification Value (CRC)

The example below uses the “pseudo” `<Track2>605011000XXXXXX6518=14</Track2>`, where **Account number** = 605011000XXXXXX6518, and **CRC**=14 (CRC is printed on the back of the card)

Request

```
<?xml version="1.0"?>
<TStream>
  <Transaction>
    <IpPort>9100</IpPort>
    <MerchantID>595901</MerchantID>
    <TranType>PrePaid</TranType>
    <TranCode>NoNSFSale</TranCode>
    <InvoiceNo>12</InvoiceNo>
    <RefNo>12</RefNo>
    <Memo>JBsPOS 7.5.0.2</Memo>
    <Account>
```

```

    <Track2>60501100XXXXXX6518=14</Track2>
  </Account>
  <Amount>
    <Purchase>2.10</Purchase>
  </Amount>
</Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
  <TextResponse>Approved</TextResponse>
  <UserTraceData></UserTraceData>
</CmdResponse>
  <TranResponse>
    <MerchantID>595901</MerchantID>
    <TranType>PrePaid</TranType>
    <TranCode>NoNSFSale</TranCode>
    <InvoiceNo>12</InvoiceNo>
    <AcctNo>605011000XXXXXX6518</AcctNo>
    <RefNo>379733</RefNo>
    <AuthCode>379733</AuthCode>
    <Amount>
      <Authorize>2.10</Authorize>
      <Purchase>2.10</Purchase>
      <Balance>195.65</Balance>
    </Amount>
  </TranResponse>
</RStream>

```

Note: The two-digit gift card verification value (Track2=CRC) method requires additional development work to build the pseudo-Track2 data and interpret the results.

Invalid or Missing CRC Error Handling when using the Track2=CRC method

1. **If the two-digit CRC is invalid or missing**, the transaction will decline with the text response: “Mag Stripe Error - Hand Key.” This message requires the card to be re-entered with the correct CRC, or re-entered without using the Track2=CRC method by using the **AcctNo** field only.
2. **If the AcctNo tag is inadvertently used in place of Track2**, the transaction will fail with “Invalid Account Number.” Using the Track2=CRC method requires the format illustrated above.

Preferred Card Verification Value (CVV)

CVVData is validated only when the gift card is used to tender a transaction (*Sale*, *NoNSFSale*, *Reload*, *Return* and corresponding voids.) “Non-tendered” transactions, *Issue*, *VoidIssue* and *Balance*, are not validated.

Developers must add a **<CVVData>** tag to the transaction request, which is used exclusively with the three-digit security code, e.g. **<CVVData>123 </CVVData>**. The value is numeric, MIN 3 digits, MAX 4 digits.

The result will be passed back as a single alpha character in the **<CVVResult>** tag in the response:

- CmdStatus is **Approved**, CVVResult is **M** indicates an approved security code match:
<CVVResult>M</CVVResult>
- CmdStatus is **Declined**, CVVResult is **P** indicates and invalid, missing or incorrect security code:
<CVVResult>P</CVVResult>

Gift cards are set to validate CVVData **if supplied**. If the CVVData tag is not provided, or if no value is present in the tag, Worldpay processes the transaction without any security check. If the CVVData tag is present and has data, Worldpay will validate the data and return the appropriate result.

Example of a Manual Gift Transaction using the 3-digit Card Verification Value

Request

```
<TStream>
  <Transaction>
    <IpPort>9100</IpPort>
    <MerchantID>337234001</MerchantID>
    <OperatorID>test</OperatorID>
    <TranType>PrePaid</TranType>
    <TranCode>Sale</TranCode>
    <InvoiceNo>123487</InvoiceNo>
    <RefNo>123487</RefNo>
    <Memo>POS.v345</Memo>
    <Account>
      <AcctNo>605011000XXXXXX0798</AcctNo>
    </Account>
    <Amount>
      <Purchase>3.52</Purchase>
    </Amount>
    <CVVData>880</CVVData>
  </Transaction>
</TStream>
```

Response

```
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Approved</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>337234001</MerchantID>
    <TranType>PrePaid</TranType>
    <TranCode>Sale</TranCode>
    <InvoiceNo>123487</InvoiceNo>
    <OperatorID>test</OperatorID>
    <AcctNo>605011000XXXXXX0798</AcctNo>
    <RefNo>532162</RefNo>
    <AuthCode>532162</AuthCode>
    <Amount>
      <Authorize>3.52</Authorize>
      <Purchase>3.52</Purchase>
      <Balance>193.48</Balance>
    </Amount>
    <CVVResult>M</CVVResult>
  </TranResponse>
</RStream>
```

StoreCard/Gift Duplicate Protection

Worldpay's gift server duplicate logic checks the card number, total amount, invoice number and date. **A duplicate is defined as the same card number, the same amount, and the same invoice number processed on the same day.** The duplicate gift transaction will decline and return:

```

<CmdResponse>
  <ResponseOrigin>Processor</ResponseOrigin>
  <DSIXReturnCode>000000</DSIXReturnCode>
  <CmdStatus>Declined</CmdStatus>
  <TextResponse>Duplicate Transaction</TextResponse>
  <UserTraceData></UserTraceData>
</CmdResponse>

```

The **Duplicate Transaction** response indicates that the same card account, amount and invoice number was already logged as an approved transaction and a duplicate has been detected.

- If the transaction is valid, (that is, if the cardholder is legitimately running a second transaction for the same amount on the same gift card), **then incrementing the *InvoiceNo* will resolve the decline.**
- If the decline was returned after a *Time out on Response* message, this indicates that the original gift transaction was authorized and does not require the transaction to be resubmitted.

StoreCard Readiness

StoreCard is built on the legacy MercuryPay Gift platform. The MercuryPay Gift Platform has been integrated into POS systems around the country since 2004 and is now integrated into hundreds of different types of POS systems.

Transaction Codes

Supported gift transactions include: *Issue*, *Virtual Issue*, *Sale*, *NoNSFSale* (supports split-tender), *Balance*, *Reload*, *Return*, *Time-out Reversal* and *Void* (*VoidIssue*, *VoidSale*, *VoidReload* and *VoidReturn*). Although most developers choose to support the full set of transactions, we recommend supporting the following transactions at minimum for the best merchant and customer experience possible:

- **Issue.** Required to perform the initial activation and load of the card.
- **NoNSFSale.** This transaction allows the merchant to deduct the remaining balance from a gift card even if the purchase amount is greater than the card balance. Using this transaction avoids the “insufficient funds” error that would normally be received.
- **Reload.** Reload is used for monetary gift card loads. *Return* should be used for merchandise returns where the merchandise value is loaded onto a gift card.
- **Balance.** Used to determine a gift card’s balance even if the card is deactivated or expired.
- **Void Issue.** Reverses card activation.
- **Void Sale.** Reverses a redemption without triggering daily reload limits.

2D Bar Code Scanning

If your merchant is interested in mobile payments, we suggest your POS integration supports 2D bar code scanning. The scanner will pull data from a QR code containing the 19-digit card number into your card number field on the POS. We recommend a keyboard (kb) emulated scanner which will operate just like a kb-card reader. Most POS systems have a USB port available and most QR scanners can easily be plugged into the POS. If your merchant is using an iPad® or tablet, they can use the built in QR code scanner; however, this may require some programming in order to integrate the scanning step into the checkout process. Another option for mobile POS users is to plug in a scanner into the tablet, or use a Bluetooth® or wireless scanner.

We have successfully tested these two types of scanners:

- Gryphon™ 4400 hand scanner
- Voyager 1400g

Apple/iOS Pass Wallet to support StoreCard with Verifone P400

Prerequisites

- Verifone Drivers T:\Departments\Software\Public\NFC\Verifone\VerifoneUnifiedDriverInstaller-5.0.4.0-B10\64\VerifoneUnifiedDriverInstaller64.msi)
- Datacap dsiPDCX Client Control (<http://files.datacapeway.com/software/dsiclient/dsiPDCX-161-Install20190220-W8.exe>)
- IOS device supporting Apple Wallet (Apple iPhone 5s or greater, (Apple watch model TBD.)

XML Example – VirtualIssue to generate the cardholder's StoreCard account

Although not required to handle Apple wallet StoreCard processing, in the example here, a PrePaid virtual issue is used along with Datacap System's dsiPDCX to issue a card virtually.

Note: If the merchant location has an existing gift card program with plastic cards, then the use of *VirtualIssue* is not necessary or can be used as a supplement to plastics. Here is a *VirtualIssue* XML request example:

```
<TStream>
  <Transaction>
    <IpPort>9100</IpPort>
    <MerchantID>755847006</MerchantID>
    <TranType>PrePaid</TranType>
    <TranCode>VirtualIssue</TranCode>
    <CardType>Digital</CardType>
    <InvoiceNo>1343</InvoiceNo>
    <RefNo>1343</RefNo>
    <Purchase>100.00</Purchase>
    <SecureDevice>NONE</SecureDevice>
    <ComPort>0</ComPort>
    <Account>
      <AcctNo>0</AcctNo>
    </Account>
    <Memo>Your POS and Version</Memo>
  </Transaction>
</TStream>
```

The transaction is passed to the StoreCard/Gift server for processing. The card will be "virtually" Issued and the response data will include the following:

- StoreCard account number
- Security code (CVV)
- Business ID

This data should be printed on the cardholder's receipt. These three values are required for the cardholder to add the card to StoreCard Manager and then to their Apple Wallet.

Here is a *VirtualIssue* XML response example:

```
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturn Code>000000</DSIXReturn Code>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Approved</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>755847006</MerchantID>
    <TranType>PrePaid</TranType>
    <TranCode>Issue</TranCode>
    <InvoiceNo>1343</InvoiceNo>
```

```

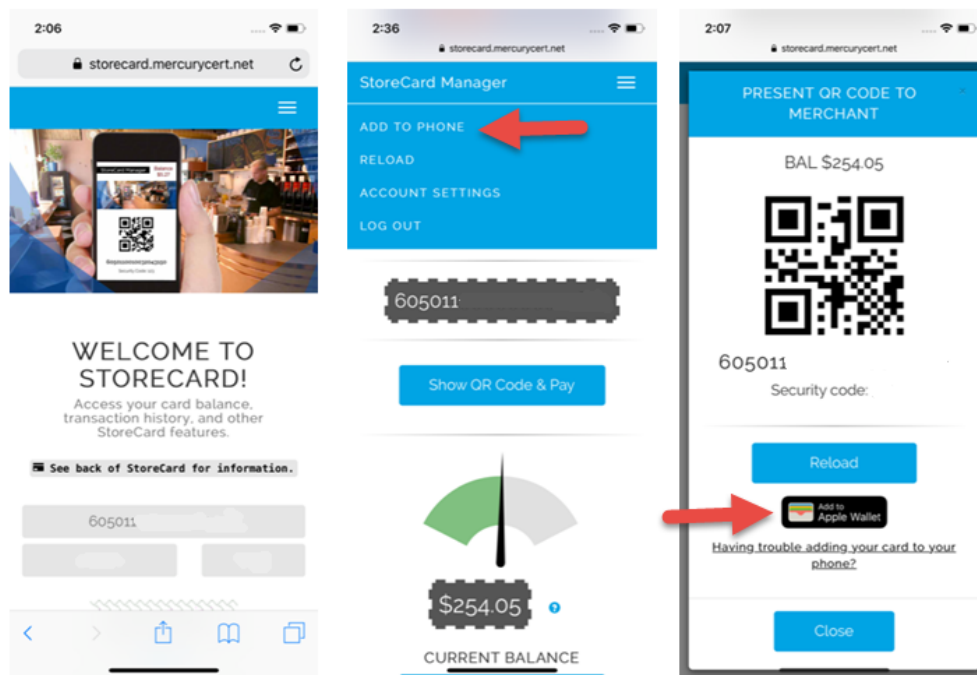
<AcctNo>60501100XXXXXX0001</AcctNo>
<CVVData>123</CVVData>
<BusinessId>12345</BusinessId>
<RefNo>330789</RefNo>
<AuthCode>330789</AuthCode>
<Amount>
  <Authorize>200.00</Authorize>
  <Purchase>200.00</Purchase>
  <Balance>200.00</Balance>
</Amount>
<RequestTranCode>VirtualIssue</RequestTranCode>
</TranResponse>
</RStream>

```

Cardholder's steps using Worldpay's StoreCard Manager

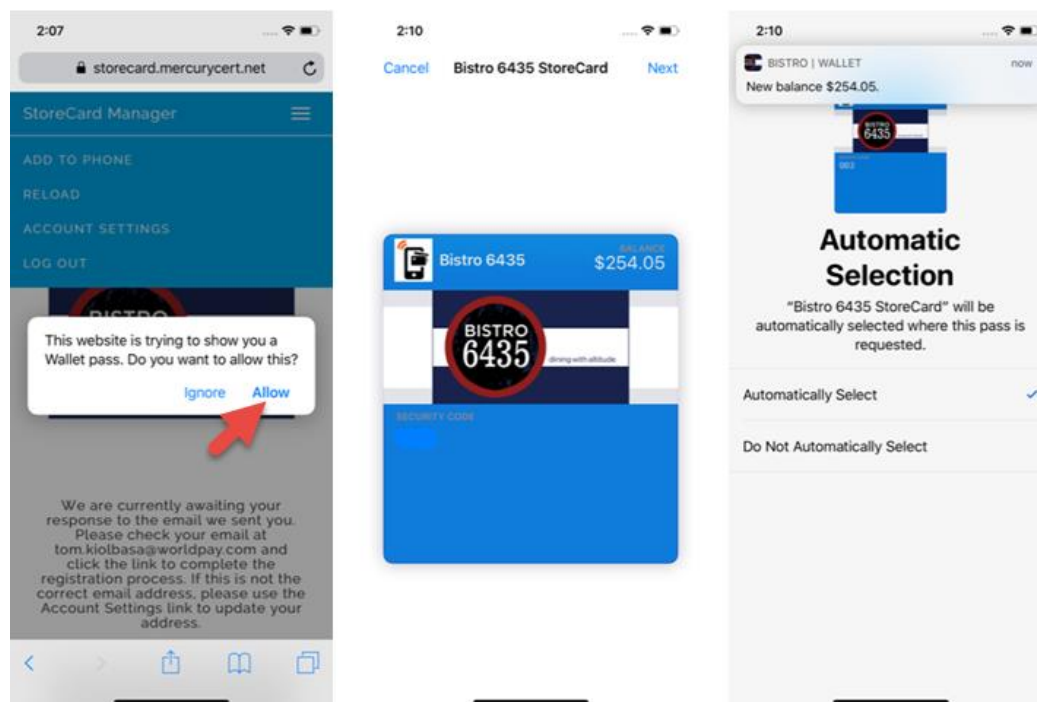
With the StoreCard account number, Security code (CVV) and Business ID in hand, (either virtually or with existing plastics) the cardholder can now add the virtual card to their iOS Apple Wallet. In the steps below, a certification/test environment of the StoreCard Manager is referenced.

1. On the iOS Device, Navigate to the Worldpay StoreCard Manager: <https://storecard.mercurycert.net/>
2. In the space provided enter the **Account Number**, **Business ID**, **CVV**, and captcha code, then click **Enter**.
3. Click in the upper right corner to view the menu options and click the **ADD TO PHONE** link.
4. Click on the button **Add to Apple Wallet**.



Note: If prompted by the browser: "This website is trying to show you a Wallet pass. Do you want to allow this?" click **Allow**.

5. The Apple Wallet application will launch with a preview of the mobile pass. Click **Next**.
6. Optionally check either box Automatically Select or Do Not Automatically Select. Click **Done**.



Using the Mobile Pass to run a PrePaid transaction

1. Open the Apple Wallet application on the IOS device and select the **StoreCard Pass**. Select the Wallet icon then select your **StoreCard Pass**. Click on the Pass to prepare for payment
2. On the POS application with the Verifone P400 in place, initiate a standard PrePaid Sale request to the dsIPDCX and follow the prompts on the device.
3. On the "**Swipe, Tap or Insert Card**" prompt, perform a gentle tap of the IOS device against the top of the P400 Terminal. This initiates the NFC payment from the device to the terminal.
4. Confirm the Apple Wallet pass displays **Done**.

StoreCard/Gift Transaction Codes

These transaction codes are applicable to both manual and swiped transactions.

TranCode	Description	Comments
Issue	Activates the account and loads a specified amount value onto the card.	<ul style="list-style-type: none"> • The card may not be previously issued . • The issue amount may be zero. • The issue amount may not exceed the merchant's agreed upon maximum issue amount value. • The <i>Issue</i> may be voided/cancelled by processing a <i>VoidIssue</i> as long as there is no transaction history on the card.
Virtual Issue	Issues a virtual card (as opposed to a physical plastic card). Once issued, it behaves like a normal gift card.	<ul style="list-style-type: none"> • Supports business need for non-physical gift cards that can be used in a mobile wallet.
Sale	Takes value off of the card, reducing the available balance by that purchase amount. Declines the transaction if there is insufficient balance.	<ul style="list-style-type: none"> • The card must already be issued. • Card must have sufficient balance to cover the purchase request. • A gift <i>Sale</i> can be cancelled by using a gift <i>VoidSale</i>.

TranCode	Description	Comments
NoNSFSale	<ul style="list-style-type: none"> Takes value off of the card, reducing the available balance by that purchase amount. If there is an insufficient account balance, the transaction will approve for the remaining balance on the card. Requires the POS to manage a "Balance Due" split-tender if there is an insufficient balance on the card to cover the total purchase amount. <p>Note Cash Out (Paid Out) is required in California. In California, customers may opt to "cash out" remaining balances under \$10.00. To accommodate this state mandate, developers can adopt the CashOut transaction described below or should build in provisions to allow using <i>Balance</i> followed by a <i>Sale</i> or <i>NoNSFSale</i> to bring the card to 0.00 balance and then provide a local POS-side "Paid out."</p>	<ul style="list-style-type: none"> For example, a gift card has a balance of \$60.00. The cardholder uses the card to perform a \$100 purchase; the <i>NoNSFSale</i> will authorize for the available balance of \$60.00, reducing the gift card balance to \$0.00. The POS must include logic which prompts for the remaining amount due, which is \$40.00 in this example. In either case, a <i>NoNSFSale</i> can be cancelled using a gift <i>VoidSale</i>.
Reload	Adds value to a card, increasing the balance. Typically used for a gift inventory line item where money is provided to load a balance.	<ul style="list-style-type: none"> The card must already be issued. Default is maximum two reloads/day..
Return	<ul style="list-style-type: none"> Adds value to a card, increasing the balance. Typically used when merchandise is returned and the value is added to a gift card in place of cash. 	<ul style="list-style-type: none"> The card must already be issued. <i>Return</i> may be used interchangeably with <i>Reload</i> however, it is important to consider tax and inventory impacts, if your POS handles this level of accounting detail.
VoidSale VoidReload VoidReturn	Cancels a previously approved <i>Sale/Reload/Return</i> , adding the voided amount back to the card.	All gift Voids must specify the <i>RefNo</i> (reference number) and <i>AuthCode</i> (authorization code) returned in the originating transaction response, as well as the card number and original transaction amount.
VoidIssue	Cancels the issuance of a card, returning the account balance to 0.00 and puts the card into a non-issued state.	<ul style="list-style-type: none"> <i>VoidIssue</i> is allowed only when there is no transaction history on the card since the original <i>Issue</i> transaction.
Balance	<ul style="list-style-type: none"> A balance inquiry returns the remaining balance of a card. In states requiring "cash-out" functionality, use <i>Balance</i> to confirm the remaining value on the card, then build in a POS side "Cashout" by sending a <i>NoNSFSale</i> for that available balance, bringing the card to 0.00 and a "paid out" at the POS for the amount of the card balance. 	<ul style="list-style-type: none"> For liability concerns, there is a default maximum value allowed on gift cards of \$500.00. Merchants may adjust this default during their gift account application. The balance inquiry is independent of a card's expiration date, so a balance will be returned even for cards that have an enforced expiration date. Cards that have been de-activated through a <i>VoidIssue</i> will not return a balance if queried using the <i>Balance</i> transaction. A TextResponse, "Account not Issued" will be returned instead.
Time-out Reversal	Used to reverse a transaction in the case of a time-out. May be used at any time within 30 minutes of the original transaction.	<ul style="list-style-type: none"> Use the original transaction request, and change the <i>TranCode</i> to Reverse. <i>AuthCode</i> is not required. If reversing a <i>VirtualIssue</i>, <i>AcctNo</i> is not required. If the original transaction is found, the response will indicate the success or failure of the reversal. If no original transaction is found, response contains NO TRANS FOUND.

TranCode	Description	Comments
Note Worldpay's gift server returns the available balance in the response for all gift card transactions. It is recommended to print the available balance on all receipts. Please review the State regulations where your POS is distributed for additional requirements.		

StoreCard/Gift Card Error Responses

Error message	Description
Gift Set Up	
No MerchantID Specified	This merchant ID was not found in our database
MerchantID not found	The merchant ID entered is not valid
Merchant Not Accepting Gift	This merchant ID is not setup for gift
Invalid TranType for Prepaid Server	The transaction type entered is not valid for gift
Invalid TranCode	The transaction code entered is not valid for gift
Database Insertion Error	Can't insert into our database
Issue	
Invalid Account Number	Account number not in Merchant's range
Account Not Issued	Card has not been issued, or has been de-activated with <i>VoidIssue</i> .
Missing Invoice Field	There was no invoice number given
Amount Field Error	Incorrect amount entered
Non Numeric Account	Account number provided is not numeric
Account Already Issued	Gift Card has already been issued
Issuance Not Authorized	Didn't find the card in the merchant's range or exceed max amount allowed to issue
Not a GIFT AccountType	Wrong issuing range type
Mag Stripe Error - Hand Key	Card did not swipe—please key in the card number
Amount Exceeds Maximum	Exceeds the maximum amount that the gift card can be issued for
Amount Cannot Be Zero	There must be an amount higher than Zero
Could not Issue	Card cannot be issued
Return	
Account has not met the Effective Date	The card's start date has not been reached
Account Deactivated	The gift card account has been deactivated and is no longer allowing transactions
Return Not Allowed	Returns are not allowed on this card
Duplicate Transaction	There has already been a return issued for this invoice number, card number and amount
Number of Returns on Card over Limit	The maximum amount of returns on the card has been met
Sale	
Account on Hold	The gift card is on hold, no transactions are allowed on this card
Account Expired	The gift card account has expired
Insufficient Account Balance	There are not enough funds on this account
Bad Reference Number	The reference number supplied is not valid
No Account Match no account for this merchant	The card number is not in the valid range for this merchant
No Amount Match	

Error message	Description
Void	
Void RefNo not found	No transaction to void was found
Void Last Transaction First	Not the last transaction
Please use VoidIssue	A void cannot void an issue, please use the <i>VoidIssue</i> transactions type
Declined - Insert Error	Can't insert into database
VoidIssue	
Account In Use	Too many transactions already on card to void issue
Could not VoidIssue	Unable to complete VoidIssue transaction

Datacap Direct and dsiPDCX XML PrePaid/StoreCard Integration Method

DNS Server List for Gift

Datacap Gift Server	Purpose	Remarks
g1.mercurycert.net	Developer Testing g1 server	<ul style="list-style-type: none"> IP Port: 9100 must be specified* Testing Server Ipconfig string: g1.mercurycert.net;
g1.mercurypay.com	Production g1 server	<ul style="list-style-type: none"> IP Port: 9100 must be specified* Production Server Ipconfig string:
g2.backuppays.com	Production g2 server	g1.mercurypay.com;g2.backuppays.com

* The DSIclientX defaults to port 9000 for credit/debit/EBT/check processing. **For Gift, port 9100 must be specified.**

Datacap Direct PrePaid/StoreCard XML Payload Examples

- Gift Card Virtual Issue
- Swiped Gift Sale
- Manual Gift Sale using CVV Data to validate card
- Swiped Gift Reload
- Manual Gift Return
- Gift VoidSale
- Swiped Gift Balance
- Swiped Gift NoNSFSale
- Time-out Reversal

XML: Gift Card Virtual Issue

VirtualIssue is designed for eCommerce applications to issue StoreCards without the need of generating corresponding plastic cards.

Note: **CardType** is configured per merchant implementation; in the example below `<CardType>Digital</CardType>` is used specifically for StoreCard.

Request

```
<?xml version="1.0" encoding="utf-16"?>
<TStream>
  <Transaction>
    <IpPort>9100</IpPort>
    <MerchantID>33234001</MerchantID>
    <TranType>PrePaid</TranType>
    <CardType>Digital</CardType>
    <TranCode>VirtualIssue</TranCode>
    <InvoiceNo>543121</InvoiceNo>
    <Amount>
      <Purchase>100.00</Purchase>
    </Amount>
  </Transaction>
</TStream>
```

```

    </Transaction>
  </TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Approved</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>337234001</MerchantID>
    <TranType>PrePaid</TranType>
    <TranCode>Issue</TranCode>
    <InvoiceNo>543121</InvoiceNo>
    <AcctNo>771XXXXXX1092</AcctNo>
    <RefNo>167458</RefNo>
    <AuthCode>167458</AuthCode>
    <Amount>
      <Authorize>1.01</Authorize>
      <Purchase>1.01</Purchase>
      <Balance>100.01</Balance>
    </Amount>
    <RequestTranCode>VirtualIssue</RequestTranCode>
  </TranResponse>
</RStream>

```

XML: Swiped Gift Sale

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <IpPort>9100</IpPort>
    <MerchantID>337234001</MerchantID>
    <OperatorID>test</OperatorID>
    <TranType>PrePaid</TranType>
    <TranCode>Sale</TranCode>
    <InvoiceNo>123469</InvoiceNo>
    <RefNo>123469</RefNo>
    <Memo>POS.v345</Memo>
    <Account>
      <Track2>60501100XXXXXX5120=250110117</Track2>
    </Account>
    <Amount>
      <Purchase>2.21</Purchase>
    </Amount>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>

```

```

        <DSIXReturnCode>000000</DSIXReturnCode>
        <CmdStatus>Approved</CmdStatus>
        <TextResponse>Approved</TextResponse>
        <UserTraceData></UserTraceData>
    </CmdResponse>
    <TranResponse>
        <MerchantID>337234001</MerchantID>
        <TranType>PrePaid</TranType>
        <TranCode>Sale</TranCode>
        <InvoiceNo>123469</InvoiceNo>
        <OperatorID>test</OperatorID>
        <AcctNo>60501100XXXXXX5120</AcctNo>
        <RefNo>532128</RefNo>
        <AuthCode>532128</AuthCode>
        <Amount>
            <Authorize>2.21</Authorize>
            <Purchase>2.21</Purchase>
            <Balance>197.79</Balance>
        </Amount>
    </TranResponse>
</RStream>

```

XML: Manual Gift Sale using CVV Data to validate card

Request

```

<?xml version="1.0"?>
<TStream>
    <Transaction>
        <IpPort>9100</IpPort>
        <MerchantID>337234001</MerchantID>
        <OperatorID>test</OperatorID>
        <TranType>PrePaid</TranType>
        <TranCode>Sale</TranCode>
        <InvoiceNo>123487</InvoiceNo>
        <RefNo>123487</RefNo>
        <Memo>POS.v345</Memo>
        <Account>
            <AcctNo>60501100XXXXXX0798</AcctNo>
        </Account>
        <Amount>
            <Purchase>3.52</Purchase>
        </Amount>
        <CVVData>880</CVVData>
    </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
    <CmdResponse>
        <ResponseOrigin>Processor</ResponseOrigin>
        <DSIXReturnCode>000000</DSIXReturnCode>
        <CmdStatus>Approved</CmdStatus>
        <TextResponse>Approved</TextResponse>
        <UserTraceData></UserTraceData>
    </CmdResponse>
    <TranResponse>
        <MerchantID>337234001</MerchantID>
        <TranType>PrePaid</TranType>
        <TranCode>Sale</TranCode>

```



```

    <InvoiceNo>123487</InvoiceNo>
    <OperatorID>test</OperatorID>
    <AcctNo>60501100XXXXXX0798</AcctNo>
    <RefNo>532162</RefNo>
    <AuthCode>532162</AuthCode>
    <Amount>
      <Authorize>3.52</Authorize>
      <Purchase>3.52</Purchase>
      <Balance>193.48</Balance>
    </Amount>
    <CVVResult>M</CVVResult>
  </TranResponse>
</RStream>

```

XML: Swiped Gift Reload

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <IpPort>9100</IpPort>
    <MerchantID>337234001</MerchantID>
    <OperatorID>test</OperatorID>
    <TranType>PrePaid</TranType>
    <TranCode>Reload</TranCode>
    <InvoiceNo>123471</InvoiceNo>
    <RefNo>123471</RefNo>
    <Memo>POS.v345</Memo>
    <Account>
      <Track2>605011001XXXXX5120=250110117</Track2>
    </Account>
    <Amount>
      <Purchase>2.23</Purchase>
    </Amount>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Approved</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>337234001</MerchantID>
    <TranType>PrePaid</TranType>
    <TranCode>Reload</TranCode>
    <InvoiceNo>123471</InvoiceNo>
    <OperatorID>test</OperatorID>
    <AcctNo>60501100XXXXXX5120</AcctNo>
    <RefNo>532130</RefNo>
    <AuthCode>532130</AuthCode>
    <Amount>
      <Authorize>2.23</Authorize>
      <Purchase>2.23</Purchase>
      <Balance>197.80</Balance>
    </Amount>
  </TranResponse>
</RStream>

```

```

    </TranResponse>
  </RStream>

```

XML: Manual Gift Return

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <IpPort>9100</IpPort>
    <MerchantID>337234001</MerchantID>
    <OperatorID>test</OperatorID>
    <TranType>PrePaid</TranType>
    <TranCode>Return</TranCode>
    <InvoiceNo>123472</InvoiceNo>
    <RefNo>123472</RefNo>
    <Memo>POS.v345</Memo>
    <Account>
      <AcctNo>6050110XXXXXX25120</AcctNo>
    </Account>
    <Amount>
      <Purchase>2.24</Purchase>
    </Amount>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Approved</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>337234001</MerchantID>
    <TranType>PrePaid</TranType>
    <TranCode>Return</TranCode>
    <InvoiceNo>123472</InvoiceNo>
    <OperatorID>test</OperatorID>
    <AcctNo>605011001XXXXXX5120</AcctNo>
    <RefNo>532131</RefNo>
    <AuthCode>532131</AuthCode>
    <Amount>
      <Authorize>2.24</Authorize>
      <Purchase>2.24</Purchase>
      <Balance>200.04</Balance>
    </Amount>
  </TranResponse>
</RStream>

```

XML: Gift VoidSale

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>

```

```

    <IpPort>9100</IpPort>
    <MerchantID>337234001</MerchantID>
    <OperatorID>test</OperatorID>
    <TranType>PrePaid</TranType>
    <TranCode>Void Sale</TranCode>
    <InvoiceNo>123470</InvoiceNo> <!--use the Invoice of the tran you wish to void-->
    <RefNo>532129</RefNo> <!--use the RefNo and AuthCode values corresponding
to the tran you wish to void-->
    <Memo>POS.v345</Memo>
    <Account>
        <AcctNo>60501100XXXXXX25120</AcctNo>
    </Account>
    <Amount>
        <Purchase>2.22</Purchase>
    </Amount>
    <TranInfo>
        <AuthCode>532129</AuthCode> <!--use the RefNo and AuthCode values
corresponding to the tran you wish to void-->
    </TranInfo>
</Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
    <CmdResponse>
        <ResponseOrigin>Processor</ResponseOrigin>
        <DSIXReturnCode>000000</DSIXReturnCode>
        <CmdStatus>Approved</CmdStatus>
        <TextResponse>Voided</TextResponse>
        <UserTraceData></UserTraceData>
    </CmdResponse>
    <TranResponse>
        <MerchantID>337234001</MerchantID>
        <TranType>PrePaid</TranType>
        <TranCode>Void Sale</TranCode>
        <InvoiceNo>123470</InvoiceNo>
        <OperatorID>test</OperatorID>
        <AcctNo>60501100XXXXXX25120</AcctNo>
        <RefNo>532129</RefNo>
        <Balance>202.26</Balance>
        <Amount>
            <Authorize>2.22</Authorize>
            <Purchase>2.22</Purchase>
            <Balance>202.26</Balance>
        </Amount>
    </TranResponse>
</RStream>

```

XML: Swiped Gift Balance

Request

```

<?xml version="1.0"?>
<TStream>
    <Transaction>
        <IpPort>9100</IpPort>
        <MerchantID>337234001</MerchantID>
        <OperatorID>test</OperatorID>
        <TranType>PrePaid</TranType>
        <TranCode>Balance</TranCode>
    </Transaction>
</TStream>

```

```

    <InvoiceNo>123474</InvoiceNo>
    <RefNo>123474</RefNo>
    <Memo>POS.v345</Memo>
    <Account>
      <Track2>605011001XXXXX5120=250110117</Track2>
    </Account>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Approved</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>337234001</MerchantID>
    <TranType>PrePaid</TranType>
    <TranCode>Balance</TranCode>
    <InvoiceNo>123474</InvoiceNo>
    <OperatorID>test</OperatorID>
    <AcctNo>60501100XXXXXX5120</AcctNo>
    <Amount>
      <Balance>202.26</Balance>
    </Amount>
  </TranResponse>
</RStream>

```

XML: Swiped Gift NoNSFSale

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <IpPort>9100</IpPort>
    <MerchantID>337234001</MerchantID>
    <OperatorID>test</OperatorID>
    <TranType>PrePaid</TranType>
    <TranCode>NoNSFSale</TranCode>
    <InvoiceNo>123474</InvoiceNo>
    <RefNo>123474</RefNo>
    <Memo>POS.v345</Memo>
    <Account>
      <Track2>605011001XXXXX5120=250110117</Track2>
    </Account>
    <Amount>
      <Purchase>225.00</Purchase>
    </Amount>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>

```

```

    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Approved</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>337234001</MerchantID>
    <TranType>PrePaid</TranType>
    <TranCode>NoNSFSale</TranCode>
    <InvoiceNo>123474</InvoiceNo>
    <OperatorID>test</OperatorID>
    <AcctNo>60501100XXXXXX25120</AcctNo>
    <RefNo>532134</RefNo>
    <AuthCode>532134</AuthCode>
    <Amount>
      <Authorize>202.26</Authorize>
      <Purchase>202.26</Purchase>
      <Balance>0.00</Balance>
    </Amount>
  </TranResponse>
</RStream>

```

XML: Time-Out Reversal

The below example is for a **Sale** that timed out.

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <IpPort>9100</IpPort>
    <MerchantID>337234001</MerchantID>
    <OperatorID>Operator 1</OperatorID>
    <TranType>PrePaid</TranType>
    <TranCode>Reverse</TranCode>
    <InvoiceNo>08011101</InvoiceNo>
    <Account>
      <AcctNo>60501100XXXXXX37961</AcctNo>
    </Account>
    <Amount>
      <Purchase>1.03</Purchase>
    </Amount>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Voided</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>337234001</MerchantID>
    <TranType>PrePaid</TranType>
    <TranCode>VoidSale</TranCode>
  </TranResponse>
</RStream>

```

```

    <InvoiceNo>08011101</InvoiceNo>
    <OperatorID>Operator 1</OperatorID>
    <AcctNo>605011001XXXXXX7961</AcctNo>
    <RefNo>167572</RefNo>
    <Balance>16.03</Balance>
    <Amount>
      <Authorize>1.03</Authorize>
      <Purchase>1.03</Purchase>
      <Balance>16.03</Balance>
    </Amount>
    <RequestTranCode>Reverse</RequestTranCode>
  </TranResponse>
</RStream>

```

PaymentsAPI: Gift and StoreCard with JSON Remote Procedure Call

Setup

JSON formatted requests and responses use HTTPS for transaction processing.

Development and Production Servers/Hosts

Development/Cert Server	Primary	https://w1.mercurycert.net/PaymentsAPI/
Note: a primary and secondary server address are supplied for production		
Production Server	Primary	https://w1.mercurypay.com/PaymentsAPI/
Production Server	Secondary	https://w2.mercurypay.com/PaymentsAPI/

Authentication

Authentication occurs via HTTPS Basic Auth using an HTTPS authorization header.

1. Create string of [username]:[password]
 - **Username:** existing MerchantID
 - **Password:** created and stored by Mercury
2. BASE64 encoded string
3. Set Authorization header value to: "Basic [encoded string]"
4. Set Content-Type header value to: "application/json"

POSTS

Transactions are posted to the appropriate development/cert or production server address followed by the transaction type appended to the end of the URL.

- For example, a credit sale to our cert server would be:
<https://w1.mercurycert.net/PaymentsAPI/PrePaid/Sale>

JSON/RPC PrePaid Transactions Example Requests and Responses

Credit card information (card number and expiration date) can be sent by swiping the card through a standard or encrypted magnetic card reader (**swipe**) or manually entered (**keyed**).

- The JSON/RPS formats below include a clear-text example followed by an encryption example.
- Bold fields are required. Gray fields are optional.
- Values used in the examples below is intended as guidance for development only.

PrePaid (StoreCard and Gift) Transactions

MercuryPay StoreCard and Gift implementation enables gift processing designed to provide your POS system with a fully integrated gift solution. Implementation follows the same JSON/RPC formats as Credit with some minor changes.

- Gift transactions use the transaction type of **Prepaid**.

- Encryption may be used to process MercuryPay StoreCards.

Note: PaymentsAPI and JSON/RPC currently support **PrePaid Issue, VoidIssue, Sale, VoidSale, NoNFSSale, Reload, VoidReload, Return, VoidReturn and Balance**. Virtual Issue is currently not supported.

/PrePaid/Issue – Clear-Text

A **PrePaid Issue** activates/initializes a new StoreCard/Gift card and sets the initial card balance; the *Issue* associates the card with the usage parameters on the MercuryPay Gift server. Once a card is activated and put into use, it remains “issued” for the life of the card.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/Issue>

Request: Swipe

```
{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "7712950000000000353=250110118",
  "Purchase": "1.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Request: Keyed (manual)

Substitute the **Track2** data with the hand-keyed account number as follows:

"AcctNo": "7712950000000000353",

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Approved",
  "UserTraceData": "",
  "MerchantID": "595901",
  "TranType": "PrePaid",
  "TranCode": "Issue",
  "InvoiceNo": "1",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "771295XXXXXXXXX0353",
  "RefNo": "564643",
  "AuthCode": "564643",
  "Authorize": "1.00",
  "Purchase": "1.00",
  "Balance": "1.00"
}
```

/PrePaid/Issue – Encryption

A **PrePaid Issue** activates/initializes a new StoreCard/Gift card and sets the initial card balance; the *Issue* associates the card with the usage parameters on the MercuryPay Gift server. Once a card is activated and put into use, it remains “issued” for the life of the card.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/Issue>

Request: Swipe

```
{
  "InvoiceNo": "1",
```

```

"RefNo": "1",
"Memo": "MPS Example JSON v1.0",
"EncryptedFormat": "MagneSafe",
"AccountSource": "Swiped",
"EncryptedBlock": "C8C8F9536826D5450E734953206E7F4DC6812C6858037F5ABF23D9F83F948AF7",
"EncryptedKey": "9012090B06349B000056",
"Purchase": "100.00",
"TerminalName": "MPS Terminal",
"ShiftID": "MPS Shift",
"OperatorID": "MPS Operator"
}

```

Request: Keyed (manual)

Substitute the **AccountSource** as follows:

```
"AccountSource": "Keyed",
```

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Approved",
  "UserTraceData": "",
  "MerchantID": "395347305=E2E",
  "TranType": "PrePaid",
  "TranCode": "Issue",
  "InvoiceNo": "1",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "605011XXXXXXXXX0146",
  "RefNo": "566983",
  "AuthCode": "566983",
  "Authorize": "100.00",
  "Purchase": "100.00",
  "Balance": "100.00"
}

```

/PrePaid/VoidIssue – Clear-Text

A **PrePaid VoidIssue** cancels the issuance of a card, removes the amount loaded during the Issue and puts the card into a non-issued state. *VoidIssue* is allowed only when there is **no transaction history on the card**.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/VoidIssue>

Request: Swipe

```

{
  "InvoiceNo": "1",
  "RefNo": "564643",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "771295000000000353=250110118",
  "Purchase": "1.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator",
  "AuthCode": "564643"
}

```

Request: Keyed (manual)

Substitute the **Track2** data with the hand-keyed account number as follows:

```
"AcctNo": "771295000000000353",
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Voided",
  "UserTraceData": "",
  "MerchantID": "595901",
  "TranType": "PrePaid",
  "TranCode": "VoidIssue",
  "InvoiceNo": "1",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "771295XXXXXXXXX0365",
  "RefNo": "564644",
  "Authorize": "1.00",
  "Purchase": "1.00",
  "Balance": "0.00"
}
```

/PrePaid/VoidIssue – Encryption

A **PrePaid VoidIssue** cancels the issuance of a card, removes the amount loaded during the Issue and puts the card into a non-issued state. *VoidIssue* is allowed only when there is **no transaction history on the card**.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/VoidIssue>

Request: Swipe

```
{
  "InvoiceNo": "1",
  "RefNo": "566983",
  "Memo": "MPS Example JSON v1.0",
  "EncryptedFormat": "MagneSafe",
  "AccountSource": "Swiped",
  "EncryptedBlock": "C8C8F9536826D5450E734953206E7F4DC6812C6858037F5ABF23D9F83F948AF7",
  "EncryptedKey": "9012090B06349B000056",
  "Purchase": "100.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator",
  "AuthCode": "566983"
}
```

Request: Keyed (manual)

Substitute the **AccountSource** as follows:

"AccountSource": "Keyed",

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Voided",
  "UserTraceData": "",
  "MerchantID": "395347305=E2E",
  "TranType": "PrePaid",
  "TranCode": "VoidIssue",
  "InvoiceNo": "1",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "605011XXXXXXXXX0146",
  "RefNo": "566984",
  "Authorize": "100.00",
  "Purchase": "100.00",
}
```

```
"Balance": "0.00"
}
```

/PrePaid/Sale – Clear-Text

A **PrePaid Sale** takes value off of the card, reducing the available balance by that purchase amount; declines the transaction if there is insufficient balance. (See **NoNSFSale** for an alternative option)

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/Sale>

Request: Swipe

```
{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "7712950000000000353=250110118",
  "Purchase": "1.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Request: Keyed (manual)

Substitute the **Track2** data with the hand-keyed account number as follows:

```
"AcctNo": "7712950000000000353",
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Approved",
  "UserTraceData": "",
  "MerchantID": "595901",
  "TranType": "PrePaid",
  "TranCode": "Sale",
  "InvoiceNo": "1",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "771295XXXXXXXXX0353",
  "RefNo": "564649",
  "AuthCode": "564649",
  "Authorize": "1.00",
  "Purchase": "1.00",
  "Balance": "4.28"
}
```

/PrePaid/Sale – Encryption

A **PrePaid Sale** takes value off of the card, reducing the available balance by that purchase amount; declines the transaction if there is insufficient balance. (See **NoNSFSale** for an alternative option)

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/Sale>

Request: Swipe

```
{
  "InvoiceNo": "2",
  "RefNo": "2",
  "Memo": "MPS Example JSON v1.0",
  "EncryptedFormat": "MagneSafe",
  "AccountSource": "Swiped",
  "EncryptedBlock": "C8C8F9536826D5450E734953206E7F4DC6812C6858037F5ABF23D9F83F948AF7",
}
```

```
"EncryptedKey":"9012090B06349B000056",
"Purchase":"1.00",
"TerminalName":"MPS Terminal",
"ShiftID":"MPS Shift",
"OperatorID":"MPS Operator"
}
```

Request: Keyed (manual)

Substitute the **AccountSource** as follows:

"AccountSource":"Keyed",
Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Approved",
  "UserTraceData": "",
  "MerchantID": "395347305=E2E",
  "TranType": "PrePaid",
  "TranCode": "Sale",
  "InvoiceNo": "2",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "605011XXXXXXXXX0146",
  "RefNo": "566986",
  "AuthCode": "566986",
  "Authorize": "1.00",
  "Purchase": "1.00",
  "Balance": "99.00"
}
```

/PrePaid/VoidSale – Clear-Text

A **PrePaid VoidSale** cancels a previously approved Sale.

- The *VoidSale* must specify the **RefNo** (reference number) and **AuthCode** (authorization code) returned in the originating transaction response, as well as the card number and original transaction amount.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/VoidSale>

Request: Swipe

```
{
  "InvoiceNo":"1",
  "RefNo":"564649",
  "Memo":"MPS Example JSON v1.0",
  "Track2":"771295000000000000353=250110118",
  "Purchase":"1.00",
  "TerminalName":"MPS Terminal",
  "ShiftID":"MPS Shift",
  "OperatorID":"MPS Operator",
  "AuthCode":"564649"
}
```

Request: Keyed (manual)

Substitute the **Track2** data with the hand-keyed account number as follows:

"AcctNo":"771295000000000000353",
Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
```

```

"CmdStatus": "Approved",
"TextResponse": "Voided",
"UserTraceData": "",
"MerchantID": "595901",
"TranType": "PrePaid",
"TranCode": "VoidSale",
"InvoiceNo": "1",
"TerminalName": "MPS Terminal",
"OperatorID": "MPS Operator",
"AcctNo": "771295XXXXXXXXX0353",
"RefNo": "564649",
"Balance": "5.28",
"Authorize": "1.00",
"Purchase": "1.00"
}

```

/PrePaid/VoidSale – Encryption

A **PrePaid VoidSale** cancels a previously approved Sale.

- The *VoidSale* must specify the **RefNo** (reference number) and **AuthCode** (authorization code) returned in the originating transaction response, as well as the card number and original transaction amount.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/VoidSale>

Request: Swipe

```

{
  "InvoiceNo": "2",
  "RefNo": "566986",
  "Memo": "MPS Example JSON v1.0",
  "EncryptedFormat": "MagneSafe",
  "AccountSource": "Swiped",
  "EncryptedBlock": "C8C8F9536826D5450E734953206E7F4DC6812C6858037F5ABF23D9F83F948AF7",
  "EncryptedKey": "9012090B06349B000056",
  "Purchase": "1.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator",
  "AuthCode": "566986"
}

```

Request: Keyed (manual)

Substitute the **AccountSource** as follows:

"AccountSource": "Keyed",

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Voided",
  "UserTraceData": "",
  "MerchantID": "395347305=E2E",
  "TranType": "PrePaid",
  "TranCode": "VoidSale",
  "InvoiceNo": "2",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "605011XXXXXXXXX0146",
  "RefNo": "566986",
  "Balance": "100.00",
  "Authorize": "1.00",
}

```



```
"Purchase": "1.00"
}
```

/PrePaid/NoNSFSale – Clear-Text

A **PrePaid NoNSFSale** takes value off of the card, reducing the available balance by that purchase amount.

- If there is an insufficient account balance, the transaction will approve for the remaining balance on the card.
- Implementing a NoNSFSale requires the POS to manage a “Balance Due” split-tender if there is an insufficient balance on the card to cover the total purchase amount.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/NoNSFSale>

Request: Swipe

```
{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "7712950000000000353=250110118",
  "Purchase": "1.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Request: Keyed (manual)

Substitute the **Track2** data with the hand-keyed account number as follows:

```
"AcctNo": "7712950000000000353",
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Approved",
  "UserTraceData": "",
  "MerchantID": "595901",
  "TranType": "PrePaid",
  "TranCode": "NoNSFSale",
  "InvoiceNo": "1",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "771295XXXXXXXXX0353",
  "RefNo": "564651",
  "AuthCode": "564651",
  "Authorize": "1.00",
  "Purchase": "1.00",
  "Balance": "4.28"
}
```

/PrePaid/NoNSFSale – Encryption

A **PrePaid NoNSFSale** takes value off of the card, reducing the available balance by that purchase amount.

- If there is an insufficient account balance, the transaction will approve for the remaining balance on the card.
- Implementing a NoNSFSale requires the POS to manage a “Balance Due” split-tender if there is an insufficient balance on the card to cover the total purchase amount.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/NoNSFSale>

Request: Swipe

```
{
  "InvoiceNo": "3",
  "RefNo": "3",
  "Memo": "MPS Example JSON v1.0",
  "EncryptedFormat": "MagneSafe",
  "AccountSource": "Swiped",
  "EncryptedBlock": "C8C8F9536826D5450E734953206E7F4DC6812C6858037F5ABF23D9F83F948AF7",
  "EncryptedKey": "9012090B06349B000056",
  "Purchase": "1.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Request: Keyed (manual)

Substitute the **AccountSource** as follows:

```
"AccountSource": "Keyed",
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Approved",
  "UserTraceData": "",
  "MerchantID": "395347305=E2E",
  "TranType": "PrePaid",
  "TranCode": "NoNSFSale",
  "InvoiceNo": "3",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "605011XXXXXXXXX0146",
  "RefNo": "566988",
  "AuthCode": "566988",
  "Authorize": "1.00",
  "Purchase": "1.00",
  "Balance": "99.00"
}
```

/PrePaid/Reload – Clear-Text

A **PrePaid Reload** adds value to a card, increasing the balance. The default allowable parameter is two reloads per day recommended for liability.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/Reload>

Request: Swipe

```
{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "7712950000000000353=250110118",
  "Purchase": "20.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Request: Keyed (manual)

Substitute the **Track2** data with the hand-keyed account number as follows:

```
"AcctNo": "7712950000000000353",
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Approved",
  "UserTraceData": "",
  "MerchantID": "595901",
  "TranType": "PrePaid",
  "TranCode": "Reload",
  "InvoiceNo": "1",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "771295XXXXXXXXX0353",
  "RefNo": "564662",
  "AuthCode": "564662",
  "Authorize": "20.00",
  "Purchase": "20.00",
  "Balance": "24.28"
}
```

/PrePaid/Reload – Encryption

A **PrePaid Reload** adds value to a card, increasing the balance. The default allowable parameter is two reloads per day recommended for liability.

Resource URL: **POST** <https://w1.mercurycert.net/PaymentsAPI/PrePaid/Reload>

Request: **Swipe**

```
{
  "InvoiceNo": "4",
  "RefNo": "4",
  "Memo": "MPS Example JSON v1.0",
  "EncryptedFormat": "MagneSafe",
  "AccountSource": "Swiped",
  "EncryptedBlock": "C8C8F9536826D5450E734953206E7F4DC6812C6858037F5ABF23D9F83F948AF7",
  "EncryptedKey": "9012090B06349B000056",
  "Purchase": "1.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Request: **Keyed (manual)**

Substitute the **AccountSource** as follows:

"AccountSource": "Keyed",

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Approved",
  "UserTraceData": "",
  "MerchantID": "395347305=E2E",
  "TranType": "PrePaid",
  "TranCode": "Reload",
  "InvoiceNo": "4",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "605011XXXXXXXXXX0146",
  "RefNo": "566989",
  "AuthCode": "566989",
}
```

```
"Authorize": "1.00",
"Purchase": "1.00",
"Balance": "100.00"
}
```

/PrePaid/VoidReload – Clear-Text

A **PrePaid VoidReload** cancels a previously approved *Reload*.

- The *VoidReload* must specify the **RefNo** (reference number) and **AuthCode** (authorization code) returned in the originating transaction response, as well as the card number and original transaction amount.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/VoidReload>

Request: Swipe

```
{
  "InvoiceNo": "1",
  "RefNo": "564662",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "7712950000000000353=250110118",
  "Purchase": "20.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator",
  "AuthCode": "564662"
}
```

Request: Keyed (manual)

Substitute the **Track2** data with the hand-keyed account number as follows:

```
"AcctNo": "7712950000000000353",
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Voided",
  "UserTraceData": "",
  "MerchantID": "595901",
  "TranType": "PrePaid",
  "TranCode": "VoidReload",
  "InvoiceNo": "1",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "771295XXXXXXXXX0353",
  "RefNo": "564662",
  "Balance": "4.28",
  "Authorize": "20.00",
  "Purchase": "20.00"
}
```

/PrePaid/VoidReload

A **PrePaid VoidReload** cancels a previously approved *Reload*.

- The *VoidReload* must specify the **RefNo** (reference number) and **AuthCode** (authorization code) returned in the originating transaction response, as well as the card number and original transaction amount.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/VoidReload>

Request: Swipe

```
{
  "InvoiceNo": "4",
  "RefNo": "566989",
  "Memo": "MPS Example JSON v1.0",
  "EncryptedFormat": "MagneSafe",
  "AccountSource": "Swiped",
  "EncryptedBlock": "C8C8F9536826D5450E734953206E7F4DC6812C6858037F5ABF23D9F83F948AF7",
  "EncryptedKey": "9012090B06349B000056",
  "Purchase": "1.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator",
  "AuthCode": "566989"
}
```

Request: Keyed (manual)

Substitute the **AccountSource** as follows:

```
"AccountSource": "Keyed",
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Voided",
  "UserTraceData": "",
  "MerchantID": "395347305=E2E",
  "TranType": "PrePaid",
  "TranCode": "VoidReload",
  "InvoiceNo": "4",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "605011XXXXXXXXX0146",
  "RefNo": "566989",
  "Balance": "99.00",
  "Authorize": "1.00",
  "Purchase": "1.00"
}
```

/PrePaid/Return – Clear-Text

A **PrePaid Return** adds value to a card, increasing the balance. Used for merchandise returns.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/Return>

Request: Swipe

```
{
  "InvoiceNo": "2",
  "RefNo": "2",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "7712950000000000353=250110118",
  "Purchase": "20.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}
```

Request: Keyed (manual)

Substitute the **Track2** data with the hand-keyed account number as follows:

```
"AcctNo": "7712950000000000353",
```

Response

```
{
```

```

"ResponseOrigin": "Processor",
"DSIXReturnCode": "000000",
"CmdStatus": "Approved",
"TextResponse": "Approved",
"UserTraceData": "",
"MerchantID": "595901",
"TranType": "PrePaid",
"TranCode": "Return",
"InvoiceNo": "2",
"TerminalName": "MPS Terminal",
"OperatorID": "MPS Operator",
"AcctNo": "771295XXXXXXXXX0353",
"RefNo": "564699",
"AuthCode": "564699",
"Authorize": "1.00",
"Purchase": "1.00",
"Balance": "25.28"
}

```

/PrePaid/Return – Encryption

A **PrePaid Return** adds value to a card, increasing the balance. Used for merchandise returns.

Resource URL: **POST** <https://w1.mercurycert.net/PaymentsAPI/PrePaid/Return>

Request: **Swipe**

```

{
  "InvoiceNo": "5",
  "RefNo": "5",
  "Memo": "MPS Example JSON v1.0",
  "EncryptedFormat": "MagneSafe",
  "AccountSource": "Swiped",
  "EncryptedBlock": "C8C8F9536826D5450E734953206E7F4DC6812C6858037F5ABF23D9F83F948AF7",
  "EncryptedKey": "9012090B06349B000056",
  "Purchase": "1.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}

```

Request: **Keyed (manual)**

Substitute the **AccountSource** as follows:

"AccountSource": "Keyed",

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Approved",
  "UserTraceData": "",
  "MerchantID": "395347305=E2E",
  "TranType": "PrePaid",
  "TranCode": "Return",
  "InvoiceNo": "5",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "605011XXXXXXXXX0146",
  "RefNo": "566991",
  "AuthCode": "566991",
  "Authorize": "1.00",
  "Purchase": "1.00",
}

```



```
"Balance": "100.00"
}
```

/PrePaid/VoidReturn – Clear-Text

A **PrePaid VoidReturn** cancels a previously approved *Return*.

- The *VoidReturn* must specify the **RefNo** (reference number) and **AuthCode** (authorization code) returned in the originating transaction response, as well as the card number and original transaction amount.

Resource URL: **POST** <https://w1.mercurycert.net/PaymentsAPI/PrePaid/VoidReturn>

Request: **Swipe**

```
{
  "InvoiceNo": "2",
  "RefNo": "564699",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "7712950000000000353=250110118",
  "Purchase": "1.00",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator",
  "AuthCode": "564699"
}
```

Request: **Keyed (manual)**

Substitute the **Track2** data with the hand-keyed account number as follows:

```
"AcctNo": "7712950000000000353",
```

Response

```
{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Voided",
  "UserTraceData": "",
  "MerchantID": "595901",
  "TranType": "PrePaid",
  "TranCode": "VoidReturn",
  "InvoiceNo": "2",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "771295XXXXXXXXX0353",
  "RefNo": "564699",
  "Balance": "24.28",
  "Authorize": "1.00",
  "Purchase": "1.00"
}
```

/PrePaid/VoidReturn – Encryption

A **PrePaid VoidReturn** cancels a previously approved *Return*.

- The *VoidReturn* must specify the **RefNo** (reference number) and **AuthCode** (authorization code) returned in the originating transaction response, as well as the card number and original transaction amount.

Resource URL: **POST** <https://w1.mercurycert.net/PaymentsAPI/PrePaid/VoidReturn>

Request: **Swipe**

```
{
  "InvoiceNo": "5",
```

```

"RefNo": "566991",
"Memo": "MPS Example JSON v1.0",
"EncryptedFormat": "MagneSafe",
"AccountSource": "Swiped",
"EncryptedBlock": "C8C8F9536826D5450E734953206E7F4DC6812C6858037F5ABF23D9F83F948AF7",
"EncryptedKey": "9012090B06349B000056",
"Purchase": "1.00",
"TerminalName": "MPS Terminal",
"ShiftID": "MPS Shift",
"OperatorID": "MPS Operator",
"AuthCode": "566991"
}

```

Request: Keyed (manual)

Substitute the **AccountSource** as follows:

```
"AccountSource": "Keyed",
```

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Voided",
  "UserTraceData": "",
  "MerchantID": "395347305=E2E",
  "TranType": "PrePaid",
  "TranCode": "VoidReturn",
  "InvoiceNo": "5",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "605011XXXXXXXXX0146",
  "RefNo": "566991",
  "Balance": "99.00",
  "Authorize": "1.00",
  "Purchase": "1.00"
}

```

/PrePaid/Balance – Clear-Text

A **PrePaid Balance** returns the remaining balance of a card.

Resource URL: POST <https://w1.mercurycert.net/PaymentsAPI/PrePaid/Balance>

Request: Swipe

```

{
  "InvoiceNo": "1",
  "RefNo": "1",
  "Memo": "MPS Example JSON v1.0",
  "Track2": "7712950000000000353=250110118",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}

```

Request: Keyed (manual)

Substitute the **Track2** data with the hand-keyed account number as follows:

```
"AcctNo": "7712950000000000353",
```

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",

```

```

    "TextResponse": "Approved",
    "UserTraceData": "",
    "MerchantID": "595901",
    "TranType": "PrePaid",
    "TranCode": "Balance",
    "InvoiceNo": "1",
    "TerminalName": "MPS Terminal",
    "OperatorID": "MPS Operator",
    "AcctNo": "771295XXXXXXXXX0353",
    "Balance": "24.28"
  }

```

/PrePaid/Balance -- Encryption

A **PrePaid Balance** returns the remaining balance of a card.

Resource URL: **POST** <https://w1.mercurycert.net/PaymentsAPI/PrePaid/Balance>

Request: **Swipe**

```

{
  "InvoiceNo": "6",
  "RefNo": "6",
  "Memo": "MPS Example JSON v1.0",
  "EncryptedFormat": "MagneSafe",
  "AccountSource": "Swiped",
  "EncryptedBlock": "C8C8F9536826D5450E734953206E7F4DC6812C6858037F5ABF23D9F83F948AF7",
  "EncryptedKey": "9012090B06349B000056",
  "TerminalName": "MPS Terminal",
  "ShiftID": "MPS Shift",
  "OperatorID": "MPS Operator"
}

```

Request: **Keyed (manual)**

Substitute the **AccountSource** as follows:

"AccountSource": "Keyed",

Response

```

{
  "ResponseOrigin": "Processor",
  "DSIXReturnCode": "000000",
  "CmdStatus": "Approved",
  "TextResponse": "Approved",
  "UserTraceData": "",
  "MerchantID": "395347305=E2E",
  "TranType": "PrePaid",
  "TranCode": "Balance",
  "InvoiceNo": "6",
  "TerminalName": "MPS Terminal",
  "OperatorID": "MPS Operator",
  "AcctNo": "605011XXXXXXXXX0146",
  "Balance": "99.00"
}

```

Web Services Direct for StoreCard/Gift Transactions

There are two Web Services URLs . Worldpay/MercuryPay's Web Services **W1** and **W2** servers are geographically separate, yet act as parallel, fully functioning entry points to Worldpay's processing platform.

Server	URL
Development Testing W1 Primary Server (Only W1 is used in testing)	https://w1.mercurycert.net/ws/ws.asmx
Production Web Services W1	https://w1.mercurypay.com/ws/ws.asmx

Production Web Services W2

<https://w2.backuppays.com/ws/ws.asmx>**Worldpay/MercuryPay Web Services Direct Libraries (WSDL)**

Development Test/Certification WSDL

<https://w1.mercurycert.net/ws/ws.asmx?WSDL>

Production WSDL

<https://w1.mercurypay.com/ws/ws.asmx?WSDL>**Using Worldpay's Web Services Libraries**

Worldpay's Web Service development URL is <https://w1.mercurycert.net/ws/ws.asmx> with the full WSDL accessed at: <https://w1.mercurycert.net/ws/ws.asmx?WSDL>

Any development environment supporting Web Services can add a reference to the above URL and have access to the methods and classes it provides.

Gift Transaction Web Services Methods and Arguments

The XML string specifies the necessary transaction information (payload) that is sent as an argument of the methods.

WS Method	Purpose and Argument Description
GiftTransaction()	<p>Send a gift card transaction</p> <p>GiftTransaction(trans as string, password as string)</p> <p>Two arguments:</p> <ol style="list-style-type: none"> 1. tran is an XML string formatted with transaction details as specified in the DSIClientX™ PI Spec 2. pw is a password string that Worldpay assigns for a particular <i>MerchantID</i> <p>Example: s = x.GiftTransaction(tran,"xyz")</p>

Note IP Port 9100 must be included in the soap-wrapped *GiftTransaction()* request

Web Services Gift/Prepaid and StoreCard Payload Examples

- Swiped Gift Issue
- Manual Gift NoNSFSale
- Swiped Gift NoNSFSale
- Gift VoidSale of Previous Manual Transaction
- Swiped Gift Reload
- Manual Gift Sale using CVVData

Web Services: Swiped Gift Issue**Request**

```
User-Agent: MPS Transact 1.2.0.5
Content-Type: text/xml; charset=utf-8
SOAPAction: http://www.mercurypay.com/GiftTransaction
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
  <GiftTransaction xmlns="http://www.mercurypay.com">
    <tran>&lt;?xml version="1.0"?&gt;
      &lt;TStream&gt;
        &lt;Transaction&gt;
          &lt;IpPort&gt;9100&lt;/IpPort&gt;
          &lt;MerchantID&gt;003503902913105&lt;/MerchantID&gt;
          &lt;OperatorID&gt;test&lt;/OperatorID&gt;
          &lt;TranType&gt;PrePaid&lt;/TranType&gt;
          &lt;TranCode&gt;Issue&lt;/TranCode&gt;
          &lt;InvoiceNo&gt;123475&lt;/InvoiceNo&gt;
          &lt;RefNo&gt;123475&lt;/RefNo&gt;
```

```

        <Memo>POS.v345</Memo>
        <Account>
        <Track2>6050110010021825120=250110117</Track2>
        </Account>
        <Amount>
            <Purchase>200.00</Purchase>
        </Amount>
    </Transaction>
    </TStream></tran>
</pw></pw>
</GiftTransaction>
</soap:Body>
</soap:Envelope>

```

Response

```

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><GiftTransactionResponse
xmlns="http://www.mercurypay.com"><GiftTransactionResult><?xml version="1.0"?>
    <RStream>
        <CmdResponse>
            <ResponseOrigin>Processor</ResponseOrigin>
            <DSIXReturnCode>000000</DSIXReturnCode>
            <CmdStatus>Approved</CmdStatus>
            <TextResponse>Approved</TextResponse>
            <UserTraceData></UserTraceData>
        </CmdResponse>
        <TranResponse>
            <MerchantID>003503902913105</MerchantID>
            <TranType>PrePaid</TranType>
            <TranCode></TranCode>
            <InvoiceNo>123475</InvoiceNo>
            <OperatorID>test</OperatorID>
            <AcctNo>6050110010021825120</AcctNo>
            <RefNo>532136</RefNo>
            <AuthCode>532136</AuthCode>
            <Amount>
                <Authorize>200.00</Authorize>
                <Purchase>200.00</Purchase>
                <Balance>200.00</Balance>
            </Amount>
        </TranResponse>
    </RStream>
</GiftTransactionResult></GiftTransactionResponse></soap:Body></soap:Envelope>

```

Note: If you have implemented E2E or MToken requiring E2E or MToken enabled *MerchantIDs*, the request may be sent, for consistency in coding, either with or without the encrypted or token data fields.

- A token is never generated or returned on gift transactions.
- Clear-text gift card data will be returned in the *AcctNo* response tag and may be stored for subsequent use.

Web Services: MANUAL Gift NoNSFSale

Request

User-Agent: MPS Transact 1.2.0.5
Content-Type: text/xml; charset=utf-8
SOAPAction: http://www.mercurypay.com/GiftTransaction

```
<?xml version="1.0" encoding="utf-8"?>
```

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<soap:Body>
  <GiftTransaction xmlns="http://www.mercurypay.com">
    <tran>&lt;?xml version="1.0"?&gt;
      &lt;TStream&gt;
        &lt;Transaction&gt;
          &lt;IpPort&gt;9100&lt;/IpPort&gt;
          &lt;MerchantID&gt;003503902913105&lt;/MerchantID&gt;
          &lt;OperatorID&gt;test&lt;/OperatorID&gt;
          &lt;TranType&gt;PrePaid&lt;/TranType&gt;
          &lt;TranCode&gt;NoNSFSale&lt;/TranCode&gt;
          &lt;InvoiceNo&gt;123476&lt;/InvoiceNo&gt;
          &lt;RefNo&gt;123476&lt;/RefNo&gt;
          &lt;Memo&gt;POS.v345&lt;/Memo&gt;
          &lt;Account&gt;
            &lt;AcctNo&gt;6050110010021825120&lt;/AcctNo&gt;
          &lt;/Account&gt;
          &lt;Amount&gt;
            &lt;Purchase&gt;4.00&lt;/Purchase&gt;
          &lt;/Amount&gt;
        &lt;/Transaction&gt;
      &lt;/TStream&gt;</tran>
    </GiftTransaction>
  </soap:Body>

</soap:Envelope>

```

Response

```

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><GiftTransactionResponse
xmlns="http://www.mercurypay.com"><GiftTransactionResult>&lt;?xml version="1.0"?&gt;

&lt;RStream&gt;
  &lt;CmdResponse&gt;
    &lt;ResponseOrigin&gt;Processor&lt;/ResponseOrigin&gt;
    &lt;DSIXReturnCode&gt;000000&lt;/DSIXReturnCode&gt;
    &lt;CmdStatus&gt;Approved&lt;/CmdStatus&gt;
    &lt;TextResponse&gt;Approved&lt;/TextResponse&gt;
    &lt;UserTraceData&gt;&lt;/UserTraceData&gt;
  &lt;/CmdResponse&gt;
  &lt;TranResponse&gt;
    &lt;MerchantID&gt;003503902913105&lt;/MerchantID&gt;
    &lt;TranType&gt;PrePaid&lt;/TranType&gt;
    &lt;TranCode&gt;NoNSFSale&lt;/TranCode&gt;
    &lt;InvoiceNo&gt;123476&lt;/InvoiceNo&gt;
    &lt;OperatorID&gt;test&lt;/OperatorID&gt;
    &lt;AcctNo&gt;6050110010021825120&lt;/AcctNo&gt;
    &lt;RefNo&gt;532137&lt;/RefNo&gt;
    &lt;AuthCode&gt;532137&lt;/AuthCode&gt;
    &lt;Amount&gt;
      &lt;Authorize&gt;4.00&lt;/Authorize&gt;
      &lt;Purchase&gt;4.00&lt;/Purchase&gt;
      &lt;Balance&gt;196.00&lt;/Balance&gt;
    &lt;/Amount&gt;
  &lt;/TranResponse&gt;
&lt;/RStream&gt;
</GiftTransactionResult></GiftTransactionResponse></soap:Body></soap:Envelope>

```


Web Services: SWIPED Gift NoNSFSale**Request**

```

User-Agent: MPS Transact 1.2.0.5
Content-Type: text/xml; charset=utf-8
SOAPAction: http://www.mercurypay.com/GiftTransaction
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
  <GiftTransaction xmlns="http://www.mercurypay.com">
    <tran>&lt;?xml version="1.0"?&gt;
      &lt;TStream&gt;
        &lt;Transaction&gt;
          &lt;IpPort&gt;9100&lt;/IpPort&gt;
          &lt;MerchantID&gt;595901&lt;/MerchantID&gt;
          &lt;OperatorID&gt;test&lt;/OperatorID&gt;
          &lt;TranType&gt;PrePaid&lt;/TranType&gt;
          &lt;TranCode&gt;NoNSFSale&lt;/TranCode&gt;
          &lt;InvoiceNo&gt;9&lt;/InvoiceNo&gt;
          &lt;RefNo&gt;9&lt;/RefNo&gt;
          &lt;Memo&gt;MPS Transact 1.2.0.5&lt;/Memo&gt;
          &lt;Account&gt;
            &lt;Track2&gt;6050110000012485491=250110117&lt;/Track2&gt;
            &lt;/Account&gt;
            &lt;Amount&gt;
              &lt;Purchase&gt;2.25&lt;/Purchase&gt;
              &lt;/Amount&gt;
            &lt;/Transaction&gt;
          &lt;/TStream&gt;</tran>
        <pw>xyz</pw>
      </GiftTransaction>
    </soap:Body></soap:Envelope>

```

Response

```

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><GiftTransactionResponse
xmlns="http://www.mercurypay.com"><GiftTransactionResult>&lt;?xml version="1.0"?&gt;
&lt;RStream&gt;
  &lt;CmdResponse&gt;
    &lt;ResponseOrigin&gt;Processor&lt;/ResponseOrigin&gt;
    &lt;DSIXReturnCode&gt;000000&lt;/DSIXReturnCode&gt;
    &lt;CmdStatus&gt;Approved&lt;/CmdStatus&gt;
    &lt;TextResponse&gt;Approved&lt;/TextResponse&gt;
    &lt;UserTraceData&gt;&lt;/UserTraceData&gt;
  &lt;/CmdResponse&gt;
  &lt;TranResponse&gt;
    &lt;MerchantID&gt;595901&lt;/MerchantID&gt;
    &lt;TranType&gt;PrePaid&lt;/TranType&gt;
    &lt;TranCode&gt;NoNSFSale&lt;/TranCode&gt;
    &lt;InvoiceNo&gt;9&lt;/InvoiceNo&gt;
    &lt;OperatorID&gt;test&lt;/OperatorID&gt;
    &lt;AcctNo&gt;6050110000012485491&lt;/AcctNo&gt;
    &lt;RefNo&gt;469955&lt;/RefNo&gt;
    &lt;AuthCode&gt;469955&lt;/AuthCode&gt;
    &lt;Amount&gt;
      &lt;Authorize&gt;2.25&lt;/Authorize&gt;
      &lt;Purchase&gt;2.25&lt;/Purchase&gt;
      &lt;Balance&gt;79.25&lt;/Balance&gt;
    &lt;/Amount&gt;
  &lt;/TranResponse&gt;
&lt;/RStream&gt;
&lt;/GiftTransactionResult>
&lt;/GiftTransactionResponse>
</soap:Body></soap:Envelope>

```

```

        </Amount>
    </TranResponse>
</RStream>
</GiftTransactionResult></GiftTransactionResponse></soap:Body></soap:Envelope>

```

Web Services: Gift VoidSale of Previous Manual Transaction

Request

```

User-Agent: MPS Transact 1.2.0.5
Content-Type: text/xml; charset=utf-8
SOAPAction: http://www.mercurypay.com/GiftTransaction
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
    <GiftTransaction xmlns="http://www.mercurypay.com">
        <tran><?xml version="1.0"?>
            <TStream>
                <Transaction>
                    <IpPort>9100</IpPort>
                    <MerchantID>003503902913105</MerchantID>
                    <OperatorID>test</OperatorID>
                    <TranType>PrePaid</TranType>
                    <TranCode>VoidSale</TranCode>
                    <InvoiceNo>123476</InvoiceNo>
                    <RefNo>532137</RefNo>
                    <Memo>POS.v345</Memo>
                    <Account>
                        <AcctNo>6050110010021825120</AcctNo>
                    </Account>
                    <Amount>
                        <Purchase>4.00</Purchase>
                    </Amount>
                    <TranInfo>
                        <AuthCode>532137</AuthCode>
                    </TranInfo>
                </Transaction>
            </TStream>
        </tran>
    </GiftTransaction>
</soap:Body>
</soap:Envelope>

```

Response

```

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><GiftTransactionResponse
xmlns="http://www.mercurypay.com"><GiftTransactionResult><?xml version="1.0"?>
    <RStream>
        <CmdResponse>
            <ResponseOrigin>Processor</ResponseOrigin>
            <DSIXReturnCode>000000</DSIXReturnCode>
            <CmdStatus>Approved</CmdStatus>
            <TextResponse>Voided</TextResponse>
            <UserTraceData></UserTraceData>
        </CmdResponse>
        <TranResponse>
            <MerchantID>003503902913105</MerchantID>

```

```

        <TranType>PrePaid</TranType>
        <TranCode>VoidSale</TranCode>
        <InvoiceNo>123476</InvoiceNo>
        <OperatorID>test</OperatorID>
        <AcctNo>6050110010021825120</AcctNo>
        <RefNo>532137</RefNo>
        <Balance>200.00</Balance>
        <Amount>
            <Authorize>4.00</Authorize>
            <Purchase>4.00</Purchase>
            <Balance>200.00</Balance>
        </Amount>
    </TranResponse>
</RStream>
</GiftTransactionResult></GiftTransactionResponse></soap:Body></soap:Envelope>

```

Web Services: Swiped Gift Reload

Request

User-Agent: MPS Transact 1.2.0.5
 Content-Type: text/xml; charset=utf-8
 SOAPAction: http://www.mercurypay.com/GiftTransaction

```

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
        <GiftTransaction xmlns="http://www.mercurypay.com">
            <tran>
                <?xml version="1.0"?>
                    <TStream>
                        <Transaction>
                            <IpPort>9100</IpPort>
                            <MerchantID>003503902913105</MerchantID>
                            <OperatorID>test</OperatorID>
                            <TranType>PrePaid</TranType>
                            <TranCode>Reload</TranCode>
                            <InvoiceNo>123478</InvoiceNo>
                            <RefNo>123478</RefNo>
                            <Memo>POS.v345</Memo>
                            <Account>
                                <Track2>6050110010021825120=250110117</Track2>
                                <Account>
                                    <Amount>
                                        <Purchase>50.00</Purchase>
                                    </Amount>
                                </Account>
                            </Transaction>
                        </TStream>
                    </tran>
                </GiftTransaction>
            </soap:Body>
        </soap:Envelope>
    
```

Response

```

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
        <GiftTransactionResponse
            xmlns="http://www.mercurypay.com">
            <GiftTransactionResult>
                <?xml version="1.0"?>

```

```

</RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>Approved</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>003503902913105</MerchantID>
    <TranType>PrePaid</TranType>
    <TranCode>Reload</TranCode>
    <InvoiceNo>123478</InvoiceNo>
    <OperatorID>test</OperatorID>
    <AcctNo>6050110010021825120</AcctNo>
    <RefNo>532139</RefNo>
    <AuthCode>532139</AuthCode>
    <Amount>
      <Authorize>50.00</Authorize>
      <Purchase>50.00</Purchase>
      <Balance>250.00</Balance>
    </Amount>
  </TranResponse>
</RStream>
</GiftTransactionResult></GiftTransactionResponse></soap:Body></soap:Envelope>

```

Web Services: Manual Gift Sale using CVVData

Request

```

<soap:Body>
  <GiftTransaction xmlns="http://www.mercurypay.com">
    <tran><?xml version="1.0"?>
      <TStream>
        <Transaction>
          <IpPort>9100</IpPort>
          <MerchantID>003503902913105</MerchantID>
          <TerminalID>003503902913105</TerminalID>
          <OperatorID>test</OperatorID>
          <TranType>PrePaid</TranType>
          <TranCode>Sale</TranCode>
          <InvoiceNo>123483</InvoiceNo>
          <RefNo>123483</RefNo>
          <Memo>POS.v345</Memo>
          <Account>
            <AcctNo>6050110000002570798</AcctNo>
          </Account>
          <Amount>
            <Purchase>3.00</Purchase>
          </Amount>
          <CVVData>880</CVVData>
        </Transaction>
      </TStream></tran>
    </GiftTransaction>
  </soap:Body>
</soap:Envelope>

```

Response

```

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><GiftTransactionResponse

```

```

xmlns="http://www.mercurypay.com"><GiftTransactionResult>&lt;?xml version="1.0"?&gt;
&lt;RStream&gt;
  &lt;CmdResponse&gt;
    &lt;ResponseOrigin&gt;Processor&lt;/ResponseOrigin&gt;
    &lt;DSIXReturnCode&gt;000000&lt;/DSIXReturnCode&gt;
    &lt;CmdStatus&gt;Approved&lt;/CmdStatus&gt;
    &lt;TextResponse&gt;Approved&lt;/TextResponse&gt;
    &lt;UserTraceData&gt;&lt;/UserTraceData&gt;
  &lt;/CmdResponse&gt;
  &lt;TranResponse&gt;
    &lt;MerchantID&gt;003503902913105&lt;/MerchantID&gt;
    &lt;TranType&gt;PrePaid&lt;/TranType&gt;
    &lt;TranCode&gt;Sale&lt;/TranCode&gt;
    &lt;InvoiceNo&gt;123483&lt;/InvoiceNo&gt;
    &lt;OperatorID&gt;test&lt;/OperatorID&gt;
    &lt;AcctNo&gt;6050110000002570798&lt;/AcctNo&gt;
    &lt;RefNo&gt;532155&lt;/RefNo&gt;
    &lt;AuthCode&gt;532155&lt;/AuthCode&gt;
    &lt;Amount&gt;
      &lt;Authorize&gt;3.00&lt;/Authorize&gt;
      &lt;Purchase&gt;3.00&lt;/Purchase&gt;
      &lt;Balance&gt;197.00&lt;/Balance&gt;
    &lt;/Amount&gt;
    &lt;CVVResult&gt;M&lt;/CVVResult&gt;
  &lt;/TranResponse&gt;
&lt;/RStream&gt;
</GiftTransactionResult></GiftTransactionResponse></soap:Body></soap:Envelope>

```

APPENDICES

Appendix 1 – MercuryPay XML Basics

Transaction Quick References

This section provides a summary of the data elements used in each transaction type.

Structure of a Transaction Request

Most transaction requests consist of a **TranType**, **TranCode**, and additional XML **Elements**. Some elements may consist of **sub-elements**. These are clearly indicated in the tables that follow.

Here is an example of a Transaction Request. This is for a swiped Credit Sale:

```
<?xml version="1.0"?>
<TStream>
  <Transaction>
    <MerchantID>118725340908147</MerchantID>
    <LaneID>02</LaneID>
    <TranType>Credit</TranType>
    <TranCode>Sale</TranCode>
    <InvoiceNo>10</InvoiceNo>
    <RefNo>10</RefNo>
    <Memo> Example XML V1.0</Memo>
    <Account>
      <Track2>Insert Track data</Track2>
    </Account>
    <Amount>
      <Purchase>1.05</Purchase>
    </Amount>
  </Transaction>
</TStream>
```

- **TStream** indicates this is a transaction request.
- **Transaction** is a “wrapper” element that contains all of the details for the transaction.
- **TranType** indicates the type of transaction, which can be *Credit*, *Debit*, *Prepaid*, or *EBT*.
- **TranCode** indicates the action of the transaction, such as *Sale*, *Return*, *VoidSale*, etc.
- Note that the **Account** and **Amount** elements contain sub-elements.

Note: Abbreviations Used in the Tables

Req: is the element *required* (Y) or *optional* (O)

Format: A = alphabetic N = numeric AN = alphanumeric

Basic Elements of a Transaction Request

Standard Datacap XML Elements

All transaction requests, except where noted, contain the elements listed below.

Element	Req	Min/Max	Format	Comments
InvoiceNo	Y	1-16	AN	Unique value per transaction assigned by payment application. This should be separate from check number, table number, etc.
LaneID	Y	1-3	AN	Only required for multi-lane or multi-terminal card present environments.
Memo	Y	1-40	AN	Dynamically build the Memo tag so it pulls the product name and version of the payment application, ensuring it stays current as

				the version changes.
MerchantID	Y	1-24	AN	Unique merchant identifier Can include special characters Format is an 11-15 digit number followed by “=” then 6-digit alpha. Example: 884500XXXX=ABCDEF
OperatorID	O	1-10	AN	Employee associated with the transaction
RefNo	Y	1-4	N	Reference Number supplied by Worldpay in the response; indicates that the transaction has been logged in the merchant's open batch and returned in the response. Optional as filler value in the request for Sale, PreAuth, PreAuthCapture, ZeroAuth and Return.
ShiftID	O	1-20	AN	Shift identification
TerminalID	O	1-24	AN	Local workstation identification. This can match the <i>LaneID</i>
TerminalName	O	1-20	AN	Generally used for workstation identification or payment application identification
TranCode	Y	1-20	A	Payment method (e.g. <i>Sale</i>)
TranType	Y	1-40	A	Payment type (e.g. <i>Credit</i>)

Account

Account is an element that contains different sub-elements depending on whether the transaction is manually entered, swiped clear-text, or swiped through an encrypted device.

Entry Method	Sub Element	Req	Min/Max	Format	Comments
Manual (keyed)	AcctNo	Y	1-19	AN	Account number
	ExpDate	Y	4	N	MMYY format
Swiped: Clear-text	Track2 or Track1	Y	1-37	AN	Data is read from magnetic stripe
Swiped: Encrypted	AccountSource	Y	1-16	AN	has one of three values: <ul style="list-style-type: none"> Swiped keyed Contactless
	EncryptedBlock	Y	72-224	AN	Values are read from encrypted device
	EncryptedKey	Y	20	AN	
	EncryptedFormat	Y	1-25	AN	Value is always MagneSafe

TranInfo

TranInfo is used for transactions that refer to an original transaction (such as voiding a sale). It provides reference information from the original transaction using the following sub-elements:

Sub-element	Req	Min/Max	Format	Comments
AuthCode	Y	1-16	AN	Returned in original transaction response
AcqRefData	Y	1-200	AN	
ProcessData	Y	12-40	AN	
CustomerCode	Only used for Corporate Card II processing	1-17	AN	Value is either the Purchase Order number, Merchant zip code, or invoice number

Using MToken

When using MToken, two elements are *always* included in the request—**RecordNo** and **Frequency**:

Element	Req	Min/Max	Format	Comments
RecordNo	Y	1-100	AN	For the initial token request, value is RecordNumberRequested
Frequency	Y	7-20	AN	Two values: <ul style="list-style-type: none"> OneTime Recurring

For all subsequent transactions, always include BOTH *RecordNo* and *Frequency*, and replace the other XML tags with the same tags ending with “**ByRecordNo**”:

TranCode used to Request a Token	TranCode for Subsequent Use of Token
Adjust	AdjustByRecordNo
CardLookUp	NA
FSASale	FSASaleByRecordNo
PreAuth	PreAuthByRecordNo
PreAuthCapture	PreAuthCaptureByRecordNo
Return	ReturnByRecordNo
ReverseFSASale	ReverseFSASaleByRecordNo
Sale	SaleByRecordNo
VoidReturn	VoidReturnByRecordNo
VoidSale	VoidSaleByRecordNo
ZeroAuth	NA

Basic XML Structure of a Transaction Response

Here is the response for the above Transaction request:

```
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>118725340908147</MerchantID>
    <AcctNo>400555XXXXXX0480</AcctNo>
    <ExpDate>1215</ExpDate>
    <CardType>VISA</CardType>
    <TranCode>Sale</TranCode>
    <Auth Code>VI0100</Auth Code>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>0009</RefNo>
    <InvoiceNo>10</InvoiceNo>
    <Memo>MPS Example XML v1.0</Memo>
    <Amount>
      <Purchase>1.05</Purchase>
      <Authorize>1.05</Authorize>
    </Amount>
    <AcqRefData>b8238953#20#20#20#20#20#20e00</AcqRefData>
    <ProcessData>|00|410100201000</ProcessData>
  </TranResponse>
</RStream>
```

- **RStream** indicates that this is a transaction response. The transaction response consists of two sections:
 - The **CmdResponse** is a quick snapshot of the authorization
 - The **TranResponse** contains more detailed information

Elements of a Transaction Response

Element	Req	Comments
CmdResponse contains the following sub-elements:		
ResponseOrigin	Y	Indicates the source of the response: <ul style="list-style-type: none"> • Client is from the <i>ActiveX</i> control • Server is from MercuryPay's platform server • Processor is from the back-end networks
DSIReturnCode	Y	Contains either a server-specific or DSIClientX-specific six-digit code corresponding to a text message. Details on these codes are provided later in this section.
CmdStatus	Y	Indicates the outcome of the transaction. <ul style="list-style-type: none"> • For <i>Credit, Debit, Prepaid, and EBT</i>: Approved or Declined. • For <i>Admin</i> requests: Success or Error. Error text responses are listed later in this section.
TextResponse	Y	This is dynamically-generated text that explains the CmdStatus .

TranResponse contains a mirror of the original request data, along with Issuer authorization data and processor-specific routing information.

AcctNo and ExpDate	Information from the card swipe. Can be stored with strong encryption until batch close.		
CardType	Indicates the type of card submitted in the request. Values can be VISA, M/C, AMEX, DCLB, DCVR, JCB, DEBIT		
AuthCode	The authorization code supplied by the Issuer. Store this code for subsequent use with a <i>VoidSale, PreAuthCapture</i> , or reversal.		
CaptureStatus	Informs the POS if the transaction was Captured or NotCaptured .		
RefNo	The reference number for any approved transaction in the merchant's current open batch. Store this number for use with subsequent transactions such as <i>Adjust</i> or <i>VoidSale</i> .		
Authorize	Indicates the dollar value authorized for the transaction.		
AcqRefData	1-200	AN	Contains card-brand specific identifiers, market data and transaction routing preferences.
ProcessData	12-40	AN	

Appendix 2 – XML Details by TranType

Credit

TranCode	Comments
Sale	Purchase is authorized and captured in a single step.
Return	Independent transaction (does not need TranInfo). Used for Sales or PreAuth/PreAuthCaptures from closed merchant batches.
PreAuth	Two-step transaction. PreAuth authorizes the transaction, and PreAuthCapture captures it.
PreAuthCapture	Mostly used in restaurants and environments where tips (gratuities) are added to the total.
VoidSale	VoidSale is used to void (cancel) a Sale, PreAuth, PreAuthCapture or VoiceAuth in an open merchant batch. VoidSales should always be sent with AcqRefData and ProcessData in the <i>TranInfo</i> section in order to route the transaction to the Issuer. When routed in this manner the VoidSale qualifies as a Reversal. Note that <i>PartialAuth</i> must be set to "allow" reversal routing.
VoidReturn	voids a <i>Return</i> in the open merchant batch.
Adjust	Adjusts a Sale.
ZeroAuth	Used to validate a card without charging it. Often used in bars and other environments where running tabs are common. Can also be used to request a token.
VoiceAuth	Used if the merchant is required to obtain voice authorization for a transaction.

Additional XML Elements

Element	Min/Max	Format	Comments
CVVData	3-4	N	3 or 4 digit value found on the credit card. It can be used along with AVS to protect against credit card fraud. Issuing bank result codes are returned in the transaction response and can be used for fine-tuning screening for fraud prevention. In the event of unauthorized credit card activity it is critical to implement local POS level settings that enable merchants to respond to mismatched or questionable CVV and AVS results.
AVS			
Address	1-20	AN	Street address only
ZIP	5-9	N	For U.S. zip codes
	6	AN	For Canadian Postal Codes
PartialAuth	5	A	Required on all card present transactions. Syntax is: <PartialAuth>Allow</PartialAuth> Enables acceptance of partial payment AND enables reversal functionality
TranInfo			
AuthCode	1-6	N	Used with <i>VoiceAuth</i> . This is the six-digit authorization code obtained from the credit brand's automated voice authorization system
AcqRefData	1-200	AN	Used in VoidSale/Reversals; includes TransactionID in subfield b
ProcessData	1-20	AN	Pipe delimited Approval code and processing information
ProcTransactionID	9-20	N	Unique transaction identifier
Amount			
Purchase	1-8	N	Purchase dollar amount; two decimals
Tax	1-8	N	Only used for Corporate Card Level II Support; two decimals
Authorize	1-8	N	Authorized amount *Required for PreAuth/PreAuthCapture; two decimals
Gratuity	N/A	N	Indicates the amount of the tip, if any; two decimals

FSA

- FSA falls under the **Credit** TranType.
- **PartialAuth=Allow** is built into this TranCode so there is no need to explicitly include it.
- Developers are encouraged to review FSA related standards, inventory eligibility requirements and membership Services as set forth by the FSA regulatory agency SIGIS available at <https://sig-is.org/>

TranCode	Comments
CardLookup	SIGIS requirement to confirm that the card being processed is an FSA Card. The response will indicate: CardUsage: FSA
FSASale	Indicates that the sale is specifically for FSA
ReverseFSASale	Reverses an FSA sale
VoidSale	Only used if <i>ReverseFSASale</i> returns a response other than “approved.”

Additional XML Elements

Element	Req	Min/Max	Format	Comments
Amount	has two sub-elements:			
• Purchase	Y	1-8	N	Total purchase dollar amount two decimals
• FSA Prescription	Y	1-8	N	dollar amount for FSA purchase component two decimals

DEBIT (U.S. PIN Debit)

PIN entry device required.

TranCode	Comments
Sale	Debit card sale Debit Sale with Cashback—cashback is only allowed on Debit
Return	Debit card return—Debit transactions cannot be Voided

Additional XML Elements

Element	Req	Min/Max	Format	Comments
PIN has two sub-elements: (supplied by Datacap when using dsiEMVUS/dsiPDCX)				
PINBlock	Y	1-20	AN	Encrypted PIN block from PINpad
DervdKey	Y	1-29	AN	Derived unique key from PINpad
Amount has two sub-elements:				
Purchase	Y	1-8	N	Dollar amount of transaction two decimal places
Cashback	O	1-8	N	Allows card holder to receive cash from the debit card Two decimal places

CardLookup

- **CardLookup** is used to determine if a card is Debit, FSA or Other. The *CardLookup* request uses a simplified request format with no *TranCode*. The request includes the *MerchantID* and *Account* fields.
- The response to the *CardLookup* will return one of three values contained in the **CardUsage** data element: **FSA**, **Debit** or **Other**.

U.S. EMV

All U.S. EMV transactions use a chip card that is inserted to the device.

TranCode	Req	Notes
EMVSale	Y	Card present/all-CVM enabled: Signature, No CVM, Offline PIN or Online PIN (for MasterCard)
EMVReturn	Y	
EMVVoidSale (Reversal)	Y	<ul style="list-style-type: none"> EMVVoidSale and EMVVoidReturn may be supported with either card-present or ByRecordNo.
EMVVoidReturn	Y	<ul style="list-style-type: none"> EMVVoidReturn may alternatively be supported by EMVSale, or SaleByRecordNo.
EMVVoiceAuth	O	
EMVZeroAuth	O	Card verification without authorization
AdjustByRecordNo	O	ByRecordNo TranCodes use a token (RecordNo) to adjust a gratuity on an EMVSale, reverse an EMVSale, or void an EMVReturn.
VoidReturnbyRecordNo	O	
VoidSaleByRecordNo	O	
SaleByRecordNo	O	New with V 1.04 or greater – initiates a sale using a stored token.

Prepaid/Gift and StoreCard

Prepaid is used for MercuryPay Gift and StoreCard.

TranCode	Comments
Issue	This is the initial transaction on a card. It activates the account and loads a specified dollar amount onto the card.
Sale	Takes value off of the card, reducing the available balance by that amount. Declines the transaction if there is insufficient balance.
NoNSFSale	Takes value off the card as with a Sale, but does not decline if there is insufficient balance on the card. Instead, the transaction will approve for the available balance on the card. This requires the POS to manage a "balance due" split-tender to cover the remaining purchase amount.
Return	Adds value to a card, increasing the balance. Typically used when merchandise is returned and the value is added to the gift card in place of cash.
Reload	Adds value to a card, increasing the available balance.
Balance	Request to see the available balance on the card.
VoidSale, VoidReload, VoidReturn	Cancels (voids) a previously approved Sale/Reload/Return.
VoidIssue	Cancels the issuance of a card, returning the account balance to zero and putting the card into a non-issued state.

EBT

- EBT has two sub-categories, which are identified by an additional element called **CardType**. The sub-categories are **FoodStamp** or **Cash**.
- EBT, also referred to as SNAP requires a thorough understanding of State and Federal eligibility requirements as outlined on the USDA's Food and Nutrition web site at <https://www.fns.usda.gov/snap/ebt>.
- EWIC is also considered a sub-category of EBT. Contact your Implementations Consultant for our comprehensive supplement guide on supporting eWIC with Datacap.

EBT Cash

TranCode	Comments
Sale	EBT Cash sale. This typically has no item eligibility requirements and does not require an FNS Number to be accepted
CashBack	Optional; enables cardholder to receive cash from the card
Balance	Request balance of value remaining on the card

Additional XML Elements

Element	Req	Min/Max	Format	Comments
Amount has one sub-element:				
• Purchase	Y	1-8	N	Dollar amount two decimal places
PIN has two sub-elements: (Supplied by Datacap when using dsiPDCX)				
• PINBlock	Y	1-20	AN	Encrypted PIN block from PINpad
• DervdKey	Y	1-20	AN	Derived unique key from PINpad

EBT Foodstamp

TranCode	Comments
Sale	Foodstamp sale
Return	Foodstamp return
Balance	Request balance of value remaining on the card.
Voucher (manually entered)	Used if there is a communications failure similar to <i>VoiceAuth</i> . Allows the merchant to obtain voice authorization to complete a food stamp sale.

Additional XML Elements

Element	Req	Min/Max	Format	Comments
Amount has one sub-element:				
• Purchase	Y	N/A	N	Dollar amount two decimal places
PIN has two sub-elements: (Supplied by Datacap when using dsiPDCX)				
• PINBlock	Y	1-20	AN	Encrypted PIN block from PINpad
• DervdKey	Y	1-20	AN	Derived unique key from PINpad
TranInfo has two sub-elements when used with EBT Voucher:				
• AuthCode	Y	1-16	AN	Obtained from voice authorization
• VoucherNo	Y	1-16	AN	Printed voucher number from merchant's voucher booklet

ADMIN

- **<Admin>** is an administrator element. It is *not* a TranType.
- The following information is for batch close procedures - i.e., a command from the POS to initiate the batch close process.
- There are multiple ADMIN level elements.
- For the purpose of this review consider these two corresponding TranCodes used in merchant initiated settlement:

TranCode	Comments
BatchSummary	Used to request a summary of transaction counts and amounts within the batch.
BatchClose	Uses the BatchSummary values to close the merchant's current open batch, enabling the transmission of funds to the merchant.

Additional XML Elements: BatchSummary

Only three elements are required for this request: *MerchantID*, *BatchSummary*, and *Memo* field.

Additional XML Elements: BatchClose

The *BatchClose* will only be successful if the *BatchSummary* count and amount totals match between the merchant's local records and the batch file.

- **Counts** are whole numbers
- **Amounts** are two decimal places, no comma (0000.00)

Element	Req	Min/Max	Format	Comments
BatchNo	Y	1-6	N	This number is obtained from the <i>BatchSummary</i> response
BatchItemCount	Y	1-8	N	Total item count in the batch
NetBatchTotal	Y	1-10	N	Total dollar amount in the batch
CreditPurchaseCount	Y	1-8	N	Total number of credit purchase transactions in the batch
CreditPurchaseAmount	Y	1-10	N	Total dollar amount of credit purchase transactions in the batch
CreditReturnCount	Y	1-8	N	Total number of credit return transactions in the batch
CreditReturnAmount	Y	1-10	N	Total dollar amount of credit return transactions in the batch
DebitPurchaseCount	Y	1-8	N	Total number of debit purchase transactions in the batch
DebitPurchaseAmount	Y	1-10	N	Total dollar amount of debit purchase transactions in the batch
DebitReturnCount	Y	1-8	N	Total number of debit return transactions in the batch
DebitReturnAmount	Y	1-10	N	Total dollar amount of debit return transactions in the batch

Appendix 3 – XML Requests and Responses

Worldpay from FIS/MercuryPay's solutions all are based a similar payload "packet" which can be sent using a number of different integration methods and programming languages. The core payload consists of XML data elements, which MercuryPay derived from Datacap's specifications.

- All interface specifications, along with integration details for dsiPDCX, dsiEMVUS, Canadian DSIEMVClientX, TranCloud are now conveniently incorporated into Datacap's Developer Portal. Go to: <http://www.dsidevportal.com> to register.

Structure of an XML Request

The example below illustrates the typical XML tags. This example shows an **encrypted credit sale** with a **one-time use token request**. Each tag is described below this figure:

```
<?xml version="1.0"?>
<TStream>
  <Transaction>
    ① <MerchantID>118725340908147</MerchantID>
    ② <LanelID>02</LanelID> <!--for environments with multiple input devices-->
    ③ <Tran Type>Credit</Tran Type>
    <Tran Code>Sale</Tran Code>
    ④ <InvoiceNo>10</InvoiceNo>
    <RefNo>10</RefNo>
    ⑤ <Memo>Example XML Memo v1.0</Memo><!--for EMV the<POSPackageID> is used-->
    ⑥ <Frequency>OneTime</Frequency> <!--Frequency and RecordNo are for tokens-->
    <RecordNo>Record Number Requested</RecordNo>
    ⑦ <PartialAuth>Allow</PartialAuth>
    ⑧ <Account>
      <EncryptedFormat>MagneSafe</EncryptedFormat> <!--encrypted swipe-->
      <AccountSource>Swiped</AccountSource>
      <EncryptedBlock>F40DDBA1F645C8DB85A6459D45AFF8002C244A0F
74402B479ABC9915EC9567C81BE99CE4483AF3D</EncryptedBlock>
      <EncryptedKey>9012090B01C4F200002B</EncryptedKey>
    </Account>
    ⑨ <Amount>
      <Purchase>1.05</Purchase>
    </Amount>
    </Transaction>
  </TStream>
```

1. <MerchantID> (MID)

Format: alphanumeric. Length: Max 24. Alpha characters are not case-sensitive.

MercuryPay merchant accounts are set up with a primary Merchant ID that is used to identify the account both internally and externally for transmitting banking information and deposits. The MID is also configured to match your POS system's supported functionality.

- The typical MID format combines an 11-digit numeric value, separated with an '=' and a 6-digit alpha value, such as: **45012345678=ABCDEF**.
- It is possible to have multiple sub-IDs under the primary MID, which are referred to as **Terminal IDs (TID)**. Each individual TID is linked to and routes funds to the primary Merchant account.

MerchantID account settings include:

- Merchant or time initiated settlement,
- EMV support,

- duplicate checking,
- contactless (default is “always on”),
- E2E and Tokenization.

It is important that the merchant settings match how your POS is configured, and that your POS configuration matches how the merchant account is enabled. For example:

- If the POS uses E2E, then the MID must be enabled to allow E2E.
- If the POS is contactless capable, then the MID must be enabled for contactless point-of-sale entry mode.

Multiple Merchant ID (MMID) and **Multiple-Merchant Tokenization (MMT)** options are available, as discussed in the Planning section of this document. Contact your Implementations consultant for more information on these options.

2. <LanID>

Format: alphanumeric. Length: 3 max. No special characters, dashes, or spaces.

LanID is **required for multi-terminal merchants** for all card-present transactions sent to MercuryPay. Each terminal or input device must have a unique, alphanumeric identifier with a minimum of 1 and a maximum of 3 characters. Examples: 012, A3, or 4. Do not use a NULL value or 00.

3. <TranType> (Transaction Type) and <TranCode> (Transaction Codes)

<TranType> includes the tenders of Credit, Debit, EBT, FSA, Gift/Prepaid, Admin, Canadian Debit/EMV and CardLookup

<TranCode> refers to the transactions (such as Sale, Return, Balance, etc.) run under these specific tenders.

See [Appendix 16](#) for a complete list of supported Transaction Types and TranCodes.

4. <InvoiceNo> and <RefNo>

<InvoiceNo> is assigned by the local POS. It's an incrementing number used to identify the transaction to Mercury, and must be unique for each transaction that is processed. Invoice logic may differ from the POS numbering logic for orders, tickets, checks or receipt numbers, especially in the case of split tenders (e.g., when one check is being paid with two different credit cards). Each transaction that is processed with separate cardholder data, including each individual part of a split tender, requires a unique *InvoiceNo*.

Format: Length: 16 maximum, right 10 unique; Data Type: Numeric only

- Only the right-ten numbers of the *InvoiceNo* field are used for credit card processing with MercuryPay. The right-ten are sent with the transaction request to the card networks to check for duplicates. Therefore, these numbers must be unique per transaction. In addition, all-numeric zeroes, alpha characters, dashes or special characters may not be used.

<RefNo> is a unique number assigned by MercuryPay and returned in the transaction response. This number indicates that an approved transaction has been “referenced” and logged as an item in the merchant’s current batch file. For integration convenience, *RefNo* may be sent in the initial request as a filler value (since the actual value is not known until the response is received.)

- **Use the actual returned RefNo** whenever submitting a subsequent transaction that modifies the original transaction (e.g., for a Sale, this would be Reversal/VoidSale or Adjust). **Note that an approved PreAuth transaction, unlike an approved Sale, does not return an actual RefNo until it is captured.**

Note: For initial transaction requests such as *Sale*, *Return* or *PreAuth*, we recommend including *RefNo* as a “filler value”, which is typically the same value as the *InvoiceNo* (coded as *RefNo=InvoiceNo*) until the transaction response returns the actual *RefNo* value. The actual *RefNo* is then used in any subsequent “references to” when running *Adjusts/Reversals/VoidSales* of that original transaction.

5. <Memo> is now <POSPackageID>

Format: Length: 40 maximum; Data Type: Alpha-Numeric; Position: <Transaction> level.

Placed in all XML transaction requests, the *Memo* tag is used to identify the POS name and version. Dynamically build the *Memo* tag so that it pulls the product name and version from the project. This way the *Memo* stays current as the application version changes.

Note: For EMV, submit the <POSPackageID> in place of Memo.

Example: <POSPackageID >POS NAME v1.10A-08</POSPackageID>

6. <Frequency> and <RecordNo>

These two tags are used in MercuryPay's tokenization method (MToken). On the MercuryPay platform, tokens are referred to as **RecordNo**. A *RecordNo* is requested with two tags:

- the *Frequency* of the token requested (**OneTime** or **Recurring**), and
- either a request for token (**RecordNumberRequested**) or the actual token returned in the original response.

Format: Length: Token/RecordNos vary in length depending on card data and it is important to code for 48-100 alpha-numeric and special characters in the response, Data Type: Alpha-Numeric; Position: <Transaction> level.

A note about Frequency

- **OneTime** tokens are used for routine, **daily** routine transaction activity. The token is valid for six months, and is dynamically refreshed with each use, which extends the expiration window by another 6 months.
- **Recurring** tokens are only used for **weekly, monthly or annual** recurring billing programs, where the amount being charged is set and agreed upon with the cardholder. These charges appear as a recurring payment on cardholder statements. Recurring tokens are valid for 24 months, and as with *OneTime* tokens they refresh with each use, extending the expiration window by another 24 months.

All use of recurring tokens must be clearly stated in the merchant's terms and conditions, and signed by the cardholder indicating acceptance. The banking industry monitors their use.

- Recurring tokens should *not* be used for Card on File transactions where the cardholder is not charged a true recurring transaction.
- Additional details on Credentials of File and Card on File 2019 card brand regulations are highlighted in the Web Services and HostedCheckout sections of this guide.

Note Corporate Credit Cards may or may not accept recurring billing; this depends on the company's policies and on the issuing bank.

MercuryPay tokens require a secure token management system to be in place. Refer to additional details about using and managing tokens in this guide.

7. <PartialAuth>

Use of this data element was mandated by the Card Brands (Visa, MasterCard and Discover) initially for **Prepaid and Card Branded Debit** cards. Its required use is merchant category-specific and is *not* required for eCommerce, Kiosk or other Customer Attended Terminals (CAT). Use of this tag allows a partial approval if the balance on the card is less than the purchase amount requested. The transaction will approve for the available balance on the card and will not decline the transaction.

Two data elements are used in the transaction response to recognize and calculate the Partial Authorization:

- A *TextResponse* of **PARTIAL AP** (which can be passed back to the POS), and
- The <**Authorized Amount**> field, which will be less than the original purchase amount. This value can be used for "Balance Due" logic.

The PARTIAL AP response requires developers to build in split-tender functionality that recognizes the partial approval and prompts for a balance due. If the cardholder is unable to complete the balance due transaction, the developer is required to send a Reversal/VoidSale of the partially approved amount.

Note: The **PartialAuth=Allow** tag is used in all card present requests (*Sale* and *PreAuth*). It is also required for any card present integration in order to support *Reversals*. A card present transaction cannot be reversed without the use of *PartialAuth* in the originating transaction.

8. <Account>

This is the cardholder account data. It may be either swiped or manually entered, and is obtained from a clear text MSR, an encrypted card reader, or a supported peripheral device. The example above contains an encrypted, swiped credit sale.

- In this case, the cardholder data is encrypted and the *Account* data elements are sent using four encryption-specific tags: **<EncryptedFormat>**, **<AccountSource>**, **<EncryptedBlock>** and **<EncryptedKey>**, as shown below:

```
<Account>
  <EncryptedFormat>MagneSafe</EncryptedFormat>
  <AccountSource>Swiped</AccountSource>
  <EncryptedBlock>F40DDBA1F645CC8DB85A6459D45AFF8002C244A0F4402B479ABC991
5EC9567C81BE99CE4483AF3D</EncryptedBlock>
  <EncryptedKey>9012090B01C4F200002B</EncryptedKey>
</Account>
```

Note: when using Datacap's dsiEMVUS or dsiPDCX controls, this is handled by the client control.

- In a non-encrypted swiped request, Track1 or Track2 data may be used (but never both):

```
<Account>
  <Track2>Track2Data</Track2>
</Account>
```

- In a non-encrypted manual request, *AcctNo* and *ExpDate* are used:

```
<Account>
  <AcctNo>549999XXXXX6781</AcctNo>
  <ExpDate>0525</ExpDate>
</Account>
```

Note For manually-entered encryption transactions the *AccountSource* must be changed to **Keyed** and the accompanying *EncryptedBlock* is generated using a manual-entry format. If this is not done, the transaction will fail and display an error message.

9. <Amount>

This is the requested purchase amount. The tags in this element will be based on the transaction type and TranCode implemented.

- On a **Credit Sale**, only the **<Purchase>** tag is used:

```
<Amount>
  <Purchase>1.03</Purchase>
</Amount>
```

- On a **Debit Sale** an optional **<CashBack>** tag may also be used:

```
<Amount>
```



```

    <Purchase>5.00</Purchase>
    <CashBack>2.00</CashBack>
  </Amount>

```

- *Credit PreAuth* requires both `<Purchase>` and `<Authorize>` tags:

```

  <Amount>
    <Purchase>2.00</Purchase>
    <Authorize>2.00</Authorize>
  </Amount>

```

- A ***Credit PreAuthCapture*** requires the original Purchase and Authorize tags. An optional `<Gratuity>` tag may be added:

```

  <Amount>
    <Purchase>2.00</Purchase>
    <Authorize>2.00</Authorize>
    <Gratuity>0.50</Gratuity>
  </Amount>

```

- A ***Credit Adjust*** requires the original Purchase and Authorize tags. An optional `<Gratuity>` tag may be added:

```

  <Amount>
    <Purchase>1.23</Purchase>
    <Authorize>1.23</Authorize>
    <Gratuity>1.00</Gratuity>
  </Amount>

```

- When sending a ***ZeroAuth***, which is used to validate a card without authorizing it, only the `<Purchase>` tag is used with a value of **0.00**:

```

  <Amount>
    <Purchase>0.00</Purchase>
  </Amount>

```

10. Additional Tags

- Use **TerminalName**, **TerminalID** and **ShiftName** to send user-supplied data that will be shown on MercuryView. Web reports can be grouped by these identifiers.
- **OperatorID** is used to identify the POS user/clerk/operator. **This tag is a PCI requirement.** It is used to identify and manage users and logging controls on your POS. *OperatorID* is required for Certification and must be a configurable value.
- Manual Entry transactions must include **AcctNo**, **ExpDate**, **CVVData** and **AVS:Street**, **AVS:Zip**

Structure of the Transaction Response

All transactions routed through MercuryPay return a consistent set of response data. Use this data to confirm the authorization, build receipts and for creating internal POS reports.

The response information contains two sections. The **CmdResponse** is a quick snapshot of the authorization, and the **TranResponse** contains more detailed information.

- *CmdResponse* shows what source generated the response (*Processor*, *Server* or *Client*), a six-digit return code value, the status of the authorization (**Approved**, **Declined**, **Success**, **Error**) and verbiage in the <TextResponse> field.
- *TranResponse* provides the original data details sent in the request plus important Issuer authorization information. It also provides the processor transaction routing identifiers, and (if requested) a token for subsequent use.

Each tag is described below:

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    ①      <ResponseOrigin>Processor</ResponseOrigin>
    ②      <DSIXReturnCode>000000</DSIXReturnCode>
    ③      <CmdStatus>Approved</CmdStatus>
          <TextResponse>AP</TextResponse>
  </CmdResponse>
  <TranResponse>
    ①      <MerchantID>395347308</MerchantID>
    ②      <AcctNo>400300XXXXXX6781</AcctNo>  <!--masked account number and ExpDate-->
          <ExpDate>XXXX</ExpDate>
    ③      <CardType>VISA</CardType>
          <TranCode>Sale</TranCode>
          <AuthCode>000017</AuthCode>
    ④      <CaptureStatus>Captured</CaptureStatus>
    ⑤      <RefNo>0016</RefNo>
          <InvoiceNo>10</InvoiceNo>
          <OperatorID>MPS Operator</OperatorID>
          <Memo>MPS Example XML v1.0</Memo>
    ⑥      <Amount>
          <Purchase>1.05</Purchase>
          <Authorize>1.05</Authorize>
        </Amount>
    ⑦      <RecordNo>D7gnRteph5J9vvsh1rKhu5kQs+BS5ueAlhESEAAjEAB3nA==</RecordNo>  <!--Token-->
    ⑧      <AcqRefData>b8238953#20#20#20#20#20#20e00</AcqRefData><!--used later if need to reverse-->
          <ProcessData>|00|410100201000</ProcessData>
  </TranResponse>
</RStream>

```

<CmdResponse> tags

1. **<Response Origin>** indicates the source of the response:
 - **Client** indicates that the response is from the locally installed client control
 - **Server** indicates it is from the locally installed NETePay or MercuryPay's platform processing servers
 - **Processor** indicates that the response is from the back-end processing networks
2. **<DSIXReturnCode>** contains either a server-specific or DSIClientX-specific six digit code corresponding to a text message.
3. **<CmdStatus>** and **<TextResponse>**
 - **<CmdStatus>** indicates the outcome of the transaction:
 - For Credit, Debit, EBT, Check and Prepaid transactions: the value returned will be **Approved** or **Declined**.

- For Admin requests: the value returned will be **Success** or **Error**. Error text responses are listed later in this guide.

<TextResponse> dynamically-generated text that explains the <CmdStatus>.

Valid Approved Text Responses

Value	Description
AP	Credit response : transaction approved by processor
APPROVED	Debit and Prepaid response : transaction approved by processor
PARTIAL AP	Credit card branded Pre-paid Debit or Gift cards response: <ul style="list-style-type: none"> • The response is based on the use of <PartialAuth> ALLOW</PartialAuth> in the request; • it may require a split tender and a balance-due prompt; • always confirm <Authorize> amount approved. See PartialAuth_ for more information.
AP NEW INFO Or AP-NEW INFO	Credit response : optional message; account information is updated—this is often the address or expiration date
AP* (Reports as AP DUPE)	Indicates that duplicate logic is enabled and has detected the same card, the same amount, and the same invoice number in a second transaction in the same batch; it is paired with <CmdStatus>Approved</CmdStatus> and assures that the cardholder is charged only once. See Preventing Unintended Duplicates for more information.
AP – NOT CAPTURED	Credit, Debit, EBT and Check response : Transactions are being sent to the back-end processing networks but are not being captured or logged with the issuing bank. Typically this only occurs prior to a back-end network outage. It requires the transaction to be resent for authorization.
REVERSED	Credit reversal response : The transaction was acknowledged as qualified to be reversed by the card-issuing bank when sending a reversal (VoidSale + AcqRefData + ProcessData).

Declined Text Responses

Declined *TextResponse* values vary according to the source of the message and whether the decline was due to an issuer-level response or if there was an error in the processing of the transaction. Below is a condensed list:

Value	Description
DECLINE	Authorization was declined based on the Issuing bank's response. Card brand regulations stipulate that declines should not be re-attempted but should be followed-up with a request for a second form of payment.
CALL xxx Referrals	Issuer has placed a notice on the card and requests the merchant to call for a Voice Authorization. xxx = "Center," "AMEX," or "DISCOVER".
PIC UP	Authorization Declined; authorization network requests picking up the card and reporting. See card brand regulations for details and appropriate action.
DECLINED-CV2 FAIL	VISA response for CVV2 mismatch. (AMEX uses INVLD CID)
INVALID EXP DATE	Expiration date entered is incorrect, invalid or no longer current.
INVALID PIN	Incorrect PIN entered on Debit or EBT transaction.
UNAUTH USER	Merchant setting does not allow debit (may also be seen with EBT).
"NO TRAN FOUND"	Failed VoidSale or Reversal attempt.
"INV ITEM NUMBER"	Typically triggered by an invalid RefNo used in VoidSales or an Adjust.

Recommendation It is important that POS developers interpret and pass appropriate response messages, as well as corresponding alerts and prompts to the end-users of their system. Merchants should be able to see the *TextResponse* of decline messages such as **DECLINED CV2FAIL**, **Call ND**, **PIC UP**, etc., so they can take appropriate action if needed.

<TranResponse> tags

The **<TranResponse>** section contains a mirror of the original transaction request data, Issuer authorization data, and processor specific routing information.

1. **<MerchantID>** is returned without the "=" sign and "Nick Name".
2. **<AcctNo>**, **<ExpDate>** and **<AuthCode>** For PA-DSS validated applications not using MercuryPay's E2E or Tokenization, this data may be stored with strong encryption until batch close in order to support a possible *VoidSale*, *Adjust*, or *PreAuthCapture*.
 - **If using an E2E device or MToken direct to our processing servers:** **<AcctNo>** returns with the center numbers masked, displaying the first six digits and last four digits of the number. The masking convention (such as XXXX or ****) varies by local client, TranCode, and whether a local NETePay server is implemented and platform. **<ExpDate>** is returned fully masked. For PA-DSS validated applications choosing not to support encryption or tokenization, this data is returned in the clear and if required for subsequent use, must be immediately secured with strong encryption methods.
3. **<AuthCode>** This is the transaction authorization code supplied by the Issuer. Store this code for subsequent use with a *Reversal*, *VoidSale* or *PreAuthCapture*.
4. **<CardType>** indicates the card type (VISA, M/C, AMEX, DCLB, DCVR, JCB, DEBIT) and **<TranCode>** indicates the transaction code submitted (Sale, Return, VoidSale, VoidReturn, PreAuth, VoiceAuth, PreAuthCapture, Voucher, Issue, Balance, Adjust, ZeroAuth).
5. **<Capture Status>** will inform the POS if the transaction was **Captured** or **NotCaptured**.
6. **<RefNo>** and **<InvoiceNo>** The *RefNo* returned in the *TranResponse* is the transaction item reference number for an approved transaction and indicates that it has been added to the merchant's open batch. For subsequent *VoidSale* or *Adjusts*, this **<RefNo>** value must be used and NOT a filler value. The *InvoiceNo* is passed back directly from the original request.
7. **<Amount>** always returns both the original purchase amount from the request, and the actual authorized amount. Additional amount values vary based on the transaction and TranCode used, such as in *Gratuity*, *CashBack* and *Balance*. The examples below illustrate the different *Amount* field response tags:

- Standard response Amount data:

```
<Amount>
  <Purchase>1.05</Purchase>
  <Authorize>1.05</Authorize>
</Amount>
```

- PreAuthCapture amount data confirming Gratuity amount:

```
<Amount>
  <Purchase>2.02</Purchase>
  <Authorize>2.02</Authorize>
  <Gratuity>0.20</Gratuity>
</Amount>
```

- **Partial approval** amount response:

```
<Amount>
  <Purchase>23.54</Purchase>
  <Authorize>20.00</Authorize>
```

</Amount>

Note: For partially approved transactions, it is important to always compare the original purchase amount with actual authorization amount in order to submit a **balance due** prompt to the POS clerk.

- **Debit Response** including confirmation of **CashBack**:

```
<Amount>
  <Purchase>2.00</Purchase>
  <Authorize>2.50</Authorize>
  <CashBack>0.50</CashBack>
</Amount>
```

- **EBT response** including **Balance**:

```
<Amount>
  <Purchase>2.00</Purchase>
  <Authorize>2.00</Authorize>
  <Balance>15.38</Balance>
</Amount>
```

8. **<RecordNo>** is the token returned if one was requested. It may be stored “as is” for subsequent use with any *ByRecordNo* transactions.

- **Format:** Min/Max length: 48 to 100 alpha-numeric and special characters.

9. **<AcqRefData>** and **<ProcessData>** contain subfield strings of card brand specific identifiers, market data and transaction routing references.

- **Format - *AcqRefData*:** Min/Max length 1-200 alpha-numeric characters
- **Format - *ProcessData*:** Min/Max length 12-40 alpha-numeric characters

Use: *AcqRefData* must be submitted with all *PreAuthCapture* requests. *AcqRefData* and *ProcessData* are BOTH required to be submitted for reversal requests. When submitted, these fields are always included under the *TranInfo* tag.

Additional Response Data: CVVResult and AVSResult codes

These are Card Verification Value and Address Verification Service result codes. This data is returned independently from the card's authorization response and is critical for identifying fraudulent transactions using key-entered card-present and card-absent transactions. Per card brand best practices, the POS should supply merchants with both the result codes and options for cancelling the authorization if needed. This enables the merchant to make an informed decision on whether to accept the transaction or not. Immediate Reversals or *VoidSales* are recommended for all CVV mismatches. See [Fraud Protection using Card Security Codes and Address Verification](#) later in this guide for more information.

Appendix 4 – Industry, TranTypes and TranCodes

Overview

This section describes the individual transaction codes (such as *Sale*, *Return*, etc.) for each major type of transaction (e.g., Credit, Debit). The combination of transaction types (TranType) and transaction codes (TranCode) form the core of Datacap/MercuryPay's payload structure across all of our processing platforms.

The industry, or merchant category, that you are serving, along with card-brand mandates, determines which *TranTypes* and *TranCodes* your POS must support, along with other considerations:

Will you need to support gratuities (tips)?

Will transactions be only card-present, only card-not-present, or a combination of both?

What peripheral payment devices will you be using?

Retail (card present)

- POS developers support at a minimum: Credit *Sale*, *Partial Authorization*, *Return*, and *VoidSale/Reversal*.
- Additional Credit transactions to consider include *Adjust*, *VoidReturn*, *CardLookUp*, *ZeroAuth* and *VoiceAuth*. *VoidReturn* and *Adjust* transactions are helpful to merchants and can reduce their need for customer support. *ZeroAuth* enables merchants to run a credit card without a charge to validate the CVV and AVS information.
- Debit, EBT (Foodstamps and Cash) and FSA and may also be supported.
- In addition, because of the proven added value gift cards provide to retail environments, consider supporting *PrePaid Issue*, *Sale*, *Reload* and *VoidSale*.

Restaurant (card present)

POS developers support at minimum, either:

- *PreAuth* allowing for partial authorization, *PreAuthCapture* (including gratuity if required), *Return* and *VoidSale/Reversal*, or
- *Sale* allowing for partial authorization, *Adjust* (to support gratuities if required), *Return* and *VoidSale/Reversal*.

More details regarding *PreAuth/PreAuthCapture* and *Sale/Adjust* are provided later in this section.

- In addition, we recommend supporting *VoidReturn*, *Adjust* and *ZeroAuth*. An *Adjust* allows the merchant to adjust an incorrect transaction amount without the need to re-authorize the card. *ZeroAuth* enables merchants to verify the card without charging the cardholder. This is very useful in environments that support running tabs (e.g., bar tabs).
- Finally, because of the proven added value gift cards provide to retail environments, consider supporting *PrePaid Issue*, *Sale*, *Reload* and *VoidSale*.

MOTO and eCommerce (card not present)

MOTO (mail order/telephone order) and eCommerce are card-not-present environment where card information is entered manually (keyed). The best processing rates often rely on the use of industry recommended fraud protection validation services: Card Security Code (CVV) and Address Verification Service (AVS).

- *PreAuth* and *PreAuthCapture* are used whenever delivery/shipment is greater than 24 hours from the initial authorization.
- If shipment is under 24 hours from the initial authorization, use *Sale* and *Adjust*.
- eCommerce is not required to support Partial Approvals, but *Reversal/VoidSales* should still be supported in the event of CVV and AVS mismatch.

- We recommend supporting *ZeroAuth*, which enables the merchant to validate the CVV/AVS without running a charge. This helps avoid the need for *Reversal/VoidSales* in the case of a mismatch.

Credit Transactions and TranCodes

Sale. A *Sale* is a credit purchase, which authorizes and completes the transaction in one step. Upon approval, MercuryPay logs the *Sale* with a reference number (*RefNo*), and adds it to the merchant's open batch for settlement. Once captured, original Sales transactions may be adjusted, reversed or voided using the reference number and additional details from the response.

PreAuth and PreAuthCapture. The credit purchase occurs in two steps, authorization and capture. This is most frequently used in table service restaurants to modify a sale with a tip; it is also used by eCommerce merchants when their ship-date is more than 24 hours after the online order.

Return. A *Return* is the opposite of a *Sale*. It is an independent, non-referenced transaction used to credit a cardholder's account. **DO NOT USE a Return to void a transaction in an open batch.** *Returns* should be used if the original purchase was processed before the merchant's current open batch. *Return* transactions may be voided using the reference number and additional details from the response.

VoidSale and Reversal. *VoidSales* are used to void a transaction in the merchant's current open batch. When a *VoidSale* is accompanied with routing data (*AcqRefData* and *ProcessData*), it may also be called a "Reversal." The routing data is supplied in the original transaction response.

ZeroAuth. A *ZeroAuth* transaction used to verify that a credit card account number is valid and that the AVS and CVV data provided by the cardholder are correct, without actually charging an amount to the cardholder's account. The *ZeroAuth* transaction can also be used to request a MercuryPay token (*RecordNo*) that can be securely stored on the local POS and used for a future transaction or for recurring billing.

VoiceAuth. (voice authorization). A *VoiceAuth*, also referred to as a **Force**, is a credit purchase request sent through the POS after a merchant calls the card brand to obtain a voice authorization.

- A voice authorization is primarily used when MercuryPay returns a "Decline-Call referral" message with a *TextResponse* such as "Call ND," "Call AMEX" or "Call Discover." In these cases, the merchant contacts the automated voice-authorization system (using the number included in their MercuryPay Welcome kit) to obtain a six-digit authorization code (*AuthCode*).
- Receiving the *AuthCode* assures the merchant that the card is good and has credit available. The voice authorization reserves funds for the merchant.
- To obtain the funds, the merchant must manually enter the *AuthCode* into their POS to capture the transaction.

Supporting Gratuities (Tips)

Gratuities are processed using a two-step sequence:

1. A card is first authorized for a base amount (pre-tip), then
2. It is finalized with the addition of the tip amount.

This is accomplished by using either of these two pairs of transactions:

- **PreAuth**, always followed by a corresponding **PreAuthCapture**. This is standard in table-service restaurants where most transactions will include a tip. This method is also standard in eCommerce environments where the ship date is more than 24 hours after the order date.
- **Sale**, followed by the occasional **Adjust**. This is the preferred method for quick-serve restaurant environments where gratuities are less common.

While *PreAuth* is always dependent on a corresponding *PreAuthCapture* for completion and funding of the transaction, a *Sale* is not dependent on an *Adjust* for finalization.

PreAuth and PreAuthCapture

The *PreAuth/PreAuthCapture* sequence authorizes the initial purchase amount with the *PreAuth*, and then uses the *PreAuthCapture* to add the amount of the tip and finalize the transaction. The *PreAuth* only *holds* funds on the cardholder account; the *PreAuthCapture* is required for the merchant to get *funded*.

The *PreAuthCapture* transaction includes the following data to qualify for the best transaction processing rate:

- account number
- expiration date
- final purchase amount (or original purchase amount when supplying gratuity)
- original authorization amount
- gratuity/tip amount (if applicable)
- authorization code from the original *PreAuth* response
- Acquirer Reference Data (*AcqRefData*) from in the original *PreAuth* response

Important PCI PA-DSS Note: *PreAuthCapture* and *Adjust* requests may NEVER be submitted with full track data. Storage of full cardholder Track Data in any form is non-compliant with the Payment Card Industry Council's Payment Application Data Security Standards (PA-DSS). MToken technology eliminates the need to store sensitive card data.

PreAuth and PreAuthCapture XML Examples

The following illustrates the XML request and response structure for a swiped *PreAuth* and *PreAuthCapture*. This example uses an encrypted initial request and tokenization in the subsequent processing sequence.

Step 1: Swiped Encrypted PreAuth Request

```
<?xml version="1.0"?>
<TStream>
  <Transaction>
    <MerchantID>118725340908147</MerchantID>
    <LaneID>02</LaneID> <!--required for multi-terminal environments-->
    <TranType>Credit</TranType>
    <TranCode>PreAuth</TranCode>
    <InvoiceNo>16</InvoiceNo>
    <RefNo>16</RefNo> <!--use RefNo=InvoiceNo on PreAuth requests-->
    <Memo>MPS Example XML v1.0</Memo>
    <PartialAuth>Allow</PartialAuth> <!--Required to "Allow" partial approvals-->
    <Frequency>OneTime</Frequency> <!--use to request a Token-->
    <RecordNo>RecordNumberRequested</RecordNo>
    <Account> <!--use for encrypted data elements in place of Track1 or Track2-->
    <EncryptedFormat>MagneSafe</EncryptedFormat>
    <AccountSource>Swiped</AccountSource>
    <EncryptedBlock>F40DDBA1F645CC8DB85A6459D45AFF8002C244A0F74402B479
ABC9915EC9567C81BE99CE4483AF3D</EncryptedBlock> <!--Always use Track2 for E2E-->
    <EncryptedKey>9012090B01C4F200002B</EncryptedKey>
    </Account>
    <Amount>
      <Purchase>2.00</Purchase> <!--Purchase=Authorize amount from the request-->
      <Authorize>2.00</Authorize>
    </Amount>
  </Transaction>
</TStream>
```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>118725340908147</MerchantID>
    <AcctNo>400300XXXXX6781</AcctNo> <!--Returned truncated with encryption and token-->
    <ExpDate>XXXX</ExpDate> <!--Returned truncated (may be obtained from request for later use)-->
    <CardType>VISA</CardType>
    <TranCode>PreAuth</TranCode>
    <AuthCode>000067</AuthCode>
    <InvoiceNo>16</InvoiceNo>
    <OperatorID>MPS Operator</OperatorID>
    <Memo>MPS Example XML v1.0</Memo>
    <Amount>
      <Purchase>2.00</Purchase>
      <Authorize>2.00</Authorize>
    </Amount>
    <AcqRefData> b8238953#20#20#20#20#20#20#20e00</AcqRefData>
    <RecordNo>fbouwsq9TPgVRCZwImSsdmngVsaEwCXjIhESEAAjEAB3qQ==</RecordNo> <!--token data to be
used in PreAuthCapture request-->
    <ProcessData>|14|410100201000</ProcessData>
  </TranResponse>
</RStream>

```

Step 2: PreAuthCapture using the token from above (“ByRecordNo”)

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <MerchantID>118725340908147</MerchantID>
    <LaneID>02</LaneID> <!--required for multi-terminal environments-->
    <TranType>Credit</TranType>
    <TranCode>PreAuthCaptureByRecordNo</TranCode>
    <InvoiceNo>16</InvoiceNo>
    <RefNo>16</RefNo> <!--use RefNo=Invoice on request-->
    <Memo>MPS Example XML v1.0</Memo>
    <Frequency>OneTime</Frequency>
    <RecordNo>fbouwsq9TPgVRCZwImSsdmngVsaEwCXjIhESEAAjEAB3qQ==</RecordNo>
    <Amount>
      <Purchase>2.00</Purchase>
      <Authorize>2.00</Authorize> <!--the original purchase amount-->
      <Gratuity>0.50</Gratuity> <!--this is the added tip amount-->
    </Amount>
    <TranInfo>
      <AuthCode>000067</AuthCode> <!--use originally returned AuthCode-->
      <AcqRefData>b8238953#20#20#20#20#20#20#20e00</AcqRefData>
    </TranInfo>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>

```

```

    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>118725340908147</MerchantID>
    <AcctNo>400300XXXXXX6781</AcctNo>
    <ExpDate>XXXX</ExpDate>
    <Card Type>VISA</Card Type>
    <TranCode>PreAuthCapture</TranCode>
    <Auth Code>000067</Auth Code>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>0022</RefNo> <!--this is the actual reference number. Use this in subsequent transactions-->
    <InvoiceNo>16</InvoiceNo>
    <OperatorID>MPS Operator</OperatorID>
    <Memo>MPS Example XML v1.0</Memo>
    <Amount>
      <Purchase>2.00</Purchase>
      <Authorize>2.50</Authorize> <!--total will be returned from processor-->
      <Gratuity>0.50</Gratuity>
    </Amount>
    <AcqRefData>Kb8238953#20#20#20#20#20#20e00</AcqRefData>
    <RecordNo>js71hlxtTuEQKmBah3++tPQOxfA4J5bCIhESEAAjEAB3wQ==</RecordNo> <!--note "fresh" Token is returned-->
    <ProcessData>|14|410100201000</ProcessData>
  </TranResponse>
</RStream>

```

Credit Sale and corresponding Adjust

The *Sale* and *Adjust* transaction sequence may also be used to handle tips, and is most appropriate for quick serve restaurants/fast food where tips are less common. The advantage of using *Sale/Adjust* is that all Sales are finalized upon *Sale* completion. Using *Sale/Adjust*, a clerk only needs to “re-touch” (*Adjust*) the occasional transaction.

In addition to handling tips, *Adjust* may be used to change the *Amount* value for a transaction which was already captured in the current open batch. It requires the reference number (*RefNo*), *AuthCode*, the full card number and expiration date (or token) from the response data of the original sale.

- The adjusted purchase amount replaces the *Purchase* amount data in the original transaction, using the original authorization code.
- To eliminate the need to store card data for *Adjusts*, request a token (*RecordNo*). This *RecordNo* can then be used in place of card data to process all adjust transactions.

Note: *Adjust transactions cannot be reversed.* Instead, a VoidSale is used.

For tip adjustments, a Gratuify tag `<Gratuify>0.00</Gratuify>` can be added to an *Adjust*. The *Gratuify* tag allows the tip adjustment to be pulled as a reporting feature in the Merchant's MercuryView™ and assists with reconciliation. As with *PreAuth/PreAuthCapture*, the processor calculates the final total amount in the response. Therefore, the *Authorize* amount should equal the original Authorized amount. As an example:

Request

```
<Amount>
  <Purchase>1.23</Purchase>
  <Authorize>1.23</Authorize>
  <Gratuity>1.00</Gratuity>
</Amount>
```

Response

```
<Amount>
<Purchase>1.23</Purchase>
<Authorize>2.23</Authorize> <!--Processor totals the Authorize amount-->
```

```
<Gratuity>1.00</Gratuity>
</Amount>
```

Void or Reverse a Transaction

Historically there have been two ways to cancel a transaction in the merchant's open batch:

- A **VoidSale** communicates with the **Processor** and the original transaction may take several business days before it is removed from the cardholder's account.
- A **Reversal**, which is a VoidSale that includes additional routing data, communicates directly with the **Issuing Bank**, and may take less time (from 2 to 72 hours) to be removed from the cardholder's account.

Reversals are the preferred way to cancel a transaction.

VoidSale - Reversal

A **VoidSale** is the *TranCode* used when sending a reversal and is used to cancel an existing *Credit Sale*, *Adjust*, *VoiceAuth*, *PreAuth*, or *PreAuthCapture* transaction that is logged with a reference number in the merchant's current open batch. (For *Return* transactions a **VoidReturn** is used.)

VoidSales require the *RefNo* and *AuthCode* from the original transaction, as well as *InvoiceNo*, the response token or full card number, expiration date and authorized purchase amount.

A **Reversal** is not a Transaction Type or a *TranCode*. It refers to the transaction routing practice of **communicating directly with the card Issuing bank** in order to release held funds back into the cardholder's account. Reversals use the same data as a *VoidSale* with the addition of two data elements added to the *TranInfo*:

```
<TranInfo>
  <AcqRefData>b8238953#20#20#20#20#20#20#20e00</AcqRefData>
  <ProcessData>14|410100201000</ProcessData>
  <AuthCode>000067</AuthCode>
</TranInfo>
```

Reversals may be used to reverse an existing *Credit Sale*, *VoiceAuth*, *PreAuth*, *PreAuthCapture*, *PreAuth* and *FSASale* (Flexible Spending Account) transactions. They are mandated for use with *Partial Approval* for card-branded, prepaid Credit and Debit cards.

Note: MercuryPay/Datacap does NOT support partial reversals.

Follow the steps below to process a *Reversal* request:

1. The original transactions must "allow" the reversal. This is done by including the **PartialAuth=Allow** tag. *Transactions processed without the PartialAuth-Allow tag cannot be reversed.* (For eCommerce developers, this "allow" feature is handled by the MercuryPay servers with no additional tag requirements.)
2. The request must include all the required *TranCode* data elements of a standard *VoidSale* (*AcctNo*, *Exp Date*, *InvoiceNo*, *RefNo*, *AuthCode*, *Purchase* and *Memo*).
3. **Two additional data elements** from the original transaction response are added at the **TranInfo** level: **AcqRefData** and **ProcessData**. This sends the request to the Issuing bank.
4. Finally, confirm that a *CmdStatus=Approved* and *TextResponse=REVERSED* message is returned; if it is not returned a standard *VoidSale* should be immediately sent without the *AcqRefData* and *ProcessData* elements.

Reversal Support for PreAuth

The *PreAuth* reversal allows an authorized yet uncaptured *PreAuth* to be reversed at the Issuing bank, canceling the initial *PreAuth* request and releasing the held "open to buy" funds to the cardholder's account. The cardholder is never charged and the "pending" timeframe that appears on the cardholder's statement is significantly reduced. Using this method, an uncaptured *PreAuth* will not be subject to the Card Brand's "Misuse

of Authorization Network” fee. *PreAuth Reversals* have many uses, particularly within hospitality verticals as they may assist in handling bar tabs.

PreAuth Reversal XML Data Elements

Although an actual host *RefNo* is not generated on PreAuths, the *RefNo* field is required in *PreAuth Reversals* and must be present in the reversal request as a filler value. For ease of coding and XML consistency, use **RefNo=InvoiceNo** for the *RefNo* value on all *PreAuth Reversal* requests.

PreAuth Reversal XML Example

Swiped Encrypted PreAuth Request

```
<?xml version="1.0"?>
<TStream>
  <Transaction>
    <MerchantID>118725340908147</MerchantID>
    <LanelID>02</LanelID>
    <TranType>Credit</TranType>
    <TranCode>PreAuth</TranCode>
    <InvoiceNo>16</InvoiceNo>
    <RefNo>16</RefNo> <!--Filler value RefNo=InvoiceNo -->
    <Memo>MPS Example</Memo>
    <PartialAuth>Allow</PartialAuth> <!--PartialAuth required for Reversal Support -->
    <Frequency>OneTime</Frequency>
    <RecordNo>RecordNumberRequested</RecordNo>
    <Account>
      <EncryptedFormat>MagneSafe</EncryptedFormat>
      <AccountSource>Swiped</AccountSource>
      <EncryptedBlock>F40DDBA1F645CC8DB85A6459D45AFF8002C244A0F74402B479
ABC9915EC9567C81BE99CE4483AF3D</EncryptedBlock>
      <EncryptedKey>9012090B01C4F200002B</EncryptedKey>
    </Account>
    <Amount>
      <Purchase>2.00</Purchase> <!--PreAuth always uses Authorize and Purchase amounts-->
      <Authorize>2.00</Authorize>
    </Amount>
  </Transaction>
</TStream>
```

Response

```
<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>118725340908147</MerchantID>
    <AcctNo>400300XXXXXX6781</AcctNo>
    <ExpDate>XXXX</ExpDate>
    <CardType>VISA</CardType>
    <TranCode>PreAuth</TranCode>
    <AuthCode>000067</AuthCode>
    <InvoiceNo>16</InvoiceNo>
    <Memo>MPS Example</Memo>
    <Amount>
      <Purchase>2.00</Purchase>
```



```

        <Authorize>2.00</Authorize>
    </Amount>
    <RecordNo>fbouwsq9TPgVRCZwlmSsdmngVsaEwCXjIhESEAAjEAB3qQ==</RecordNo>
    <AcqRefData>b8238953#20#20#20#20#20#20e00</AcqRefData><!--AcqRefData and ProcessData used for
reversal-->
    <ProcessData>|14|410100201000</ProcessData>
</TranResponse>
</RStream>

```

PreAuth Reversal Request (VoidSale + AcqRef + ProcessData)

Request

```

<?xml version="1.0"?>
<TStream>
    <Transaction>
        <MerchantID>118725340908147</MerchantID>
        <LaneID>02</LaneID>
        <TranType>Credit</TranType>
        <TranCode>VoidSaleByRecordNo</TranCode><!--Use VoidSaleByRecordNo when sending Token-->
        <InvoiceNo>16</InvoiceNo>
        <RefNo>16</RefNo> <!--Required Reversal filler value RefNo=InvoiceNo-->
        <Memo>MPS Example</Memo>
        <RecordNo>fbouwsq9TPgVRCZwlmSsdmngVsaEwCXjIhESEAAjEAB3qQ==</RecordNo>
        <Frequency>OneTime</Frequency>
        <Amount>
            <Purchase>3.00</Purchase>
        </Amount>
        <TranInfo> <!--required data to trigger Reversal-->
            <AcqRefData>b8238953#20#20#20#20#20#20e00</AcqRefData><!--AcqRefData code to
maximum length = 200-->
            <ProcessData>|14|410100201000</ProcessData> <!--Processing Routing data-->
            <Auth Code>000067</Auth Code>
        </TranInfo>
    </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
    <CmdResponse>
        <ResponseOrigin>Processor</ResponseOrigin>
        <DSIXReturnCode>000000</DSIXReturnCode>
        <CmdStatus>Approved</CmdStatus> <!--reversal was successful-->
        <TextResponse>REVERSED</TextResponse>
        <UserTraceData></UserTraceData>
    </CmdResponse>
    <TranResponse>
        <MerchantID>118725340908147</MerchantID>
        <AcctNo>4003xxxxxx56781</AcctNo>
        <ExpDate>XXXX</ExpDate>
        <CardType>VISA</CardType>
        <TranCode>VoidSalebyRecordNo</TranCode>
        <Auth Code>000067</Auth Code>
        <CaptureStatus>Captured</CaptureStatus>
        <RefNo>16</RefNo>
        <InvoiceNo>16</InvoiceNo>
        <Memo>MPS Example</Memo>
        <Amount>
            <Purchase>3.00</Purchase>
            <Authorize>3.00</Authorize>
        </Amount>
    </TranResponse>
</RStream>

```

```

        </Amount>
        <AcqRefData>Kb8238953#20#20#20#20#20#20#20e00</AcqRefData>
        <RecordNo>js71hlxtTuEQKmBah3++tPQOxfA4J5bClhESEAAjEAB3wQ==</RecordNo>
    </TranResponse>
</RStream>

```

Recommendations

1. Use the `<PartialAuth>Allow</PartialAuth>` tag on the original request.
2. Include the `AcqRefData` and `ProcessData` on `VoidSale` requests so that the transaction is communicated to the Issuing bank.
3. Code to only allow for one Reversal per transaction.
4. Use `InvoiceNo` as filler data for `RefNo`.
5. Check `CmdStatus` and `TextResponse` for Approved/REVERSED.

Note: It is important to build in provisions in your system **to allow only one reversal request attempt** to be sent per transaction request. Multiple reversal requests are not recommended. **If the reversal request returns anything other than a CmdStatus of "Approved" and a TextResponse of "REVERSED"**, do NOT attempt a reversal retry. Submit a `VoidSale` instead.

ZeroAuth (Zero Authorization)

The **ZeroAuth** transaction enables merchants to validate a credit card without charging an amount to the cardholder's account. It can also be used to request a token if you are using MercuryPay's MToken solution in your POS. Example:

Request

```

?xml version="1.0"?>
<TStream>
    <Transaction>
        <MerchantID>118725340908147</MerchantID>
        <LaneID>02</LaneID>  <!--for environments with multiple input devices-->
        <TranType>Credit</TranType>
        <TranCode>ZeroAuth</TranCode>
        <InvoiceNo>12345</InvoiceNo>
        <Frequency>OneTime</Frequency><!--Token Request-->
        <RecordNo>RecordNumberRequested</RecordNo>
        <Account><!--for swiped use Track2-->
            <AcctNo>Card Data</AccountNo>
            <ExpDate>12/20</ExpDate>
        </Account>
        <CVVData>123</CVVData><!--CVV and AVS for fraud protection. CVV is not sent with a swiped
entry-->
        <AVS>
            <Address>4 Corporate Square</Address>
            <Zip>30329</Zip>
        </AVS>
        <Memo>MPS Example V1.0</Memo>
    </Transaction>
    <Amount>
        <Purchase>0.00</Purchase>
    </Amount>
</TStream>

```

ZeroAuth Response

```

<?xml version="1.0"?>

```

U.S. PIN Debit

Sale. A Debit *Sale* is a real-time debiting of the cardholder's account.

- Returns cannot be voided or reversed.
- Returns do not allow *CashBack*.

- Because of the real-time nature of debit processing, there is no PIN-based equivalent for *VoidSale*, *Adjust*, *PreAuth*, or *PreAuthCapture* transactions. Amounts will either be withdrawn from the cardholder's account with a *Debit Sale* or be added to the account by *Debit Return*.
- All PIN-based debit transactions must be processed using Track2.
- All PIN-based debit transactions may only be swiped.

U.S. PIN Debit XML Examples

The following example is an encrypted Debit *Sale* with cash back using an E2E encrypted enabled device on a *MerchantID* that is both E2E and token enabled.

- Although debit transactions may never be sent *ByRecordNo* (that is, using a stored token), this example includes the *Frequency* and *RecordNo* tags. The use of these tags is optional and ignored when sending a debit transaction to a token-enabled Merchant ID.

Debit Sale

Request

```
<?xml version="1.0"?>
<TStream>
  <Transaction>
    <MerchantID>118725340908147</MerchantID>
    <LaneID>02</LaneID> <!--required for multi-terminal environments-->
    <TranType>Debit</TranType>
    <TranCode>Sale</TranCode>
    <InvoiceNo>30</InvoiceNo>
    <RefNo>30</RefNo>
    <Memo>MPS Example XML v1.0</Memo>
    <Frequency>OneTime</Frequency>
    <RecordNo>RecordNumberRequested</RecordNo>
    <Account>
      <EncryptedFormat>MagneSafe</EncryptedFormat>
      <AccountSource>Swiped</AccountSource>
      <EncryptedBlock>005234EAD7AD5A279B8FB947BC15A123FFD775D03386B54F6B37
CB54AB33106FFC105588E28B0799</EncryptedBlock> <!--handled by Datacap's client controls when using dsiEMVUS or
dsiPDCX-->
      <EncryptedKey>9500030000040C20001D</EncryptedKey>
    </Account>
    <Amount>
      <Purchase>2.00</Purchase>
      <CashBack>0.50</CashBack>
    </Amount>
    <PIN>
      <PINBlock>0A04E4C40F6A41B2</PINBlock> <!--pulled and parsed from the PINpad -->
      <DervdKey>3D0100000B200007</DervdKey> <!--pulled and parsed from the PINpad -->
    </PIN>
  </Transaction>
</TStream>
```

Debit Sale Response

```
<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>APPROVED</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>118725340908147</MerchantID>
    <AcctNo>400300XXXXXX6781</AcctNo>
    <CardType>DEBIT</CardType>
    <TranCode>Sale</TranCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>0032</RefNo>
    <InvoiceNo>30</InvoiceNo>
```

```

        <OperatorID>MPS Operator</OperatorID>
        <Memo>MPS Example XML v1.0</Memo>
        <Amount>
            <Purchase>2.00</Purchase>
            <Authorize>2.50</Authorize>
            <CashBack>0.50</CashBack>
        </Amount>
    </TranResponse>
</RStream>

```

CardLookup

CardLookup is used by the POS to determine if the card is a Debit or FSA card. The *CardLookup* request uses a simplified format of **TranType=CardLookup**, with no *TranCode*.

Request

```

<?xml version="1.0"?>
<TStream>
    <Transaction>
        <MerchantID>112438931977591</MerchantID>
        <TranType>CardLookup</TranType> <!--Note there is no TranCode used in this request-->
        <Memo>JBsPOS 7.5.0.2</Memo>
        <Account>
            <Track2>Track2Data</Track2>
        </Account> <!--may also be sent with encryption elements>
    </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
    <CmdResponse>
        <ResponseOrigin>Server</ResponseOrigin>
        <DSIXReturnCode>000000</DSIXReturnCode>
        <CmdStatus>Approved</CmdStatus>
        <TextResponse>SUCCESSFUL LOOKUP</TextResponse>
        <UserTraceData></UserTraceData>
    </CmdResponse>
    <TranResponse>
        <MerchantID>112438931977591</MerchantID>
        <AcctNo>XXXXXXXXXX6781</AcctNo> <!--truncated if paired with E2E or tokenization-->
        <ExpDate>0513</ExpDate>
        <CardUsage>DEBIT</CardUsage> <!--Value can be FSA, DEBIT or OTHER-->
        <CardType>VISA</CardType>
        <TranType>CardLookup</TranType>
        <TranCode>CardLookup</TranCode>
    </TranResponse>
</RStream>

```

Response when requesting a Token (RecordNo)

```

<?xml version="1.0"?>
<RStream>
    <CmdResponse>
        <ResponseOrigin>Server</ResponseOrigin>
        <DSIXReturnCode>000000</DSIXReturnCode>
        <CmdStatus>Approved</CmdStatus>
        <TextResponse>SUCCESSFUL LOOKUP</TextResponse>
        <UserTraceData></UserTraceData>
    </CmdResponse>

```

```

</CmdResponse>
<TranResponse>
  <MerchantID>395347306</MerchantID>
  <AcctNo>400300XXXXXX6781</AcctNo>
  <ExpDate>XXXX</ExpDate>
  <CardUsage>OTHER</CardUsage>
  <CardType>VISA</CardType>
  <TranType>CardLookup</TranType>
  <TranCode>CardLookup</TranCode>
  <RecordNo>fLhQcVloIp6pq4X7emYELLPrGVr5t6o1lhESEAAjEA SDKw==</RecordNo>
</TranResponse>
</RStream>

```

Notes

- *CardLookup* does not authenticate the card number! It only indicates whether the card passed a valid MOD10 check (Luhn), and its BIN/IIN usage response is either *FSA*, *DEBIT* or *OTHER*.
- A token may be requested, but tokens returned from a *CardLookup* are processed as *MANUAL* transactions.
- AVS and CVV result data is unsupported in a *CardLookup* request, and must be supplied separately in subsequent transactions. If AVS and CVV results are needed, *ZeroAuth* is a good option. Note that neither *CardLookup* nor *ZeroAuth* can be processed "*ByRecordNo*".

Appendix 5 – Card Branded Prepaid Debit Cards

Overview

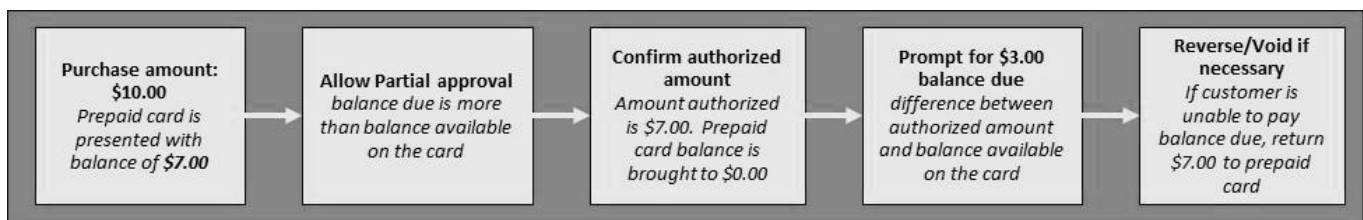
The card brands (Visa, MasterCard, Discover and American Express) have developed StoredValue PrePaid card products branded with their company logos. These “Prepaid Credit and Debit Cards” as they are known, are processed like credit cards and do not require PIN entry, but they function more like stored-value gift cards. Only the available balance may be used. These cards carry with them a unique set of mandated processing features that developers must support:

- The cards must be usable down to a zero balance even if the purchase amount is higher. As long as there is a remaining balance on the card, POS systems must accept partial approvals up to that balance, and prompt for a balance-due for the remaining purchase amount.
- If the cardholder cannot provide a second form of payment, the POS must be able to return the value to the prepaid card. This is accomplished by sending a **reversal** of the partially approved amount. If the reversal fails, a standard *VoidSale* should be sent.

<PartialAuth>

These features are referred to as **partial authorization** and issuer-level reversals, and are required for card-present merchants. The card brands mandated these features to ensure their prepaid cards will always be partially approved if needed, and always reversed when necessary.

The process flow for a partial authorization is illustrated below. The example is for a \$10.00 purchase using a prepaid card with a \$7.00 balance.



Partial Authorization Transactions

The XML tag **<PartialAuth>Allow</PartialAuth>** is required to be included in all *Credit Sale* and *PreAuth* transactions for all card-present merchant categories except for eCommerce, unattended Kiosk, and Customer Attended Terminal (CAT) merchant environments (including car washes).

- **PartialAuth=Allow** enables the issuer to pass back an authorized amount even if that amount is less than the purchase amount, and ensures that a reversal can be sent if necessary. By allowing partial approvals, prepaid cards will only decline for an insufficient balance when the balance is zero.
- **If the available balance is less than the purchase amount**, then the available balance amount will be used as the authorized amount to partially approve the transaction.
- If a partial authorization is returned, the POS is notified two ways:
 - The *TextResponse* message returns “**PARTIAL AP**”
 - The *Authorize* amount will be less than the purchase amount.
- An authorized amount that is less than the original Purchase amount requires a split tender. The POS will need to subtract the *Authorize* amount from the original *Purchase* amount and prompt for a “balance due” with a second form of payment.
 - If the cardholder cannot pay balance due amount, a *Reversal* must be sent. For *Reversal* support, developers will need to code to store both *AcqRefData* and *ProcessData* from the original

transaction response and send these in the *TranInfo* section of a *VoidSale*. See the section on *VoidSale* and *Reversal* for details.

- Running partial reversals and running reversals on transactions from prior batches is not supported. Developers should build in code to prevent either of these from accidentally occurring.

Note: An adjustment or gratuity should *never* be added for any transaction that receives a *PARTIAL AP* response since this indicates that the card balance is already at 0.00.

Partial Authorization XML Example

The example below uses an encrypted request with a "trigger" amount of 23.55 used to simulate a partial approval. MercuryPay provides a number of test cards to help with the development of different processing features. In this example, the test card is using an amount that "triggers" the *PARTIAL AP* text response and a lower authorized amount.

Request

```
<?xml version="1.0"?>
<TStream>
  <Transaction>
    <MerchantID>118725340908147</MerchantID>
    <LaneID>02</LaneID>
    <TranType>Credit</TranType>
    <TranCode>Sale</TranCode>
    <InvoiceNo>21</InvoiceNo>
    <RefNo>21</RefNo>
    <Memo>MPS Example XML v1.0</Memo>
    <Frequency>OneTime</Frequency>
    <RecordNo>RecordNumberRequested</RecordNo>
    <PartialAuth>Allow</PartialAuth> <!--Required "Allow" tag-->
    <Account>
      <!--Encrypted data shown here in place of <Track1> or <Track2>-->
      <EncryptedFormat>MagneSafe</EncryptedFormat>
      <AccountSource>Swiped</AccountSource>
      <EncryptedBlock>2F8248964608156B2B1745287B44CA90A349905F905514ABE3979D7957F1380470
5684B1C9D5641C</EncryptedBlock> <!--Use Track2 only-->
      <EncryptedKey>9500030000040C200026</EncryptedKey>
    </Account>
    <Amount>
      <Purchase>23.55</Purchase> <!--"Trigger" Amount used with this card-->
    </Amount>
  </Transaction>
</TStream>
```

Response

Note in the response below, the transaction approves with a *PARTIAL AP* (partial approval) and is authorized for \$20.00 (less than the original purchase amount).

```
<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>PARTIAL AP</TextResponse> <!--Note TextResponse-->
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>118725340908147</MerchantID>
    <AcctNo>400555XXXXXX0480</AcctNo>
    <ExpDate>XXXX</ExpDate>
    <CardType>VISA</CardType>
```

```

    <TranCode>Sale</TranCode>
    <AuthCode>VI2000</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>0026</RefNo>
    <InvoiceNo>21</InvoiceNo>
    <OperatorID>MPS Operator</OperatorID>
    <Memo>MPS Example XML v1.0</Memo>
    <Amount>
        <Purchase>23.55</Purchase>
        <Authorize>20.00</Authorize>    <!--Compare amounts-->
    </Amount>
    <!--data below to be saved for potential reversal if necessary and used in <TranInfo>-->
    <AcqRefData>b8238953#20#20#20#20#20#20#20#20e00</AcqRefData>
    <RecordNo>ewNvAAvYCYIFRbS06i1ssdjV+8YOv2nIhESEAAjEAB3yg==</RecordNo>    <!--Token-->
    <ProcessData>|00|410100201000</ProcessData>
</TranResponse>
</RStream>

```

Reversal XML Example

If the cardholder does not have the balance, a reversal may be processed. This VoidSale/Reversal example is for a non-encrypted Sale. If the cardholder is unable to pay the balance, the amount charged to the prepaid card must be refunded.

Request

```

<?xml version="1.0"?>
<TStream>
    <Transaction>
        <MerchantID>112438931977591</MerchantID>
        <LaneID>02</LaneID>
        <TranType>Credit</TranType>
        <TranCode>VoidSale</TranCode><!--Reversal also uses VoidSale-->
        <RefNo>0021</RefNo>    <!-- use actual RefNo returned from Sale; not filler value-->
        <InvoiceNo>111</InvoiceNo>
        <Memo>MPSTest</Memo>
        <Account>
            <AcctNo>Card Data</AcctNo>    <!--AcctNo and Exp Date only or Token-->
            <ExpDate>1220</ExpDate>
        </Account>
        <Amount>
            <Purchase>20.00</Purchase>    <!--Purchase amount is Authorized amount from Sale-->
        </Amount>
        <TranInfo>    <!--Includes AcqRefData, ProcessData and AuthCode from Sale-->
        <AcqRefData>Kb8238953#20#20#20#20#20#20#20#20e00</AcqRefData>
        <ProcessData>|00|410100701000</ProcessData>
        <AuthCode>DI2000</AuthCode>
        </TranInfo>
    </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
    <CmdResponse>
        <ResponseOrigin>Processor</ResponseOrigin>
        <DSIXReturnCode>000000</DSIXReturnCode>
        <CmdStatus>Approved</CmdStatus>
    </CmdResponse>
</RStream>

```

```
<TextResponse>REVERSED</TextResponse><!--note Text Response is REVERSED-->
<UserTraceData></UserTraceData>
</CmdResponse>
<TranResponse>
  <MerchantID>112438931977591</MerchantID>
  <AcctNo>601190XXXXXX5677</AcctNo>
  <ExpDate>1215</ExpDate>
  <CardType>DCVR</CardType>
  <TranCode>Void Sale</TranCode>
  <CaptureStatus>Captured</CaptureStatus>
  <InvoiceNo>111</InvoiceNo>

  <Memo>MPSTest</Memo>
  <Amount>
    <Purchase>20.00</Purchase>
    <Authorize>20.00</Authorize>
  </Amount>
  <AcqRefData>K</AcqRefData>
</TranResponse>
</RStream>
```

Appendix 6 – Merchant Settlement

Batch Closing Batch Capture

In addition to managing payment modules, many POS systems also manage the end-of-day batch closing and settlement procedures. MercuryPay offers a variety of capture and close methods to meet the needs of today's complex payment systems.

Merchant Categories and Settlement Options

The first thing to consider is your targeted merchant category, which helps determine who initiates the settlement, how transactions are captured, and how and when the settlement takes place:

Merchant Category	End-of-Day Settlement and Funding Considerations
1. Standard Retail	<p>Time Initiated refers to daily auto-settlement at a specified time.</p> <ul style="list-style-type: none"> Daily scheduled settlement time defaults to 4:00 AM EST for U.S. merchants and provides two-business day funding. Speak with a merchant account representative for more information on alternative batching times. <p>You should consider your business's time zone in relationship to the settlement times.</p>
2. Table service restaurant with tip modification	Merchant Initiated. End of day settlement is typically initiated locally after the final tip adjustments have been added to the batch.
3. Quick serve restaurant (QSR/fast food)	Tip adjustments are rare and settlement varies, but this environment is typically Time Initiated .
4. Mail order/telephone order (MOTO)	
5. eCommerce/Hosted Checkout	End of day is typically Time Initiated .
6. Unattended Kiosks (also referred to as Customer Attended Terminals (CAT))	

Finalizing and Capturing Transactions

As authorizations are processed to MercuryPay they are captured (logged) in the merchant's open batch file. This batch file only contains completed authorizations, meaning everything except *PreAuths*. *PreAuths* are added after the *PreAuthCapture* is sent to complete the transactions. Authorizations can be captured into an open batch in one of two ways:

- **Stream Capture.** The POS system sends authorization requests and captures throughout the business day.
- **Batch Capture.** The POS system sends authorization requests but the merchant modifies the transactions in their local system before uploading the batch as part of their end-of-day procedures.

In standard retail environments, where a time-initiated batch close is the rule, transactions are "streamed" to MercuryPay and a single "auto" capture method is used to settle the open batch.

In environments requiring tip or other transaction modifications, where a merchant-initiated batch close is more common, capture method is determined by the POS architecture:

- **For Stream Capture**, the POS sends the finalized *PreAuthCapture* transaction as soon as the tip amount is entered into the POS system to close the check. In this approach, any subsequent deletions or corrections require sending a *Reversal/VoidSale* or an *Adjust* before the final *BatchClose* command is sent.
- **For Batch Capture**, the POS sends the finalized transactions as a part of the merchant's end-of-day closing procedure. During the business day, a *PreAuth* is used to initially authorize the transaction. The POS then holds this information in its own, local settlement file. When the merchant performs the end-of-

day procedures, the tip amounts are entered and the finalized set of transactions are checked for accuracy. Any corrections needed are handled at this time, before the final *BatchClose*.

- End of Day Sequence in Batch Capture Environments:
 1. POS system confirms that the local batch file is accurate.
 2. POS sends a *BatchClear* command to remove any unrequired captures.
 3. POS sends the capture transactions, one by one to MercuryPay.
 4. POS sends a *BatchSummary* to verify that item counts and amount values match.
 5. POS sends the *BatchClose*.
 6. In the event there is an upload failure or problem with the *BatchClose*:
 - Send a *BatchClear* to void the *PreAuthCaptures* that were sent on the first attempt.

Note Do not clear the batch locally or when debits are present.

- Start the *BatchClose* process from the beginning.

Note The Batch capture method requires the use of a **BatchClear**. **Caution should always be used** when using *BatchClear* to prevent the loss of the captured batch data. Always confirm that the batch file has been captured completely by issuing a *BatchSummary* and then issuing the final *BatchClose*.

Batch Settlement Recommendations

- All POS systems using the “Batch” close method **must** be set up with merchant-initiated close.
- For end-of-day batch close with a composite batch including PIN-based debit transactions and Returns, proceed with the steps outlined above, but **do not** send a *BatchClear* request for either step one or step six.
- Systems incorporating composite batches must support an Invoice field. MercuryPay’s enhanced duplicate checking will catch any potential duplicates in the event it is necessary to resend a batch fail.

“Host Based” Time or Merchant Initiated Settlement

MercuryPay/Datacap is predominately a “host-based” processor. Worldpay from FIS authorizing platform authorizes transactions throughout the day and our back end settlement platform maintains a merchant’s batch file.

- The file can be sent at a given time to close the batch, or allow a merchant to initiate the batch close at a time they choose. **Time Initiated** means that Worldpay initiates the close. **Merchant Initiated** is when the merchant initiates the close.
- Although we have provisions in place to accommodate “terminal-based” batch methods, the advantage of a host-based environment is that merchants can never lose a batch due to a hardware failure or a localized mistake. Completed transactions are always logged at MercuryPay until settlement.

Time-Initiated Batch Close

This may also be referred to as “auto-settle.”

- **Recommended for:** all general retail, quick serve/fast food, and eCommerce. These verticals do not typically process tips or need to modify the original authorization amount.

Time-initiated batch close is an auto-settle function whereby all captured transactions are settled in one batch. Defaulted close time is 4:00 AM Eastern Standard Time (EST). It provides standard two-day funding. Other close times are available.

- Time-initiated avoids the common issue of a merchant forgetting to close their batch, or having problems matching up their totals and number of transactions.
- The developer does not need to write code to support a *BatchClose* transaction.

- The maximum size for a time-initiated batch is approximately 4500 transactions. If this number is ever reached, processing continues seamlessly as the merchant batch will increment to the next batch number.

Merchant-Initiated Batch Close

Merchant-initiated means there is a local, merchant-facing POS *BatchClose* command from the POS which initiates the batch close process.

- Recommended for:** table service restaurants, nightclubs and bars where tip adjustments commonly occur, and for merchants who need local control over the batch closing time.
- Merchants have local control of settlement times, and therefore deposits.
- Merchants who are open late or are located on the West Coast, Hawaii or Alaska, can process their tip adjustments and make sure that all their totals are correct before sending a *BatchClose*.

Additional Merchant Initiated Considerations

- Merchants must close their batch within 24 hours of the original authorization or they will face downgrades (higher transaction rates). This is because authorization codes increasingly lose their interchange authenticity over time and inherently carry higher risk. If a batch remains open too long, there is built-in protection to stop accepting additional capture transactions and return the message, "MUST BALANCE NOW" or "USE DUP THEN BAL".
- The maximum size for a merchant-initiated batch is approximately 4900 transactions. It is important to avoid hitting this threshold. If it is reached, the merchant will receive a MUST BALANCE NOW message, and will have no recourse for adding gratuities or adjusting any transactions. Processing will be temporarily interrupted until the batch is successfully closed. Businesses that reach this threshold often have multiple locations routing transactions to a single Merchant ID. An easy way to avoid reaching the threshold and risk being unable to process is simply to implement different Merchant ID's at each location.

Batch Summary/Close Technical Requirements

The merchant-initiated batch close uses a two-step set of **Admin** transactions requests: **BatchSummary** and **BatchClose**.

BatchSummary and *BatchClose* are meant to offer a quick, snapshot comparison of batch item counts and totals between the Merchant batch file stored on MercuryPay's servers and their local totals on the POS application. The steps and XML examples below illustrate the logic behind this sequence.

Note: **Counts** are formatted as integers. **Amounts** are formatted as two decimal places, including the decimal.

Step	XML
1. POS connects to MercuryPay's servers and sends an ADMIN BatchSummary request.	<pre><?xml version="1.0"?> <TStream> <Admin><!--"Admin" is the high level element here--> <MerchantID>118725340908147</MerchantID> <TranCode>Batch Summary</TranCode> <Memo>MPS Transact 1.2.0.5</Memo> </Admin> </TStream></pre>
2. POS receives response confirming MercuryPay's <i>Batch Summary</i> Credit and Debit count and Amount values. <i>Total Items</i> , <i>NetBatchTotal</i> and <i>BatchNo</i> are also	<pre><?xml version="1.0"?> <RStream> <CmdResponse> <ResponseOrigin>Processor</ResponseOrigin> <DSIXReturnCode>000000</DSIXReturnCode> <CmdStatus>Success</CmdStatus> <TextResponse>OK</TextResponse> <UserTraceData></UserTraceData></pre>

Step	XML
included.	<pre> </CmdResponse> <BatchSummary> <MerchantID>118725340908147</MerchantID> <BatchNo>1225</BatchNo> <BatchItemCount>18</BatchItemCount> <NetBatchTotal>16.79</NetBatchTotal> <CreditPurchaseCount>6</CreditPurchaseCount> <!--Credit numbers--> <CreditPurchaseAmount>42.25</CreditPurchaseAmount> <CreditReturnCount>5</CreditReturnCount> <CreditReturnAmount>31.54</CreditReturnAmount> <DebitPurchaseCount>5</DebitPurchaseCount> <!--Debit numbers--> <DebitPurchaseAmount>10.17</DebitPurchaseAmount> <DebitReturnCount>2</DebitReturnCount> <DebitReturnAmount>4.09</DebitReturnAmount> </BatchSummary> </RStream> </pre>
3. The POS can now programmatically compare the information from the <i>BatchSummary</i> (as shown above) to the counts and totals in the local POS reports, then prompt the merchant to confirm and continue or flag errors and make corrections. The POS <i>BatchClose</i> must match the Summary counts and totals EXACTLY.	
4. If totals match, then the POS sends an Admin BatchClose request. The local <i>BatchClose</i> amount and count values MUST match the values received from the <i>BatchSummary</i> .	<pre> <?xml version="1.0"?> <TStream> <Admin> <MerchantID>118725340908147</MerchantID> <OperatorID>test</OperatorID> <TranCode>Batch Close</TranCode> <BatchNo>1225</BatchNo> <BatchItemCount>18</BatchItemCount> <!-- All counts are whole integers--> <NetBatchTotal>16.79</NetBatchTotal> <!--Amounts are formatted as two decimal places--> <CreditPurchaseCount>6</CreditPurchaseCount> <CreditPurchaseAmount>42.25</CreditPurchaseAmount> <CreditReturnCount>5</CreditReturnCount> <CreditReturnAmount>31.54</CreditReturnAmount> <DebitPurchaseCount>5</DebitPurchaseCount> <DebitPurchaseAmount>10.17</DebitPurchaseAmount> <DebitReturnCount>2</DebitReturnCount> <DebitReturnAmount>4.09</DebitReturnAmount> <Memo>MPS Transact 1.2.0.5</Memo> </Admin> </TStream> </pre>
5. Response is returned confirming the success of <i>BatchClose</i> and repeats the values. If the totals do NOT match, an error will be returned with <i>TextResponse</i> of INVLD BAL STL . Similarly, if a batch is settled successfully and then is attempted to be closed a second time, an error will be returned with a <i>TextResponse</i> of INVLD BAL STL .	<pre> <?xml version="1.0"?> <RStream> <CmdResponse> <ResponseOrigin>Processor</ResponseOrigin> <DSIXReturnCode>000000</DSIXReturnCode> <CmdStatus>Success</CmdStatus> <TextResponse>OK TEST</TextResponse> <UserTraceData></UserTraceData> </CmdResponse> <BatchClose> <MerchantID>118725340908147</MerchantID> <OperatorID>test</OperatorID> <BatchNo>1225</BatchNo> <BatchItemCount>18</BatchItemCount> <NetBatchTotal>16.79</NetBatchTotal> <CreditPurchaseCount>6</CreditPurchaseCount> <CreditPurchaseAmount>42.25</CreditPurchaseAmount> <CreditReturnCount>5</CreditReturnCount> <CreditReturnAmount>31.54</CreditReturnAmount> <DebitPurchaseCount>5</DebitPurchaseCount> <DebitPurchaseAmount>10.17</DebitPurchaseAmount> </pre>

Step	XML
	<pre><DebitReturnCount>2</DebitReturnCount> <DebitReturnAmount>4.09</DebitReturnAmount> <ControlNo>033204606</ControlNo> </BatchClose> </RStream></pre>

Recommendation

Safeguards should be implemented to alert the merchant that an open batch is pending and needs to be closed. It is important to train the end-users of merchant-initiated batch close POS systems to **close their batches daily**.

Appendix 7 – MercuryPay Security Features

Point-to-Point Encryption and Tokenization

Point-to-Point Encryption Overview

Point-to-Point Encryption (**P2PE**) first evolved on the MercuryPay platform as End-to-End Encryption (**E2E**), a secure industry standard that encrypts cardholder data from the point of entry. This data remains encrypted until it reaches its intended destination where it is decrypted and used to process the transaction. This guide will use the term E2E and P2PE interchangeably.

P2PE is an increasingly critical security feature and is strongly recommended for all card-present POS development. Security is increased because there is no clear-text cardholder data in the POS application. This also removes developers from the scope of PA-DSS.

As a PA-DSS validated and listed level-one service provider, MercuryPay is the only end point of the E2E pathway using our custom secure decryption environment to validate, decrypt and process the data. MercuryPay's cardholder data environment (CDE) is reviewed quarterly under stringent security audits which include a thorough analysis of our decryption environment. Given that the CDE handles cardholder data, this environment is required to be, and has been determined to be, compliant with PCI DSS.

MercuryPay's supported encryption peripherals are PCI PTS compliant and listed. Our device manufacturers are leaders in both the payment and encryption industries--VeriFone, Infinite Peripherals, IDTech, Ingenico and MagTek. As per PCI encryption security standards, devices are only allowed to be injected by certified third-party injection vendors.

MercuryPay's Tokenization Technology: MToken™ Overview

Using a process similar to our E2E service, **MToken** is a security technology that replaces sensitive cardholder data with a token for POS system storage and subsequent use. When requested by the POS, a token is generated and returned with the transaction authorization approval response. The token can then be used in place of the cardholder account number to perform subsequent transactions if required, or it can be stored safely for card-on-file recurring billing programs.

E2E and MToken Combined

E2E encryption with tokenization provides the best overall card data security solution. By using end-to-end encryption for initial card data entry, and storing a token for subsequent transactions, the point-of-sale system is removed from handling, storing, or transmitting unencrypted card data altogether.

- MercuryPay's end-to-end encryption service eliminates the risks involved with cardholder data starting from the point of card entry into the encrypted reader directly to the MercuryPay platform.
- By replacing card data with a token record, MercuryPay's tokenization processing helps merchants achieve PCI compliance and enable them to safely implement features such as recurring billing, card-on-file, and gratuity handling.

Technical Details: MercuryPay Encryption Specifications

Note: When implementing a direct integration using the Datacap DSIClientX control or MercuryPay's Web Services platform, developers are (used to be) required to first complete a direct PIN Pad integration where they work with the device manufacturer to learn how to parse and validate the encrypted output blocks; this was done before they can be built into a transaction request. When using Datacap's comprehensive suite of secure devices along with the dsiEMVUS and dsiPDCX, this effort is all handled by the Datacap controls. Though the details below may no longer be relevant for, or in the control of, EMV era Datacap developers, the end logic of building the XML request with encryption would apply.

XML Account level elements in the transaction requests

For E2E encrypted transaction requests, four additional data elements are added on the **Account** level of the transaction request:

```

<Account>
  <EncryptedFormat>MagneSafe</EncryptedFormat>
  <AccountSource>Swiped</AccountSource>
  <EncryptedBlock>EncryptedBlock</EncryptedBlock>
  <EncryptedKey>EncryptedKey</EncryptedKey>
</Account>

```

These elements are described below:

Element	Req	Min	Max	Type	Description
EncryptedFormat	Y	1	25	AN	Encryption format: MagneSafe . MagTek's MagneSafe v1.0 specification standards were used as a basis to design MercuryPay's decryption server's protocols
AccountSource	Y	1	25	AN	Source of encrypted block: Swiped, Keyed, Contactless
EncryptedBlock	Y	72	224	AN	Encrypted Track2 or manual entry pseudo Track data. Although both track blocks will be returned, only Track2 EncryptedBlock is used.
EncryptedKey	Y	20	20	AN	Derived Unique Key from Secure Card Reader

Encrypted Data Best Practices

- Never store the *EncryptedBlock* or *EncryptedKey* after authorization.
- In systems that support "Store and Forward" functionality, *EncryptedBlock* or *EncryptedKey* information may be stored prior to authorization only.
- Unless performed on a validated system, developers must code to accept manually-entered card data *only* through the encryption device and not through any native key-entry options; this would result in sending non-encrypted data, and cause the request to fail. Failure to use a supported encryption device for manual entry may expose the developer to additional PA-DSS and PCI remediation requiring a validated security assessor audit.

E2E XML Examples

Encrypted Credit: Sale

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <MerchantID>12345678</MerchantID> <!--MIDs must be E2E enabled-->
    <LaneID>02</LaneID>
    <TranType>Credit</TranType>
    <PartialAuth>Allow</PartialAuth>
    <TranCode>Sale</TranCode>
    <InvoiceNo>1</InvoiceNo>
    <RefNo>1</RefNo>
    <Memo>Product v1.1</Memo>
    <Account>
      <EncryptedFormat>MagneSafe</EncryptedFormat>
      <AccountSource>Swiped</AccountSource>
      <EncryptedBlock>D7019F295F395B56233CA41F357CD5563D6FB07E78D
418EB296473AFC02989237D97F9B40D503272</EncryptedBlock><!-- encrypted block to a minimum of 72 and maximum
length of 224-->
      <EncryptedKey>9500530000000720000F</EncryptedKey>
    </Account>
    <Amount>
      <Purchase>2.25</Purchase>
    </Amount>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>12345678</MerchantID>
    <AcctNo>549999XXXXXX6781</AcctNo>
    <ExpDate>XXXX</ExpDate>
    <CardType>M/C</CardType>
    <TranCode>Sale</TranCode>
    <AuthCode>MC0225</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>0001</RefNo>
    <InvoiceNo>1</InvoiceNo>
    <Amount>
      <Purchase>2.25</Purchase>
      <Authorize>2.25</Authorize>
    </Amount>
    <AcqRefData>b8238953#20#20#20#20#20#20#20e00</AcqRefData>
    <ProcessData>|00|410100201000</ProcessData>
  </TranResponse>
</RStream>

```

Note: Truncating <AcctNo> and <ExpDate> in Response: All credit card data will be returned truncated for encrypted transaction responses.

- The first six and last four digits of the *AcctNo* will be returned with middle six digits replaced with Xs,
- *ExpDate* will be masked by Xs.

StoreCard/Gift encrypted PrePaid NoNSFSale Example

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <lpPort>9100</lpPort>
    <MerchantID>023120578416030</MerchantID>
    <LaneID>02</LaneID>
    <TranType>PrePaid</TranType>
    <TranCode>NoNSFSale</TranCode>
    <Memo>Product v1.1</Memo>
    <InvoiceNo>2</InvoiceNo>
    <RefNo>2</RefNo>
    <Account>
      <EncryptedFormat>MagneSafe</EncryptedFormat>
      <AccountSource>Swiped</AccountSource>
      <EncryptedBlock>BF8BAF6BBE158E40EC D308AFD01359C29F0300D4AC06A3BF3300B2774C3812D9CF5052
C7532CBDFD</EncryptedBlock>
      <EncryptedKey>9500530000000720000F</EncryptedKey>
    </Account>
    <Amount>
      <Purchase>3.10</Purchase>
    </Amount>
  </Transaction>
</TStream>

```


E2E Error Responses

Connectivity Errors

In the event that MercuryPay's encryption service is unavailable, the following error will be returned:

```
<ResponseOrigin>Server</ResponseOrigin>
<DSIReturnCode>004116</DSIReturnCode>
<CmdStatus>Error</CmdStatus>
<TextResponse>Decryption Service Unavailable.</TextResponse>
```

MercuryPay's E2E Merchant Check Mismatch Error

- If the merchant is not setup for E2E but the E2E encryption data elements (*EncryptedFormat*, *AccountSource*, *EncryptedBlock*, *EncryptedKey*) are sent in the request, then the transaction will return "Merchant setting does not accept Encrypted data."
- If merchant is setup for E2E but the request does not contain the E2E Encryption elements, then the transaction will return "Merchant setting requires Encrypted data."

```
<CmdResponse>
  <ResponseOrigin>Server</ResponseOrigin>
  <DSIReturnCode>004115</DSIReturnCode>
  <CmdStatus>Error</CmdStatus>
  <TextResponse>Merchant setting requires Encrypted data</TextResponse>
```

E2E Invalid Field Errors

If one of the E2E Encryption elements is missing in the transaction request, an "Invalid Field" will be returned qualified by the missing element.

```
<CmdResponse>
  <ResponseOrigin>Server</ResponseOrigin>
  <DSIReturnCode>100254</DSIReturnCode>
  <CmdStatus>Error</CmdStatus>
  <TextResponse>Invalid Field - Encrypted Block</TextResponse>
```

Decryption Service Unavailable errors: Corrupt, Malformed or Incorrect Data

- In the event that one of the E2E encryption data elements contains corrupted, malformed or in any way incorrect data, MercuryPay's encryption service will return a text response containing Decryption Service Unavailable followed by the field detected to contain the failed data.
- These failure messages vary and are dynamically generated based on the reason for the failure:

```
<CmdResponse>
  <ResponseOrigin>Server</ResponseOrigin>
  <DSIXReturnCode>004116</DSIXReturnCode>
  <CmdStatus>Error</CmdStatus>
  <TextResponse>Decryption Service Unavailable-200 Response with Status: Failure
  Message:Invalid EncryptedBlock</TextResponse>
```

Note: Merchant setting, Invalid Field and some "Decryption Service Unavailable" error messages described above are typically determined by MercuryPay's servers *prior* to decryption. If an encrypted transaction request passes this initial set of checks it is passed to MercuryPay's decryption server.

There is another set of error messages at this stage of the decryption process: In the event that an incorrect *EncryptedKey* is sent, or the *EncryptedBlock* is not decryptable, in the wrong format, or if the decrypted data is invalid, the following error format is used. Note the differences in the failure message for "Failed to decrypt," "Encrypted block invalid," or "Decrypted data invalid":

- Error # 004118 - Decryption Service Unavailable. - 200 Response With Status:Failure Message:**Failed to decrypt**

- Error # 004118 - Decryption Service Unavailable. - 200 Response With Status:Failure
Message:**Encrypted block invalid**
- Error # 004118 - Decryption Service Unavailable. - 200 Response With Status:Failure
Message:**Decrypted data invalid**

MercuryPay Tokenization – MToken

MToken Overview

Tokenization is, from beginning to end, a proprietary and closed process. There is no key loaded or injected into any external device. MercuryPay is the sole holder, generator and custodian of the Token process that generates and then unlocks a token. The POS must always request a Token first or the request will fail. A token is uniquely generated at MercuryPay *per transaction*, goes through several algorithmic/cipher processes, and is then returned to the POS in a standard Base64 format—the POS simply returns it, “as-is”, in place of card data.

Note: Other tokenization methods exist on the broader Worldpay from FIS integrated payments platform including Express PASS tokenization and Worldpay’s OmniToken. MToken is unique to MercuryPay and cannot be exchanged or shared with the other token methodologies.

MTokens are unique, dynamically created card data reference numbers.

- Referred to as **RecordNo** in the Datacap vernacular, tokens may only be generated “by request.” That is, systems enabled to support MercuryPay’s tokenization will first request a *RecordNo* with the relevant *Frequency*.
- This is done with a standard transaction request using two additional XML tags: **<RecordNo>** and **<Frequency>**. Token frequencies may either be for **OneTime** use or for **Recurring**, card-on-file environments.
- MercuryPay processes the transaction with the token request normally, but then routes the response to our tokenization servers where a *RecordNo* is generated based on the request criteria.
- The *RecordNo* is then passed back to the local POS with the authorization response using the same XML tags.
- The POS, per business need, may store the token “as-is” for future use, be that retouching a transaction for tip modification or building recurring databases for weekly, monthly or annual billing programs.

Note: A token will be returned for any credit transaction in place of the account number and expiration date. **The token can only be decrypted by MercuryPay.** When decrypted, the token will only refer to the cardholders Personal Account Number (**PAN**) and Expiration Date, and is processed as a keyed, card-not-present transaction.

Merchant Settings

As with E2E encryption, merchant accounts are set up to reflect POS system tokenization capability. If the merchant is setup for tokenization *and* both *RecordNo* and *Frequency* tags are in the request, then MercuryPay will build and return the token record on all credit responses.

MToken Types

- A **OneTime** token expires 6 months from the date the token record is generated. With each subsequent use a new token is generated and returned, adding an additional 6 months. Recommended for all daily use transactions or for “card-on-file” supported systems and used for non-recurring, card-on-file transactions.
- A **Recurring** token expires 24 months from the date the token record is generated. With each subsequent use a new token is generated and returned, adding an additional 24 months. Used for recurring environments where weekly, monthly or annual subscription billing is implemented. Recurring tokens are *not* recommended for daily transaction activity. “Recurring payment” may appear on the

cardholder's bank statement and may impact additional charges. When using a Recurring MToken with a card on file/credentials on file program, additional card brand regulations apply.

Note: Corporate Credit Cards may or may not accept recurring billing; this is dependent on the corporation's policies and on the issuing bank.

MToken Maintenance, Storage and Best Practices

Developers are required to securely handle, store and maintain MToken Records.

- Only store a token record if there is a business need and if this information supports your targeted business/merchants.
- Note that although AVS data may be stored as non-sensitive card data, it is important to remember that CVV data *may never be stored*.
- Build in retention and expiration timelines for any stored token record. Note that because of truncated card expiration data returned in the token response, developers should, if needed, obtain the expiration date prior to processing the request.
- Always dispose of expired token records.
- Tokens are dynamically refreshed per use. A new token record will always be returned with each subsequent transaction. If storing token data, store only the most recent token record returned.
- Tokens are generated for **OneTime** or **Recurring**. A token generated for *OneTime* use cannot be used for *Recurring*; a token generated for *Recurring* use cannot be used for *OneTime*.
- Token use is relative to a single Merchant location Business ID (**BID**) but may be used across other locations under certain conditions using MercuryPay Multiple-Merchant Tokenization.

Multiple-Merchant Tokenization (MMT)

MMT enables a single business entity with multiple locations to use MercuryPay's MToken platform across all of their locations **if** they have a centrally housed, cross-location database infrastructure in place to support it.

Tokens are tied to merchant locations and can only be used in MMT situations if they are "grouped". Upon request, MercuryPay's Underwriting account specialists can authorize the creation of an MMT grouping that allows token sharing from one location to another. In most cases this will be implemented at the merchant level without issue, but there are safeguards in place to prevent inadvertent users of MMT outside the prescribed group. For example, if "ungrouped" Merchant A tries to redeem a token issued from "grouped" Merchant B, Merchant A would receive a response of "Token Invalid."

RecordNo and Frequency Specifications

Tokens may be requested within standard XML either with clear text cardholder data in validated applications, or alongside MercuryPay's E2E encryption data. **<RecordNo>** is used to request a token record, and **<Frequency>** is used to define the specific type of token to be generated.

- The initial *RecordNo* element will always contain **RecordNumberRequested**.
- The *Frequency* element has two options:
 - **OneTime** – token used in subsequent transactions.
 - **Recurring** – tokens used for recurring billing.

```
<Transaction>
  <RecordNo>RecordNumberRequested</RecordNo>
  <Frequency>OneTime</Frequency>
</Transaction>
```

Specifications for each element are listed below:

Element	Req	Min	Max	Type	Description
Transaction:	Y	1	100	AN	Used on the initial request to generate a token using

RecordNo						'RecordNumberRequested'. Use the actual token returned from MercuryPay in subsequent "ByRecordNo" requests
Transaction: Frequency	Y	7	20	AN		Type of Token requested: 'OneTime' or 'Recurring'
TranResponse: RecordNo	Y	48	100	AN		Token record returned from Mercury

Note: *Frequency* values must stay consistent in all subsequent uses of MToken requests.

- A **OneTime Sale** request must be followed with a **OneTime VoidSaleByRecordNo** Request
- A **Recurring PreAuth** request can only be followed by a **Recurring PreAuthCaptureByRecordNo** request. Inconsistent use of *Frequency* will cause the subsequent transaction request to fail with the following error:
Error # 004119- Token Service Unavailable. - 200 Response With Status: Failure Message: Parse token failure.

XML Example: Credit Sale, OneTime Token Record

Request

```
<?xml version="1.0"?>
<TStream>
  <Transaction>
    <MerchantID>12345678</MerchantID> <!--MID must be token enabled-->
    <OperatorID>Test</OperatorID>
    <LaneID>02</LaneID>
    <TranType>Credit</TranType>
    <PartialAuth>Allow</PartialAuth>
    <TranCode>Sale</TranCode>
    <InvoiceNo>20</InvoiceNo>
    <RefNo>20</RefNo>
    <Memo>Product v1.1</Memo>
    <RecordNo>RecordNumberRequested</RecordNo> <!--tag used to request a token on response-->
    <Frequency>OneTime</Frequency> <!--indicates one-time token-->
    <Account>
      <Track2>Track Data</Track2>
    </Account>
    <Amount>
      <Purchase>2.25</Purchase>
    </Amount>
  </Transaction>
</TStream>
```

Response

```
<?xml version="1.0" ?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
  </CmdResponse>
  <TranResponse>
    <MerchantID>023358150511666</MerchantID>
    <AcctNo>*****6781</AcctNo> <!--Note truncated Account number-->
    <ExpDate>XXXX</ExpDate> <!--ExpDate always returned as XXXX-->
    <CardType>VISA</CardType>
    <TranCode>Sale</TranCode>
```

```

    <AuthCode>000027</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>0003</RefNo>
    <InvoiceNo>0020</InvoiceNo>
    <OperatorID>Test</OperatorID>
    <Memo>Product v1.1</Memo>
    <Amount>
        <Purchase>2.25</Purchase>
        <Authorize>2.25</Authorize>
    </Amount>
    <AcqRefData>b8238953#20#20#20#20#20#20#20e00</AcqRefData>
    <RecordNo>BuFzLtekGFrTsiCOxI59PCQUfZe32C3YYXgXuPuFU64yEAQQADIQAIX
</RecordNo><!--typical token length is 48-53 char, but should be coded to 100-->
    <ProcessData>|00|210100201000</ProcessData>
</TranResponse>
</RStream>

```

XML Example: Credit Sale, Recurring Token Record

Request

```

<?xml version="1.0" ?>
<TStream>
    <Transaction>
        <MerchantID>12345678</MerchantID>
        <OperatorID>TEST</OperatorID>
        <LaneID>02</LaneID>
        <TranType>Credit</TranType>
        <PartialAuth>Allow</PartialAuth>
        <TranCode>Sale</TranCode>
        <InvoiceNo>0021</InvoiceNo>
        <RefNo>0021</RefNo>
        <Memo>Product v1.1</Memo>
        <RecordNo>RecordNumberRequested</RecordNo>
        <Frequency>Recurring</Frequency> <!--note frequency is the only change-a recurring token is valid for
24 months-->
        <Account>
            <Track2>Track Data</Track2>
        </Account>
        <Amount>
            <Purchase>3.25</Purchase>
        </Amount>
    </Transaction>
</TStream>
Response
<?xml version="1.0"?>
<RStream>
    <CmdResponse>
        <ResponseOrigin>Processor</ResponseOrigin>
        <DSIXReturnCode>000000</DSIXReturnCode>
        <CmdStatus>Approved</CmdStatus>
        <TextResponse>AP</TextResponse>
        <UserTraceData></UserTraceData>
    </CmdResponse>
    <TranResponse>
        <MerchantID>023358150511666</MerchantID>
        <AcctNo>400300XXXXXX6781</AcctNo>
        <ExpDate>XXXX</ExpDate>
        <CardType>VISA</CardType>
        <TranCode>Sale</TranCode>
        <AuthCode>000027</AuthCode>
        <CaptureStatus>Captured</CaptureStatus>
    </TranResponse>
</RStream>

```

```

    <RefNo>0004</RefNo>
    <InvoiceNo>0021</InvoiceNo>
    <OperatorID>TEST</OperatorID>
    <Memo>Product v1.1</Memo>
    <Amount>
      <Purchase>3.25</Purchase>
      <Authorize>3.25</Authorize>
    </Amount>
    <AcqRefData>b8238953#20#20#20#20#20#20#20e00</AcqRefData>
    <RecordNo>Im1taJnzdyWvOEJgDSFxgOAmukvmORuntn/wRb+YK/szEAQQADIQAAAR</RecordNo>
    <ProcessData>|00|210100221000</ProcessData>
  </TranResponse>
</RStream>

```

Using E2E Encryption with an initial token request

End-to-End encryption technology adds another layer of security to the tokenization process as it eliminates the handling of any card data directly from the point of swipe entry. E2E and Tokenization must *both* be supported on the local POS system and configured on the MercuryPay Merchant ID.

XML Example: Credit Sale, One-Time Token Record using E2E data

Request

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <MerchantID>118725340</MerchantID>
    <OperatorID>Test</OperatorID>
    <LaneID>02</LaneID>
    <TranType>Credit</TranType>
    <TranCode>Sale</TranCode>
    <PartialAuth>Allow</PartialAuth>
    <InvoiceNo>20</InvoiceNo>
    <RefNo>20</RefNo>
    <RecordNo>RecordNumberRequested</RecordNo>
    <Frequency>OneTime</Frequency>
    <Account>
      <EncryptedFormat>MagneSafe</EncryptedFormat>
      <AccountSource>Swiped</AccountSource>
      <EncryptedBlock>CBF9CFE5A8B8E5B181ADD888141E87732AAE3387A819C2F79921E4BF00AD21
EF24480F7840DB03FD</EncryptedBlock>
      <EncryptedKey>9500530000000720003D</EncryptedKey>
    </Account>
    <Amount>
      <Purchase>2.25</Purchase>
    </Amount>
  </Transaction>
</TStream>

```

Response

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>11872534</MerchantID>

```



```

<AcctNo>*****6781</AcctNo>
<ExpDate>XXXX</ExpDate>
<CardType>VISA</CardType>
<TranCode>Sale</TranCode>
<AuthCode>000027</AuthCode>
<CaptureStatus>Captured</CaptureStatus>
<RefNo>0004</RefNo>
<InvoiceNo>20</InvoiceNo>
<Amount>
    <Purchase>2.25</Purchase>
    <Authorize>2.25</Authorize>
</Amount>
<AcqRefData>b8238953#20#20#20#20#20#20e00</AcqRefData>
<RecordNo>58MuH48lgTDvJirMWmmXKbGlsv7cmO2CcahIPML0FTsyEAQQADIQAAle</RecordNo>
    <ProcessData>|00|210100200000</ProcessData>
</TranResponse>
</RStream>

```

Subsequent Usage of Token: "ByRecordNo"

As illustrated in the above examples, a Token will be passed back to the POS on all credit TranCodes (Debit, EBT and Gift will not return a Token). The returned token tag is "Record Number" (RecordNo) and all subsequent requests using a token will be submitted by adding *ByRecordNo* to the TranCode.

- The following table lists the TranCodes for Token Requests and corresponding subsequent usage *ByRecordNo* TranCodes:

Credit TranCodes Used to Request a Token	TranCodes for Subsequent Use of Token
Adjust	AdjustByRecordNo
CardLookUp	NA
FSASale	FSASaleByRecordNo
PreAuth	PreAuthByRecordNo
PreAuthCapture	PreAuthCaptureByRecordNo
Return	ReturnByRecordNo
ReverseFSASale	ReverseFSASaleByRecordNo
Sale	SaleByRecordNo
VoidReturn	VoidReturnByRecordNo
VoidSale	VoidSaleByRecordNo
ZeroAuth	NA

- When building a *ByRecordNo* XML request, use the appropriate *ByRecordNo* TranCode from the above table and replace the entire Account level tags with the actual token using the *RecordNo* tag. Note, too, that the *Frequency* must match the original requested token frequency.

Note: A *CardLookUp* or *ZeroAuth* transaction may be used to confirm card usage, verify account data, and to request a *RecordNo* without doing an actual authorization on the card. *CardLookUp* does not authenticate the card number, but only indicates that the card passes a valid MOD10 check (Luhn) and its BIN usage response is FSA, DEBIT or OTHER. *ZeroAuth* can be used to confirm that the AVS and CVV data is accurate. Tokens returned from a *CardLookUp* or *ZeroAuth* do not regenerate track data/sensitive authentication data; when used for any subsequent *ByRecordNo* transaction are processed as *keyed*.

XML Example: ReturnByRecordNo using Token Record in place of cardholder data**Request**

```

<?xml version="1.0"?>
<TStream>
  <Transaction>
    <MerchantID>0233581</MerchantID>
    <LaneID>02</LaneID> <!--required for multi-terminal environments-->
    <OperatorID>Test</OperatorID>
    <TranType>Credit</TranType>
    <TranCode>ReturnByRecordNo</TranCode> <!--Note use of Token TranCode-->
    <PartialAuth>Allow</PartialAuth>
    <InvoiceNo>0025</InvoiceNo>
    <RefNo>0025</RefNo>
    <RecordNo>BuFzLtekGFrTsiCOxI59PCQUfZe32C3YYXgXuPuFU64yEAQQADIQAAX</RecordNo> <!--note in place of Account level setting is the actual RecordNo and Frequency-->
    <Frequency>OneTime</Frequency> <!--must match original requested Frequency-->
    <Amount>
      <Purchase>3.75</Purchase>
    </Amount>
  </Transaction>
</TStream>

```

Response

Note: The **ByRecordNo** transaction response truncates *AcctNo* and *ExpDate*. As seen in the example below, all card data will be returned truncated for tokenized transactions.

- For an increased level of card data security, MercuryPay highly recommends storing only the last four digits of the *AcctNo* field for all POS-side reports.

```

<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Processor</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>0233581</MerchantID>
    <AcctNo>*****6781</AcctNo>
    <ExpDate>XXXX</ExpDate> <!--Expiration is always returned truncated. If required for local records, always pull from the request-->
    <CardType>M/C</CardType>
    <TranCode>Return</TranCode>
    <AuthCode>000089</AuthCode>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>0032</RefNo>
    <InvoiceNo>0025</InvoiceNo>
    <OperatorID>Test</OperatorID>
    <Amount>
      <Purchase>3.75</Purchase>
      <Authorize>3.75</Authorize>
    </Amount>
    <AcqRefData>b8238953#20#20#20#20#20#20#20#20e00</AcqRefData>
    <RecordNo>BKjylZ/QSYkM30QclgXGIXHr+fRxLFt73MJZPAeQhjoyEAQQADIQAAX</RecordNo> <!--a new token is always returned in the response-->
    <ProcessData>|20|210100700000</ProcessData>
  </TranResponse>
</RStream>

```



```
<MerchantID>023358150511666</MerchantID>  
<LaneID>02</LaneID>  
<OperatorID>Test</OperatorID>  
<TranType>Credit</TranType>  
<TranCode>PreAuthCaptureByRecordNo</TranCode>  
<PartialAuth>Allow</PartialAuth>  
<InvoiceNo>0027</InvoiceNo>  
<RefNo>0027</RefNo>  
<RecordNo>3Bt/yWkeon6zWfJpVDi4uLSs/A1x68uBKib/m82zkW0yEAQQADlQAAc</RecordNo>  
<Frequency>OneTime</Frequency>  
<Amount>  
    <Purchase>5.25</Purchase>  
    <Authorize>5.25</Authorize>  
    <Gratuity>1.00</Gratuity>  
</Amount>  
<TranInfo>  
    <AuthCode>000038</AuthCode>  
    <AcqRefData>Kb8238953#20#20#20#20#20#20e00</AcqRefData>  
</TranInfo>  
</Transaction>  
</TStream>
```

Response

```
<RStream>
  <CmdResponse>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>AP</TextResponse>
  </CmdResponse>
  <TranResponse>
    <MerchantID>023358150511666</MerchantID>
    <AcctNo>400300XXXXXX6781</AcctNo>
    <ExpDate>XXXX</ExpDate>
    <Card Type>VISA</Card Type>
    <TranCode>PreAuth Capture</TranCode>
    <Auth Code>000038</Auth Code>
    <CaptureStatus>Captured</CaptureStatus>
    <RefNo>0012</RefNo>
    <InvoiceNo>00027</InvoiceNo>
    <Amount>
      <Purchase>5.25</Purchase>
      <Authorize>2.11</Authorize>
      <Gratuity>1.00</Gratuity>
    </Amount>
    <AcqRefData>K8238953#20#20#20#20#20#20e00</AcqRefData>
    <RecordNo>|1u0bAbY+5uBhZHnEa2AJ9yC4xGakBCwG7E91cWmXikyEAQQADIQAADA </RecordNo>
    <ProcessData>|15|210100700000</ProcessData>
  </TranResponse>
</RStream>
```

Using Tokenization for Recurring Billing

The initial and subsequent use of tokens requested for recurring billing purposes follows the same set of XML requests except that the initial frequency is requested as “Recurring”. Recurring token records expire after 2 years. It is important to have provisions in place to store token records appropriately and to always store only the most recently returned token data. When used, Recurring tokens will appear on the cardholder’s statement as a “Recurring Charge” and may be subject to additional issuer fees.

Important Notes

- Certain credit and debit cards have restrictions that do not allow recurring transactions. If these restrictions are in place, the issuing bank may decline a transaction that includes the recurring tokens in the transaction data.
- In addition, recurring tokens should *only* be used for recurring billing.
- Additional Card on File/Credentials on File requirements will apply.
- Token frequency values must match on all subsequent usage. A *recurring* token can only be used for a subsequent *recurring* request just as a *onetime* token can only be used for a *onetime* request.

Card Brand Network Note Initial token requests will always require an actual transaction to be submitted in order to retrieve the generated token. Sending a *PreAuth* as a way of receiving a token with no intention of capturing the transaction constitutes a "**misuse of the authorization network**" and causes additional fees generated by the card associations. It is recommended to use the *CardLookup* or *ZeroAuth* transaction for this purpose—a token is also returned in the response and the card is not used for authorization.

MToken Errors Responses

There are four categories of MToken error responses: Connectivity, Merchant Check, Invalid Fields and Corrupted Data errors.

Connectivity Errors

A connection failure to the tokenization server will return an "Unavailable" response. The following response occurs if the **Decryption service** is unavailable:

```
<CmdResponse>
  <ResponseOrigin>Server</ResponseOrigin>
  <DSIReturnCode>004118</DSIReturnCode>
  <CmdStatus>Error</CmdStatus>
  <TextResponse>Decryption Service Unavailable. - Status Code Not 200 - 503</TextResponse>
```

If the **Token service** is unavailable, the following response occurs:

```
<CmdResponse>
  <ResponseOrigin>Server</ResponseOrigin>
  <DSIReturnCode>004119</DSIReturnCode>
  <CmdStatus>Error</CmdStatus>
  <TextResponse>Token Service Unavailable.- 200 Response With Status:Error Message:Error
  generating token</TextResponse>
```

Merchant ID Token Setup Error

For a tokenization request to be validated, MercuryPay merchants must send both the required tokenization data elements AND be set up in the MercuryPay merchant tokenization tables. Only the credit TranCodes listed above are checked. Debit, EBT, Gift/PrePaid and ADMIN transactions are ignored.

- If a merchant is setup for tokenization AND both *RecordNo* and *Frequency* elements are in the request, then MercuryPay builds the token record. The token is sent back in the *RecordNo* element. In the event of a decline/error response, no token is generated.

Transactions will error if:

- *RecordNo* and *Frequency* elements are in the request, **but the merchant is not setup for tokenization**:

```
<CmdResponse>
  <ResponseOrigin>Server</ResponseOrigin>
  <DSIReturnCode>004117</DSIReturnCode>
  <CmdStatus>Error</CmdStatus>
```

```

    <TextResponse>Merchant Setting Does Not Accept RecordNo and
    Frequency.</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>

```

- If the merchant is setup for tokenization in the merchant tables, but **the request does not contain RecordNo and Frequency elements**:

```

<CmdResponse>
  <ResponseOrigin>Server</ResponseOrigin>
  <DSIReturnCode>004116</DSIReturnCode>
  <CmdStatus>Error</CmdStatus>
  <TextResponse>Merchant Setting Requires RecordNo and Frequency.</TextResponse>
</CmdResponse>

```

Invalid field or missing field errors

- **RecordNo is missing in the data element field.** This error returns the “Merchant Setting Requires RecordNo” message because the absence of a *RecordNo* is validated by the Merchant Table settings.

```

<CmdResponse>
  <ResponseOrigin>Server</ResponseOrigin>
  <DSIReturnCode>004116</DSIReturnCode>
  <CmdStatus>Error</CmdStatus>
  <TextResponse>Merchant Setting Requires RecordNo.</TextResponse>

```

- **Frequency is missing** in the data element field.

```

<CmdResponse>
  <ResponseOrigin>Server</ResponseOrigin>
  <DSIReturnCode>100256</DSIReturnCode>
  <CmdStatus>Error</CmdStatus>
  <TextResponse>Invalid Field-Frequency</TextResponse>

```

Corrupt RecordNo

- There is corrupted data in the *RecordNo* element, there is a mismatch in token frequency, or the token has expired:

```

<CmdResponse>
  <ResponseOrigin>Server</ResponseOrigin>
  <DSIReturnCode>004119</DSIReturnCode>
  <CmdStatus>Error</CmdStatus>
  <TextResponse>Token Service Unavailable. – 200 Response With Status:Failure
  Message:Parse token failure</TextResponse>

```

- Corrupt Frequency error:

```

<CmdResponse>
  <ResponseOrigin>Server</ResponseOrigin>
  <DSIReturnCode>100256</DSIReturnCode>
  <CmdStatus>Error</CmdStatus>
  <TextResponse>Invalid Field-Frequency</TextResponse>
  <UserTraceData></UserTraceData>
</CmdResponse>

```


Appendix 8 – Duplicate Transactions and TimeOuts

Preventing Unintended Duplicate Transactions and TimeOuts

The unintended duplication of a payment transaction (that is, charging the card twice for the same transaction) is one of the challenges for POS developers. Detecting and preventing unintended duplicate transactions is part strategy and part art: the strategy is to differentiate between intended and unintended duplicates; the art is using variables that can detect a *true* duplicate, using the response from MercuryPay based on those variables, and then, maybe, putting on the brakes.

MercuryPay supports three types of logic designed to check for, suppress or allow duplicates. All act on the merchant's current open batch, and support the following transactions:

- **Credit:** *Sale, FSASale, PreAuthCapture, VoiceAuth and Return.* *PreAuth* is not on this list and cannot be checked using this duplicate logic.
 - **Debit:** *Sale, Return*
1. **Enhanced logic.** This compares three variables: card number, purchase amount, and invoice number. When a duplicate is detected a message is returned, **APPROVED APDUPE**. The cardholder is only charged once. ***This is the preferred and more robust duplicate logic supported by Mercury***, as it can detect unintentional duplicates while allowing intentional duplicates to go through in the normal course of business. The key to this form of duplicate checking is correctly implementing the invoice number (**<InvoiceNo>**).
 2. **Legacy logic.** This compares two variables: card number and purchase amount. In this case, duplicates are detected and stopped. The message returned is **DECLINED APDUPE**. This approach is typical for Canadian configurations and for systems that have implemented a transaction-level override function. The advantage of this method is that if a transaction with the same card number and purchase amount is detected, it will simply decline. The only way around this is to either change the amount value, or resend with a an override tag.
 3. **No duplicate logic.** The last tier disables checking for duplicates altogether. All duplicate transactions are processed. This method should be used with discretion, as it offers no protection against duplicate transactions. However it is appropriate for POS systems that manage their own duplicate logic.

Note: Because transaction routing now flows through Worldpay from FIS Express Gateway, some changes in the duplicate logic outlined in this guide may no longer be applicable. On the Express platform duplicates will be suppressed if they are received on the same Merchant account, the same amount, card and within the last 25 transactions in a merchant's open batch.

For U.S. developers, avoiding unintended duplicate transactions requires the correct use of the **<InvoiceNo>** data element. The *InvoiceNo* is a number generated by the POS system for each transaction, which increments each time it's generated, and serves as the unique identifier for that transaction. Incrementing the *InvoiceNo* allows transactions using the same card with the same amount to be processed normally, because each transaction will have a unique *InvoiceNo*.

Duplicates usually do not happen in the normal course of business--they happen when a transaction response is not received by the POS, and the merchant swipes (or keys) the card again.

If a transaction request DOES NOT receive a response for whatever reason (this can be due to Internet connectivity issues, legitimate timeouts) the POS needs to be able to retry the transaction using the same data as the first time. In this case, DO NOT increment the *InvoiceNo*, or change the amount. Simply resubmit the exact same transaction. In the meantime, the first transaction MAY have already hit the processing platform and is logged in the merchant batch file. By resubmitting the exact same data, the Tier 1 logic can work its magic:

- **If the transaction was NOT previously captured**, the response data will return the standard approved or declined processor's response.
- **If the transaction WAS previously captured**, the response data will indicate this by showing a *CmdStatus* of APPROVED, but the *TextResponse* will be **AP***. The second transaction approves, your POS recognizes "AP*" and the first transaction will not be charged.

MercuryPay's duplicate logic recognizes transactions with the same card number, same amount, and same *InvoiceNo* within the same open batch. Using the enhanced Tier 1 logic, the system flags the two transactions and transfers the approval to the most recent one processed. The cardholder is never charged twice. MercuryPay replaces the duplicated request with the original transaction information: *AuthCode*, *AcqRefData*, *ProcessData*, *RefNo*, CVV and AVS Result.

Handling Transaction Time-Outs

In this era of high-speed internet, transaction timeouts, though rare, still happen. Normally, transaction processing should take 2-3 seconds. However, internet latency and backend processing issues may impact normal processing times.

There are multiple "hops" in the typical transaction path:

1. From the merchant's POS environment to MercuryPay's NETePay Server
2. From the NETePay Server on to Express Gateway and then on to Vantiv RAFT
3. From Vantiv RAFT to the back-end Issuing Authorizing Networks
4. From the Authorizing Networks back to RAFT, Express and MercuryPay
5. Back from MercuryPay's platform to the merchant's POS

Timeouts can occur when transactions take longer than expected to travel through the above path. The expected times are based on values configured within the POS and along the transaction path. If the transaction time exceeds the configured thresholds, the transaction response may return a TIMEOUT message, or it may return no response at all. It depends on where the timeout occurs along the path.

As a rule, the POS should always set a response timeout value **greater** than the sum of all timeout values along the transaction path. If the timeout value is *less* than the sum of all the values along the path, there may not be any response message.

- For example, once a transaction enters MercuryPay's front door, it is timed as it passes through our system to the back end authorization networks. The longest we will wait is 30 seconds. If there is a delay greater than 30 seconds, the platform sends a Server Side TIMEOUT message to the POS. However, if the POS has a timeout value set for *less* than 30 seconds, the transaction will time out without receiving a response, and the POS will not know if the transaction has been authorized.
- Whether a timeout message is received or not, and to avoid the potential for duplications, it is recommended to resend the transaction with the same amount, card and *InvoiceNo*, as outlined in the previous section.

Error message	Description	Action
Timeout on Response	Returned if MercuryPay did not get a response from the processing back-end networks.	Re-run the exact transaction without incrementing the <i>InvoiceNo</i> . APPROVED/AP* indicates MercuryPay had the original request logged and the card will not be charged twice.
Timeout waiting for server Response	Returned if the merchant's POS did not get a response from MercuryPay.	A standard processor APPROVED or DECLINED response indicates that the original request was not logged.

Appendix 9 – Fraud Protection

Card Security Codes and Address Verification

In all merchant environments, manually entered (keyed) credit card data carries a higher risk of fraud. Whereas card-not-present merchants typically have practices in place to protect their business from fraud, the card-present environment is more vulnerable. Therefore, we highly recommend that POS systems provide fraud protection mechanisms that allow their merchants to make good business decisions. These fraud protection mechanisms are the implementation of Card Verification Data (**CVV**) and Address Verification Service (**AVS**). Using these options also enables merchants to qualify for better processing rates.

It is important to note that CVV and AVS results are returned *independently* from the authorization process. It is common for an issuer to **approve** a transaction but also send a **“no match”** CVV and/or AVS response. Regardless if the issuing bank sends back an approval, if the CVV or AVS data indicate a no-match or an otherwise questionable, potentially fraudulent result, POS applications should communicate the result along with the authorization response AND any applicable corrective measures. The appropriate corrective measure is determined by the merchant's environment, amount of the transaction, how often the card is being used and, if applicable, any shipping-billing address mis-match. In other words, merchants can only prevent fraud and protect themselves from chargebacks if the POS application they are using supplies them the features to do so.

General "rules of thumb" actions are provided below, but per card brand regulations, the variables that determine the action are relative to the merchant environment. As such, AVS and Card Security Codes are important tools used to help make business decisions on whether to proceed with a sale or delivery of goods and services. In all cases of questionable or partial CVV or AVS results where the transaction is APPROVED by the issuing bank, your POS application should build in configurable merchant settings to:

1. Prompt the mis-match to the end user,
2. Allow for the option to reject the transaction immediately by running a Reversal-VoidSale,
3. Allow the option to both process a Reversal-VoidSale and to allow for re-entry of the mis-matched data
4. Allow the option to accept the transaction authorization as-is, accepting the risk of the potential fraudulent transaction.

Card Verification Data (CVV/CVC2/CID)

Card verification data is referred to by several different acronyms: **CVV2/CVC2** (Cardholder Verification Value)/**CID** (Amex and Discover). It is a three- or four-digit code found on the back of a credit card (for Amex it is on the front of the card) that is used when card information needs to be manually entered (Mail Order/Telephone Order (MOTO) and eCommerce transactions) to determine that the card is physically with the customer who is placing the order, since the code can only be found on the card itself.

- The CVV request data element is: **<CVVData>123</CVVData>**
Format: Numeric only. Length: Min 3 Max 4
- The CVV response data element is: **<CVVResult>M</CVVResult>**

The CVV request can be sent as part of a *Sale* or *PreAuth* request, in which case the results are returned along with the Issuer authorization. It can also be sent using *ZeroAuth*, in which case just the CVV result is returned.

Below are "rules of thumb" to follow for the CVV Code received in the transaction response:

CVV Code	Definition	Explanation	Recommended Action	
			If sent with <i>Sale</i> or <i>PreAuth</i>	If sent with <i>ZeroAuth</i>
M	Match	The card security code supplied matches the value on the card.	Complete transaction	Complete transaction

CVV Code	Definition	Explanation	Recommended Action	
			If sent with <i>Sale</i> or <i>PreAuth</i>	If sent with <i>ZeroAuth</i>
N	No Match	The card security code supplied does not match the value on the card.	Reverse/Void and retry with updated information	Evaluate and retry with updated information
P	Not Processed	The request was not processed.	Reverse/Void the transaction and then Retry	Evaluate and retry, ask for another form of payment, or complete the transaction
U	Not Supported	The verification service is unavailable or issuer does not support this service.	Evaluate, then Reverse/Void and ask for another form of payment or Complete transaction	Evaluate and retry, ask for another form of payment, or complete the transaction

Note To avoid phishing threats for CVV data it is recommended to allow no more than three attempts that return a non-matching response.

Address Verification Service (AVS)

Address Verification (AVS) results codes are determined by the card's BILLING five or nine -digit postal code/ZIP and Street Address.

- The address verification service compares billing address information with what is on record at the issuing bank.
- The *AVSResult* Code is returned to the POS in the transaction response but is independent of the transaction authorization.
- Use of AVS impacts processing rates and aids in the prevention of chargeback retrieval requests.
- AVS is highly recommended to be implemented any time there is a business need for manual/keyed input in card present merchant locations and especially in card-not-present (MOTO and eCom) environments.

The AVS request data elements include *Address* and *Zip*. Example:

```
<AVS>
  <Address>150 Mercury Village</Address>
  <Zip>81301</Zip>
</AVS>
```

Format: *Address*: Alphanumeric. Length: min 1 max 20 *Zip*: for the U.S: numeric, for Canada: alphanumeric. Length: min 5 max 9

For the greatest accuracy and protection, send both the *Address* and *Zip*. However, in merchant card-present environments that do not ship merchandise the general rule of thumb is to always send the *Zip*, and then if required, the *Address*. In either case, never send just the *Address*.

- **For better rates with Visa and MasterCard**, submit AVS with at least the zip code.
- **For better rates with Discover and Amex**, submit AVS with BOTH street address and zip code.

Note Addresses that start with alpha characters, such as "PO Box", or that include "Suite" or "Apt No" may or may not be allowable, depending on the Issuer's records.

The AVS request can be made using either *ZeroAuth* (where no purchase information is included) or using *Sale* or *PreAuth* (as part of the full transaction request). If it is sent as part of a *Sale* or *PreAuth*, the result returned is independently of the authorization, just as it is with CVV.

- It is possible for the transaction to be approved and at the same time respond with a "No Match" AVS response.
- **A transaction will NEVER decline based on an AVS mismatch.** Therefore, it is critical for the local POS to be able to interpret the response codes and enable the merchant to respond to mismatched or questionable AVS results.

Rules of Thumb: The general "rules of thumb" actions below are based on two levels of matching: postal code/ZIP + address match exactly, or there is a partial match of postal code/ZIP or address.

- For card present environments, at a minimum, postal code/ZIP is recommended unless the business need involves off-premise delivery or shipping of merchandise.
- For all card not present environments, BOTH postal code/ZIP and ADDRESS are necessary for chargeback protection.

Below are common "rules of thumb" to follow for the AVS Code received in the transaction response:

AVS Code	Definition	Result/System Message	Recommended Actions
Y	Match	Street address and ZIP Code match	Exact match: complete transaction.
N	No Matches	Street address and ZIP Code do not match	No Match: Reverse and process a second authorization with updated information; do not allow multiple "phishing" attempts—use max of three tries.
Z	Zip Match	ZIP Code matches, but Street address does not match	Partial match: Evaluate based on business criteria; Reverse and retry with updated information or complete transaction
A	Address Match	Street address matches, but ZIP Code does not match	Partial match: Evaluate based on business, confirm cardholder ID; Reverse and retry with updated information or complete transaction
R	Retry	Retry / system unavailable	Issuer system message. Reverse the transaction and then Retry; if a second Retry is received, reverse and consider a second form of payment
U	Unavailable	Street and ZIP Code information unavailable or unverifiable	System message indicating AVS data is unavailable or un-verifiable. Evaluate based on business criteria; Reverse, check cardholder ID, ask for another form of payment or complete transaction

Industry AVS Response codes and descriptions vary slightly by card brand.

For a comprehensive list of domestic and international Address Verification Codes, refer to each card brand's acceptance guidelines.

Card Brand	AVS Code	Meaning
American Express	Y	Address and zip both match (5 or 9-digit zip)
	A	Address only matches (5 or 9-digit zip)
	Z	Zip only matches (5 or 9-digit zip)
	N	Neither address nor zip match (5 or 9-digit zip)
	U	Information unavailable
Discover Card	R	Retry, system unavailable
	Y	Address and zip match (5 or 9-digit zip)
	A	Address matches, zip code does not (5 or 9-digit zip)
	Z	Zip code matches, address does not (5-digit zip)
	N	Neither address nor zip match (5 or 9-digit zip)
MasterCard	U	System unavailable, retry
	Y	Address and zip both match (5 or 9-digit zip)
	A	Address only matches (5 or 9-digit zip)
	Z	Zip only matches (5 or 9-digit zip)
	N	Neither address nor zip match
	E	AVS not supported; system unavailable
	R	Retry
Visa	U	Information unavailable; no data from issuer
	Y	Address and zip both match (5 or 9-digit zip)

A	Address matches ZIP does not (5 or 9-digit zip)
Z	Zip only matches or Address not included (5 or 9-digit zip)
N	Neither address nor zip match (5 or 9-digit zip)
R	Retry, system unavailable
U or S	System unavailable or issuer is not an AVS participant
Z or P	Postal code matches, address not verifiable or Incompatible format
D	International Address and postal code match
Y	Address and postal/ZIP code match
C	Domestic, International not verified due to incompatible formats

Appendix 10 – Contactless Payments

MercuryPay supports magnetic-stripe contactless payments from a variety of different payment engines.

- **Credit cards** enabled for contactless payments contain both a regular magnetic stripe as well as a chip enabled for Near Field Communication (**NFC**).
- **Mobile wallets** enabled for contactless payments, such as Applepay, also use NFC technology, however the original track data is never stored on the mobile device.

Both NFC-enabled credit cards and mobile wallets work with a NFC-enabled peripheral device.

If a merchant accepts contactless payments using an NFC-enabled peripheral, the card, fob or wallet is “tapped” on the reader instead of swiped. The track data is in the same format as a magnetic swipe. For mobile wallets, the customer imports the card(s) to their wallet.

NFC contactless devices are available from most major device manufacturers including Ingenico, Equinox, IDTech and VeriFone.

Confirmation of Contactless Capable MerchantID

Developers certify their system’s POS entry capability for supporting contactless transactions, regardless of whether all transactions are contactless, magnetically swiped or manually entered. MercuryPay automatically flags all card present merchant accounts as “contactless capable”. This flag does not impact swiped or key-entered transactions, but must be in place when the source of the card data entry is via a contactless device. If a system attempts to send contactless transactions through a *MerchantID* which does not have the “contactless capable” flag enabled, an error of “INVALID DATA” will be returned in the *TextResponse* sent to the POS.

Transaction XML requirements

In order to support contactless payments, the POS developer utilizes an additional Account level key value pair in the payload request:

```
<AccountSource>Contactless</AccountSource>
```

If a POS system does not use MercuryPay’s P2PE, the key value pair <AccountSource> is only required when a contactless payment is accepted. MercuryPay automatically changes the entry method for swiped or manual transactions based on <Track2>, <Track1> or <AcctNo> and <ExpDate>. Contactless track 2 is not an ISO format; therefore MercuryPay cannot determine the entry method correctly without the proper AccountSource identification. MercuryPay also pulls the “Contactless” tag to our merchant reporting to help with reconciliation.

XML Example Request: Contactless non-encrypted Credit Sale transaction Request

```
<Account>
  <AccountSource>Contactless</AccountSource>
  <Track2>549999XXXXXX6781=0908101543219871234</Track2>
</Account>
```

XML Example Request: Contactless, encrypted Credit Sale transaction

Note: E2E capable POS systems require AccountSource for each entry method (Swiped, Keyed or Contactless).

```
<Account>
  <EncryptedFormat>MagneSafe</EncryptedFormat>
  <AccountSource>Contactless</AccountSource>
  <EncryptedBlock>F40DDBA1F645CC8DB85A6459D45AFF8002C244A0F74402B479
ABC9915EC9567C81BE99CE4483AF3D</EncryptedBlock>
  <EncryptedKey>9012090B01C4F200002B</EncryptedKey>
</Account>
```

NFC integration options

1. Datacap System Inc. dsiPDCX, dsiEMVUS: The Datacap dsiPDCX and EMVUS client controls supports multiple devices with NFC. The client control handles all device communication and builds the 'Contactless' AccountSource.
2. Direct integration to a contactless supported peripheral: Work with device manufacturers' SDK for direct device integration to a PINpad that supports NFC.
3. Integrate directly to a Contactless Reader: Work with the device manufacturer for direct integration to a contactless reader that supports NFC.
4. Blackline with Dejavoo terminal direct to web services
5. Stand-Alone Terminal with NFC enabled

Appendix 11 – Corporate Card Support on MercuryPay

Level II data – Tax and CustomerCode

For the lowest rates when processing business/commercial (corporate) cards, it is necessary to supply **Tax** (as a sub-field in the *Amount* field) and **CustomerCode** (in the *TranInfo* field) with all credit transactions. The use of this data lowers rates for retail and supermarket merchants, but not restaurants.

- **Tax:** Put the `<Tax>9.99</Tax>` tag in the *Amount* section of the XML request. The value format is numeric with 2 decimal places. Since the POS already calculates the correct tax amount for each transaction, simply parse the actual tax amount from the POS into the Tax tag. The Purchase amount tag will be the total charge, including tax. The tax field breaks out the tax amount for Level II reporting purposes only.
- **CustomerCode:** Submit `<CustomerCode>99999</CustomerCode>` in the *TranInfo* section. The value should be the actual PO number, the merchant's zip code, or the transaction invoice number. This is sufficient to meet the Level II reporting requirements.

Note: Corporate cards do not support recurring billing. This is dependent on the company's policies and those of the issuing bank. Level III folio data is not supported on the MercuryPay platform.

Appendix 12 – EBT and FSA on MercuryPay

Electronic Benefits (EBT): Food Stamps and Cash Assistance Benefits

Electronic Benefits are State-run government programs for electronically issuing food and cash benefits. **EBT** (Electronic Benefits Transfer) cards are government-encoded, stored-value cards:

- EBT Cards are loaded monthly,
- EBT Cards do not use an expiration date
- EBT Cards have limited use based on item eligibility.

Recipients use an EBT card to make purchases from participating merchants. Some merchants process only food benefit transactions, some only accept cash benefit transactions, and some accept both.

- **Food benefits** are federally authorized benefits that can be used only to purchase eligible food items and non-alcoholic beverages. Food benefits are distributed through the Supplemental Nutrition Assistance Program (**SNAP**), formerly known as the Food Stamp program. FNS oversees the regulations for item eligibility and licenses qualified merchants to accept SNAP benefits. More information is available here: <http://www.fns.usda.gov/snap>

Note: for details on supporting eWIC, please contact your implementation consultant for a comprehensive guide.

Cash Benefits may be used to purchase any item at participating retailers, obtain cash back, and make a cash withdrawal from participating ATMs. EBT Cash cards are similar to Debit cards. Merchants do not need to register with a state agency to accept EBT cash transactions. For this reason, we recommend that all systems integrated to PIN Debit also support EBT cash transactions.

PIN and Manual Entry

All EBT cards require cardholder PIN entry. However, unlike PIN Debit, Food Stamp *Sale*, Food Stamp *Return* and EBT Cash *Sale* transactions can be manually entered if necessary. Expiration date and security verification data are not required. Industry statistics estimate that approximately 30% of EBT card transactions are manually entered, making it important to support this from a business perspective. Whether swiped or manually entered, customers must still enter their PIN to complete the transaction.

Non-ISO Format

EBT cards are not required to follow the financial card ISO formats that are expected for credit and debit cards. The physical EBT cards are State-issued, and as a result, Track1 data may differ in length and encoding formats. However, Track2 data is available and should always be used when sending EBT transactions.

Supporting EBT Transactions

POS systems must be able to do the following:

- Distinguish between EBT eligible items from ineligible items
- Keep eligible items separated of flagged
- Not charge state or local tax on the EBT eligible items
- Accept both swiped and manually entered EBT cards
- Print the balance from the response on the customer paper receipt

Supported EBT Transactions

EBT is a separate TransactionType with several different TranCodes. EBT transaction requests require an additional parameter called **CardType**. This is used to indicate whether the card being used is “food stamp” or “cash”.

CardType	TransactionType	Remarks
Food Stamp	Sale	swiped and keyed food stamp sale; cash back not allowed
	Return	swiped and keyed food stamp return

	Voucher Sale	keyed only
	Balance	swiped and keyed food stamp balance
Cash	CashSale	swiped and keyed; cash back allowed when <i>MaxEBTCashBack</i> >0
	CashBalance	swiped and keyed

Notes: EBT *CashReturn* is an unsupported *TransactionType*.

If supporting cash benefit transactions, EBT *Cash Sale* is minimally required for certification.

If supporting food stamp transactions, the following transactions are required for certification:

- Food Stamp Sale,
- Food Stamp Return, and
- Food Stamp Voucher Sale

Food Stamp Voucher Sale is used if there is a communications failure. It allows the merchant to obtain a voice authorization to complete a food stamp sale. Upon successful registration with their state FNS, merchants receive a booklet of numbered food stamp vouchers.

To perform a voucher sale, the merchant first manually records the information in the voucher booklet, notes the voucher number, then calls the designated state authorization agency to receive an automated authorization code.

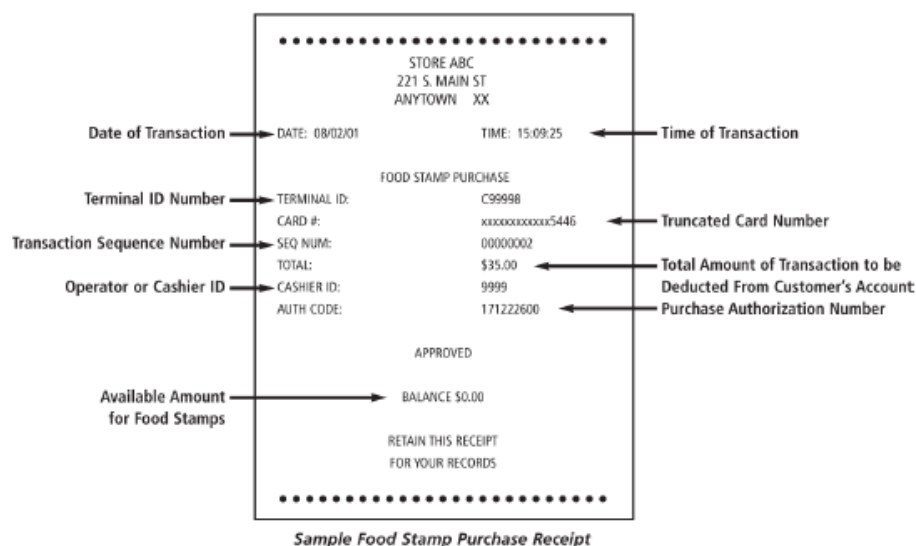
To get paid on the authorization, merchants must submit the transaction on their POS, including all required data:

1. card number,
2. authorization code, and
3. Voucher number from the printed voucher.

This is the only EBT transaction where a PIN is not required.

EBT Balances and Printed Receipts

it is important to support **Food Stamp Balance** and **Cash Balance** as separate transaction requests. Balance amounts are returned by the processor on all EBT transactions and should be printed on all EBT receipts, whether approved or declined.



EBT Request and Response XML Examples

Foodstamp Voucher Sale

Request

```

<?xml version="1.0"?>
<TStream>
<Transaction>
<MerchantID>112438931977591</MerchantID>
<LaneID>02</LaneID>      <!-- needed in multi-lane environments-->
<TranType>EBT</TranType>
<CardType>Foodstamp</CardType><!-- CardType can be FoodStamp or Cash -->
<TranCode>Voucher</TranCode>
<InvoiceNo>1</InvoiceNo>
<RefNo>1</RefNo>
<Memo>MPS Example XML v1.0</Memo>
<Account>
<AcctNo>40030XXXXXX6781</AcctNo>
</Account>
<Amount>
<Purchase>1.00</Purchase>
</Amount>
<TranInfo>
<AuthCode>123456</AuthCode><!-- Notice the Authcode and Voucher number -->
<VoucherNo>14</VoucherNo>
</TranInfo>
</Transaction>
</TStream>

```

Response

```

<?xml version="1.0">
<RStream>
<CmdResponse>
<ResponseOrigin>Processor</ResponseOrigin>
<DSIXReturnCode>000000</DSIXReturnCode>
<CmdStatus>Approved</CmdStatus>
<TextResponse>APPROVED</TextResponse>
<UserTraceData></UserTraceData>
</CmdResponse>
<TranResponse>
<MerchantID>112438931977591</MerchantID>
<AcctNo>400300XXXXXX6781</AcctNo>
<CardType>Foodstamp</CardType>
<TranCode>Voucher</TranCode>
<AuthCode>123456</AuthCode>
<VoucherNo>14</VoucherNo>
<CaptureStatus>Captured</CaptureStatus>
<RefNo>0080</RefNo>
<InvoiceNo>1</InvoiceNo>
<OperatorID>MPS Operator</OperatorID>
<Memo>MPS Example XML v1.0</Memo>
<Amount>
<Purchase>1.00</Purchase>
<Authorize>1.00</Authorize>
<Balance>15.38</Balance>      <!-- notice balance is returned in response -->
</Amount>
<AcqRefData>K</AcqRefData>
</TranResponse>
</RStream>

```


Food Stamp Sale - Swiped**Request**

```

<?xml version="1.0"?>
<TStream>
<Transaction>
<MerchantID>112438931977591</MerchantID>
<LaneID>02</LaneID>      <!-- only needed in multi-lane environments -->
<TranType>EBT</TranType>
<CardType>Foodstamp</CardType>      <!-- Note CardType is Foodstamp -->
<TranCode>Sale</TranCode>
<InvoiceNo>1</InvoiceNo>
<RefNo>1</RefNo>
<Memo>MPS Example XML v1.0</Memo>
<Account>
<Track2>400300XXXXXX6781=09085025432198712345</Track2>
<Name>MPS TEST</Name>
</Account>
<Amount>
<Purchase>1.00</Purchase>
</Amount>
<PIN>      <!--PIN is always required-->
<PINBlock> BAE0298BAE74CCEE</PINBlock>
<DervdKey> 3D01000042E00003</DervdKey>
</PIN>
<TerminalName>MPS Terminal</TerminalName>
<ShiftID>MPS Shift</ShiftID>
<OperatorID>MPS Operator</OperatorID>
</Transaction>
</TStream>

```

Response

```

<?xml version="1.0">
<RStream>
<CmdResponse>
<ResponseOrigin>Processor</ResponseOrigin>
<DSIXReturnCode>000000</DSIXReturnCode>
<CmdStatus>Approved</CmdStatus>
<TextResponse>AP*</TextResponse>
<UserTraceData></UserTraceData>
</CmdResponse>
<TranResponse>
<MerchantID>112438931977591</MerchantID>
<AcctNo>400300XXXXXX6781</AcctNo>
<ExpDate>0513</ExpDate>
<CardType>Foodstamp</CardType>
<TranCode>Sale</TranCode>
<CaptureStatus>Captured</CaptureStatus>
<RefNo>0080</RefNo>
<InvoiceNo>15</InvoiceNo>
<OperatorID>MPS Operator</OperatorID>
<Memo>MPS Example XML v1.0</Memo>
<Amount>
<Purchase>1.00</Purchase>
<Authorize>1.00</Authorize>
</Amount>
</TranResponse>
</RStream>

```

EBT Cash Sale (Keyed)**Request**

```

<?xml version="1.0"?>
<TStream>
<Transaction>
<MerchantID>112438931977591</MerchantID>
<LaneID>02</LaneID>      <!--only needed for multi-lane environments-->
<TranType>EBT</TranType>
<CardType>Cash</CardType>      <!--note CardType is Cash-->
<TranCode>Sale</TranCode>
<InvoiceNo>1</InvoiceNo>
<RefNo>1</RefNo>
<Memo>MPS Example XML v1.0</Memo>
<Account>
<AcctNo>400300XXXXXX6781</AcctNo>
</Account>
<Amount>
<Purchase>1.00</Purchase>
<CashBack>0.50</CashBack>
</Amount>
<PIN>      <!--PIN is always required-->
<PINBlock>3AB02859D999C774</PINBlock>
<DervdKey>12899900278000CC</DervdKey>
</PIN>
<TerminalName>MPS Terminal</TerminalName>
<ShiftID>MPS Shift</ShiftID>
<OperatorID>MPS Operator</OperatorID>
</Transaction>
</TStream>

```

Response

```

<?xml version="1.0">
<RStream>
<CmdResponse>
<ResponseOrigin>Processor</ResponseOrigin>
<DSIXReturnCode>000000</DSIXReturnCode>
<CmdStatus>Approved</CmdStatus>
<TextResponse>AP*</TextResponse>
<UserTraceData></UserTraceData>
</CmdResponse>
<TranResponse>
<MerchantID>112438931977591</MerchantID>
<AcctNo>400300XXXXXX6781</AcctNo>
<CardType>Cash</CardType>
<TranCode>Sale</TranCode>
<CaptureStatus>Captured</CaptureStatus>
<RefNo>0080</RefNo>
<InvoiceNo>20</InvoiceNo>
<OperatorID>MPS Operator</OperatorID>
<Memo>MPS Example XML v1.0</Memo>
<Amount>
<Purchase>1.00</Purchase>
<Authorize>1.00</Authorize>
</Amount>
<AcqRefData>K</AcqRefData>
</TranResponse>
</RStream>

```

Flexible Spending Account (FSA)

A Flexible Spending Account (**FSA**) is a tax-favored program offered by employers that allows their employees to pay for eligible out-of-pocket health care and dependent care expenses with pre-tax dollars. FSA cards are annually loaded, “stored value” cards that are only good for their available balance.

MercuryPay supports two necessary components of the FSA transaction approval cycle:

- FSA transactions
- An FSA Card Lookup database used to identify valid FSA BIN ranges

In addition to the MercuryPay certification, developers will need to independently register with **SIGIS**, an organization responsible for promoting an industry standard to meet IRS requirements for operating an Inventory **Information Approval System (IIAS)**. The IIAS database is used to determine which items qualify for FSA at the point of purchase. Worldpay IP recommends that developers interested in supporting FSA begin the SIGIS registration process immediately upon starting their FSA integration.

Note: FSA Certification and SIGIS registration are **not required** for the following Merchant Category Codes (MCCs) that are approved to accept FSA based on IRS regulations:

5047 Dental/Lab/Medical/Ophthalmic Hospital Equip & Supplies
 5975 Hearing Aids-Sales, Service, Supply Stores
 5976 Orthopedic Goods-Artificial Limb Stores
 7277 Debt, Marriage, Personal-Counseling Service*
 8011 Doctors-not elsewhere classified
 8021 Dentists, Orthodontists
 8031 Osteopathic Physicians
 8041 Chiropractors
 8042 Optometrists, Ophthalmologists
 8043 Opticians, Optical Goods, and Eyeglasses
 8049 Chiroprodists, Podiatrists
 8050 Nursing and Personal Care Facilities ("Dependent Care FSA")
 8062 Hospitals
 8071 Dental and Medical Laboratories
 8099 Health Practitioners, Medical Services-not elsewhere classified
 8351 Child Care Services ("Dependent Care FSA")

*Individual counselling is eligible, but Debt and Marriage Counselling are not FSA-eligible expenses

FSA certification and SIGIS qualification **is required** for merchant environments that fall under the SIGIS 90% rule or who use a mixture of IIAS FSA eligible and non-eligible goods.

Required FSA Transactions

There are three FSA specific transactions required for certification:

1. **CardLookup.** Used to determine that the card is an FSA card. Only legitimate FSA cards can be used for eligible FSA transactions. To determine this, developers must first send a *CardLookup* that returns a Card Usage of “FSA.” “DEBIT” and “OTHER” are the alternate usage responses. If “FSA” is not returned in the response, the transaction may still be completed as a standard credit or debit sale.
2. **FSASale.** This is the standard authorization request when using an FSA card. *FSASale* includes support for partial payments based on the available balance on the card. If the purchase amount exceeds the available balance, the card issuer will return **PARTIAL AP** in the response indicating that the transaction was only partially approved. The POS system should include “balance due” logic. More details are provided below.

The **FSA Prescription** subfield must be included in the *Amount* field of all *FSASale* transactions. The *Purchase* amount includes the prescription amount and the *FSAPrescription* breaks out the prescription amount separately:

<Amount>

```
<Purchase>6.00</Purchase>
<FSAPrescription>4.00</FSAPrescription>
</Amount>
```

3. **ReverseFSASale.** Reverses the sale and returns the funds to the cardholder's available balance within 2-72 hours. Always send *AcctNo* and *ExpDate* in the *Account* field. If the *ReverseFSASale* fails, send a *VoidSale*. POS developers are required to include void functionality in case the *ReverseFSASale* fails or declines.

Partial Approval, "Balance Due" and Reversal Support

SIGIS regulations stipulate that FSA cards must be usable down to a 0.00 balance. This is why partial approvals must be built into the *FSASale* transactions. POS systems must be able to detect a partial approval and prompt the cardholder for a "balance due" for the remaining purchase amount.

- In the event of a partial approval, the POS will receive a *TextResponse* of **PARTIAL AP** and the *Authorized* amount will be less than the originally requested *Purchase* amount. Developers should use both the *TextResponse* and an amount comparison when building the "balance due" message.
- If the cardholder cannot complete the purchase with another form of payment, SIGIS regulations require that the POS attempt to reverse the transaction using *ReverseFSASale*. If the *ReverseFSASale* fails, declines or is rejected, the POS must void the *FSASale* using a standard *VoidSale*.

FSA XML Examples

Card Lookup

Request

```
<?xml version="1.0"?>
<TStream>
  <Transaction>
    <MerchantID>112438931977591</MerchantID>
    <TranType>CardLookup</TranType> <!--Note there is no TranCode used in this request-->
    <Account>
      <Track2>400555XXXXXX0480=12055025432198712345</Track2>
    </Account>
  </Transaction>
</TStream>
```

The *CardLookup* response receives the confirmation from MercuryPay's BIN tables of FSA usage:

Response

```
<?xml version="1.0"?>
<RStream>
  <CmdResponse>
    <ResponseOrigin>Server</ResponseOrigin>
    <DSIXReturnCode>000000</DSIXReturnCode>
    <CmdStatus>Approved</CmdStatus>
    <TextResponse>SUCCESSFUL LOOKUP</TextResponse>
    <UserTraceData></UserTraceData>
  </CmdResponse>
  <TranResponse>
    <MerchantID>112438931977591</MerchantID>
    <AcctNo>400555XXXXXX0480</AcctNo>
    <ExpDate>0512</ExpDate>
    <CardUsage>FSA</CardUsage>
    <CardType>VISA</CardType>
    <TranType>CardLookup</TranType>
    <TranCode>CardLookup</TranCode>
  </TranResponse>
</RStream> <!--Note: if requested with a Token Frequency and RecordNumberRequested, MercuryPay would return a Token record number as well-->
```


Appendix 13 – Card Security, PCI and Regulations Overview

Note: The information supplied in this appendix is not comprehensive or intended to act as a substitute for a developer's need to thoroughly understand the payment industry, state or federal regulations.

It has become increasingly important for all participants in the payments industry to become vigilant in understanding and complying with all card brand, State and Federal industry regulations, as well as Payment Application Card Data Security (PA-DSS) requirements. Before a POS developer embarks on the process of building a POS, it is **absolutely critical** to have a solid working knowledge of why these regulations are in place and how complying with these regulations helps to mitigate potential payment processing risks and vulnerabilities.

The purpose of this overview is to emphasize the critical nature of these regulations: Merchants are increasingly vulnerable to fraud, malware and being hacked. **It is the responsibility of the POS integrator to understand and comply with all industry laws, regulations and card data security standards.** The entire industry is working on tightening unwarranted and criminal access to unencrypted card data, clear text files, identity information, open WIFI networks, un-segmented internet networks, and all manner of stored data files. Payment applications that expose any of these vulnerabilities are unacceptable by the current industry standards.

Developers who wish to board merchants and process with MercuryPay are required to either:

- confirm validation with a certified Payment Application Qualified Security Assessor (PA-QSA),
- use an existing validated, compliant application, such as HostedCheckout or dsipDCX, or
- Demonstrate a removal-of-scope compliance path by using a combination of Point-to-Point Encryption (P2PE/E2E) and Tokenization.

There are multiple governing bodies overseeing the payments industry regulations:

- the PCI Security Standards Council (PCI SSC),
- State and Federal legislation, and
- the card brands themselves (Visa, MasterCard, Discover and American Express).

Note! It is important to review, understand and comply with these regulations as they apply to your payment application and as your POS system is implemented with your merchant base.

Payment Card Industry Security Standards Council

The **PCI Security Standards Council (PCI SSC)** was established in 2006 by five card brands: American Express, Discover, JCB, MasterCard and Visa. Its purpose is to oversee and implement security standards for merchants, developers and service providers. The goal of these security standards is to help ensure that payment processing is secure and protects sensitive cardholder data.

While these standards are maintained by the PCI SSC, it is up to each card brand such as Visa or MasterCard to enforce their use. Card brands require adherence to the PCI standards for anyone who stores, processes, or transmits cardholder data. Failure to comply can result in fines, especially in the event that card data is compromised or stolen.

Note that PA-DSS and PCI DSS are *not* best practices; they are the *minimum standards*. If an application is validated, it doesn't mean it's hacker-proof. Similarly, merchants are not automatically PCI DSS compliant if they are using PA-DSS compliant software.

PCI SSC Governed Card Data Security Standards

There are three broad categories of standards – one for merchants/service providers, one for developers' software applications, and the third for payment devices. The PCI Council released version 3.2 of these standards in June 2016.

PCI DSS (Payment Card Industry Data Security Standard) There are 12 standards with over 280 requirements for *any* entities that store, process or transmit cardholder data. This applies to both merchants and service providers. Not every requirement is applicable in all cases. All of the card brands have adopted these

standards and have incorporated the requirement to comply with them into their contractual agreements with acquiring and issuing financial institutions. They can be thought of as standards for an *organization*.

PA-DSS (Payment Application Data Security Standard) This outlines the rules and regulations governing the approval of security for payment application software such as point-of-sale (POS) systems. It contains 14 standards that cover personally identifiable cardholder data—the account number (PAN), expiration date, cardholder name and service code. These can be thought of as standards for an *application*.

An application is PA-DSS eligible if it stores, processes or transmits cardholder data as part of *authorization* or *settlement* AND is sold, distributed, or licensed to third parties. If a merchant writes their own software application for their own use, it is *not* PA-DSS eligible. However, the merchant is still required to follow PCI DSS. The requirements in PA-DSS align with all requirements in the PCI DSS.

PCI PTS (Payment Card Industry PIN Transaction Security) This standard complements the PCI DSS and outlines the rules and regulations governing the approval of security for PIN entry. The PCI Council maintains a matrix of PTS certification versions and corresponding dates governing their deployment and use.

PCI Point-to-Point Encryption (P2PE) This standard applies to encryption, decryption and key management standards for secure cryptographic devices (SCD). A solution is designated as PCI P2PE validated once it has undergone a rigorous review by a P2PE Assessor, and the assessment has been accepted by the PCI Council. Validated PCI P2PE solutions are listed on the PCI website.

Qualified Integrators and Resellers (QIR)

QIR's are authorized by the PCI Council to install, configure, and support validated PA-DSS payment applications for merchants and service providers. This assures that the implementation of the payment application supports the customer's PCI DSS compliance. Resellers, developers and sole proprietors can become a certified QIR by completing a course on the PCI website.

United States Federal and State Laws

Receipt Printing

Each Card Brand has guidelines in their Merchant Operating Guides to help identify what data should be displayed on merchant and customer receipts. However, it is important to note that Federal and State legislation has been implemented to help reduce cardholder data being stolen from receipts. Important laws include:

The Fair and Accurate Credit Transaction Act of 2003 (FACTA)

This law pertains to the physical printing of cardholder data and dictates, among other things, that “no person that accepts credit cards or debit cards for the transaction of business shall print more than the last five digits of the card number or the expiration date upon any receipt provided to the cardholder at the point of the sale or transaction.” While the intent of this law is to mask card data printed on the customer copy, there have been cases of merchants being sued when the customer has walked away with the merchant copy.

It is important that State laws are also reviewed, as in for example:

- **Tennessee - Tennessee Consumer Protection Act:**
<http://tn.gov/consumer/documents/LexisNexis2010.pdf>
- **California – Song-Beverly Credit Card Act of 1971, section 1747.09:** <http://www.leginfo.ca.gov/cgi-bin/displaycode?section=civ&group=01001-02000&file=1747-1748.95>
- **The Dodd-Frank Wall Street Reform and Consumer Protection Act**
<http://www.gpo.gov/fdsys/pkg/PLAW-111publ203/pdf/PLAW-111publ203.pdf>
 - Merchants may now set a minimum charge for credit card purchases of up to 10 dollars.
 - The Federal Reserve can cap debit transaction fees that merchants pay to banks.
 - Debit card interchange rates are capped at **21 cents plus 0.05%** of the transaction, with the possibility of an additional cent if certain security criteria are met.
 - Merchants may choose the debit network that processes their transactions.

Gift Cash Out

Some States are now mandating a POS function to “cash out” prepaid cards when amounts are \$10.00 or less. Some example States are California and Massachusetts.

Card Brand Regulations

As one becomes more familiar with the regulations, the minor differences among the brands will become clearer. In some cases, the difference might simply be the fees assessed. The association's regulations can be enforced through fines or revoked privileges to accept a specific Card Brand. Generally, the Card Brands communicate to the Issuers and Acquiring Banks about continuous or serious violations. This section discusses the more frequent regulations referred to in our practice.

Card Brand Mandates – some examples

MasterCard LaneID

In an effort to combat and reduce fraud, MasterCard issued a compliance mandate that requires merchants using multiple card acceptor devices (i.e. workstations, lanes, or peripheral terminals) to pass a unique terminal ID with each transaction. Adding a unique identifier to each card acceptor device allows a transaction to be traced through the networks and back to the terminal from which it originated.

In response to this initiative, MercuryPay implemented a new data field called **LaneID**.

PrePaid Partial Authorization and Reversal Support

- **Partial Approvals** When the transaction amount exceeds the funds available on the card, merchants can conduct split-tender purchases by allowing debit card issuers (including prepaid) to approve a portion of the original transaction amount in the authorization request. The merchant can then initiate split-tender processing and obtain the remainder of the purchase amount from another form of payment.
- **Authorization Reversal** Authorization reversals free up debit card holders' open-to-buy amounts by reducing issuer holds on available balances when transactions are not completed. This reduces the number of declines at the POS and the number of cardholder complaints that are unpleasant for all parties involved.

Common Transaction Downgrades

A downgrade is when the Card Brands charge the merchant an extra transaction fee. Transaction downgrades can occur for many reasons. Both Visa and MasterCard assess fees to merchants when the authorization of cardholder data is abused or not performed correctly. Two examples of these downgrades are the **Misuse of Authorization** and **Zero Floor Limit** fees. Currently, these transaction fees are assessed at a minimum of three months past the original authorization date.

Misuse of Authorization Fee

The *Misuse of Authorization Fee* applies to authorizations that are not followed by a matching clearing transaction (or in the case of a cancelled or timed out authorization, not properly reversed.) It also applies to transactions where the clearing amount is less than the authorized amount (e.g. over-authorization.)

- The fee can be avoided by clearing (settling) your transactions within 10 days for non-Travel and Entertainment (T&E) Merchant segments, and 20 days for T&E Merchant.
- If an authorization is not needed, the authorization must be electronically reversed within 24 hours for face-to-face authorizations, and within 72 hours for card-absent authorizations.

Zero Floor Limit Fee

Visa's *Zero Floor Limit* applies to cleared transactions that can't be matched to a previously approved or partially-approved authorization. In short, it applies to settlement transactions submitted without a proper authorization.

Processing Integrity Fee

MasterCard combined both scenarios described above and imposes the *Processing Integrity Fee*. This fee applies in the following instances:

- **Card-present** Transactions are not settled, cleared, or reversed within 24 hours of the original authorization transaction/request.
- **Card-not-present** Transactions are not settled, cleared, or reversed within 72 hours of the original authorization transaction/request.
- **No reversal** An authorization transaction cannot be matched to a corresponding settlement record after a period of 120 days.

Storing, Displaying and Printing Card Data: Integration Best Practices

Storage and Display

- **The following items should never be stored:**
 - Full track data from any track
 - PIN/PIN Block
 - Security code printed on back of card (CVV2, CVC2, CID)
- **Unencrypted storage of the account number** is limited to the first six and last four digits of the card number. Everything in-between must be masked.
- **Whenever a card number is displayed**, it must be masked in the same fashion, first six and last four digits of the card number, unless there is a specific need to see the full card number.
Note Printing masked card data is more restrictive than storage or display. Do not print first six. Print last four only.
- **If a card number is stored unmasked**, it must be encrypted in accordance with the PCI-DSS. The PABP (Payment Application Best Practices) can be referenced as it's targeted to developers of POS systems.
- **When encrypting:** Approved standards are standardized algorithms (like in ISO and ANSI) and well-known commercially available standards (like Blowfish) that meet the intent of strong cryptography. Examples of approved standards are AES (128 bits and higher), TDES (two or three independent keys), RSA (1024 bits) and ElGamal (1024 bits)
- **If the account number is stored**, the cardholder name, and expiration date must also be encrypted. Worldpay IP recommends encrypting these data elements regardless in an effort to protect consumer privacy.
- **Sensitive, encrypted card data** can only be stored as long as there's a business reason. For most systems, it's best to delete the stored data after a successful batch close.
- **Systems that support "Offline" or "Store and Forward" functionality** are allowed *prior* to authorization, per the card associations and per PCI data storage requirements. That is, any **pending, unauthorized offline transaction** is considered "in-process" and may be stored. Additional considerations apply:
 - Transactions processed offline or held prior to authorization where a real-time authorization is not obtained at the time of sale carry significantly higher risk.
 - Underwriting and risk criteria must also be met including: business type, dollar amounts being processed, and a relatively short time (1-10 days) before the offline transactions are authorized and settled.
 - For large dollar transactions, offline-capable systems should still utilize the card associations' automated voice authorizations procedures.

Printing Cardholder Receipts

The Fair and Accurate Credit Transaction Act of 2003 (**FACTA**) was written with the intent of masking card data on the customer's physical copy of the receipt. There have been well-documented incidents of legal action that occurred as a result of this FACTA violation. The current industry standard is to only display the last 4 digits of the card number and to completely mask the expiration date on customer receipt copies:

- Print only the last four digits: **ACT # : *****5220**
- Print the expiration date masked:**EXP : XX/XX**

A complete example of a receipt can be found in most card brand regulations such as the Visa's *Card Acceptance Guide* and the *Chargeback Management Guidelines*. These guides can be downloaded from Visa's merchant site.

Printing Merchant Receipts

To protect consumers and to be safe from lawsuits, ***we recommend masking the card number and expiration date*** on the merchant receipt in the same way as the cardholder receipt. Alternatively, provide a configurable option in POS software to allow the card number and expiration date to be masked on the merchant copy. Make this the default option.

We anticipate that more state laws, federal law, and card brand rules will follow. In anticipation of additional legislation governing physical receipt requirements, MercuryPay recommends masking the merchant copy to avoid complications.

General Card Data Compliance Summary

Use of Track1 and Track2 Data

Track data is for immediate use in building authorization requests only, and *under no circumstances may either Track1 or Track2 data be stored for any length of time subsequent to authorization.*

Cardholder Verification Value

Cardholder Verification Value codes (i.e., CVV2, CVC2, CID) are the three- or four-digit numbers printed on the signature line on the back of a card. These are used for security purposes in transactions when a card cannot be swiped due to a damaged magnetic stripe, or when the card is not physically present (e.g. eCommerce). ***As in the case of track data, CVV is for immediate use in an authorization, and must not be stored for any length of time.***

PIN Data

PIN Verification Values (PVV), PIN Blocks, and derived encryption keys ***must not be stored for any length of time subsequent to authorization*** whether encrypted or not.

Cardholder Data Storage and Retention

Cardholder data is defined as any personally identifiable data associated with a cardholder. It includes account numbers, expiration dates, addresses, names, social security numbers, etc. When cardholder data is stored by the merchant for any length of time, proper security measures must be taken.

- ***If full card numbers are stored, they must be encrypted using a strong encryption method (at least 128-bit TDES, 256-bit AES, etc.)*** At a minimum, the account number must be encrypted, but it is advisable that all cardholder data be encrypted.
- For the purposes of performing tip adjustments, restaurants maintain full card numbers until the batch is closed. Log files must not contain credit card numbers—they should be truncated if they appear in log files.

Retail/QSR

Validated retail/QSR (quick-service restaurant) merchants who process Sale transactions may find it useful to store encrypted card data until the batch is closed. Since *Void*s and *Adjust*s can only be performed while the batch is open, there is no need to retain card data after the batch is closed. Tokenization may always be used as a secure alternative.

Restaurants

In order to perform *PreAuthCapture*, *Adjust* and *VoidSale*, table-service restaurants processing with validated systems will need to store full card numbers, expiration dates, and other transaction information until the batch is closed. ***At minimum, the card numbers must be encrypted using a strong encryption method during this time.*** Since *Adjust* and *VoidSale* transactions can only be performed while the batch is open, the card numbers

should be truncated or discarded entirely after the batch is closed. Tokenization may always be used as a secure alternative.

- **When displayed unencrypted, card numbers must be truncated, or masked, to show no more than the first six and last four digits of the number.** This does not apply to employees who need to see full numbers, so the option to view card numbers both unencrypted and unmasked can be a managerial feature. However, it is prudent not to allow easy bulk exports or bulk printing of unencrypted card numbers. Tokenization may always be used as a secure alternative.
- The perpetual storage of card data, even in encrypted form, puts the merchant at risk. According to the card associations, card data can only be stored as long as there is a ***legitimate business need***. Worldpay IP recommends that encrypted card data be deleted immediately upon batch close.
 - However, merchants who employ electronic signature capture and who do not have paper receipts may retain the electronic receipt records for two years. This is because merchants are required for legal reasons to securely store receipts (whether paper copy, or digital image including signature) for two years.
- As an alternative to encrypted storage of cardholder data, the card numbers can be truncated at the beginning of storage: no more than the first six and last four digits may be stored unencrypted.

Appendix 14 – eCommerce Environments

Ecommerce Solutions on MercuryPay

MercuryPay supports two paths to an eCommerce solution:

1. developers may develop their own solutions based on **Web Services**, or
2. Use our **HostedCheckout** solution.

eCommerce Website Requirements

Before a business can be approved to process eCommerce transactions, it must have a valid website with the requirements listed below. Should the products or services offered through an eCommerce merchant account change, MercuryPay must be notified immediately. A merchant that currently has a retail or MOTO account with MercuryPay must submit an entirely separate application for eCommerce. MercuryPay issues eCommerce merchants unique Merchant ID numbers upon underwriting approval.

The following eCommerce policies and procedures have been established to comply with Visa®, MasterCard®, Discover® and MercuryPay regulations:

1. **Product(s) Sold.** Ecommerce merchants must maintain a valid website with complete descriptions of the products and services sold. All federal and state laws apply in addition to the card association regulations regarding all products and services sold.
2. **Security Policy.** Ecommerce merchants must submit transactions for authorization and settlement through a secure online gateway. The security protocols used to protect a customer's information must be disclosed.
3. **Privacy & Return Policy.** The website must contain a return, refund and privacy policy. *Terms of Use* or *Terms and Conditions* must also be properly disclosed on the website.
4. **Delivery Policy.** Ecommerce merchants set their own policies and restrictions regarding delivery of goods. Any restrictions on delivery must be clearly stated on the website.
5. **Customer Service Contact.** Contact information for your business must be easily accessible to customers. It must show a physical address along with an electronic mail address and telephone number.
6. **Card Acceptance Brand Marks.** Full-color brand marks are required by Visa®, MasterCard® and Discover®. The American Express® logo must be displayed if accepted.

eCommerce Best Practices

1. **Shipping and Delivery.** The shipping method, time frame and delivery procedures should be clearly stated. After the payment has been accepted, the product must ship immediately. The customer cannot be charged unless the product is in inventory and ready to ship.
2. **Transaction Receipt.** A transaction receipt via electronic mail should be provided to the customer and must include your business name, web address, contact information and the terms and conditions of the sale. Always keep the transaction receipt, shipping record (UPS, Airborne, FedEx or USPS) and proof-of-delivery on file. This information will need to be presented in the event of a chargeback.
3. **Transaction Verification.** AVS (the Address Verification System) should be used to validate the billing address of the cardholder. Use the Card Verification Value (CVV2) to further validate the transaction.
4. **Suspicious Transactions.** It is highly recommended to develop in-house policies and procedures for handling irregular or suspicious transactions and provide appropriate training for sales staff. Any order that seems suspicious should be further investigated. Contact the cardholder for verification. Orders that fail AVS or CVV2 validation should be flagged for research. If an order is high-risk or possibly fraudulent, void the transaction immediately. Shipping orders that are not validated by AVS or CVV2 could result in chargebacks and loss of merchandise. All eCommerce merchants should be extremely suspicious of international orders. Some geographic areas have a higher frequency of fraud and black market stolen credit card information.

Additional policies and procedures should be put into place to monitor the following types of orders:

- Rush orders
- Random orders (orders not specifying color, size, etc.)
- Shipping address not the same as billing address
- Shipments to post office boxes or business addresses
- Transactions associated with anonymous email addresses
- Orders from internet addresses at free email services

5. **Acknowledgement of Policies.** Policies should require a cardholder to click to accept them. Not doing so leaves merchants open to claims by the cardholder that they were not aware of the merchant's policies at the time of the transaction.

Handling Errors in eCommerce Transactions

Categories of Error Handling

General Response Categories	TextResponse	Suggested Verbiage for Response	Remarks
<ul style="list-style-type: none"> • Invalid Credentials/MID • Merchant ID not Found Response due to configuration error 	INVLD MERCH ID, INVLD CREDENTIALS MERCHANT ID NOT FOUND, MerchantID not found (Gift) Invalid TranType for Prepaid Server	"Web Site Application Error. Please Contact ###-###-####"	Send corresponding email alert notification to Gateway admin/merchant.
Communication, Connectivity and Socket Response Errors	EDC UNAVAILABLE, DB UNAVAIL, TIMEOUT ON RESPONSE, TIMEOUT WAITING FOR RESPONSE, MAX CONNECTION ERROR, SOCKET CONNECTION FAILED, NO CONNECTION TO ANY SERVER, HOST UNAVAIL, ISSUER UNAVAIL	"We're sorry. We are experiencing a temporary connectivity error. Please try again."	
Transaction Level: Approvals	AP, AP*, AP NEW INFO APPROVED STANDIN AP-NOT CAPTURED	"Approved, Successful!"	
Transaction Level: Declines	DECLINED, OTHER NOT EXCEPTED, PIC UP, EXCEEDS MAX AMT, DECLINE TRY LATER	"We're sorry, your transaction has declined."	
Transaction Level Declines: Invalid Data Errors	INVLD ACCT, INVALID EX DATE, INVALID REFERENCE NUMBER, INVLD TRAN CODE, INVALID FIELD: _____.	"Your account information has been entered incorrectly. Please try again".	Maximum 3 retries.
Transaction Level Declines: Duplicate Transaction	(AP*- signifies duplicate logic detected duplicate and did not charge the card twice)	"Duplicate transaction data has been entered. Transaction declined."	

General Response Categories	TextResponse	Suggested Verbiage for Response	Remarks
Transaction Level Declines: Call Referrals	AP DUPE CALL AE, CALL DISCOVER, CALL ND, CALL XXXX	"Your transaction has declined: please contact us at ###-###-#### to complete your transaction."	
Verification Result codes: CVV	M=CVV Match N=CVV no Match P=Not Processed DECLINED-CV2 FAIL	"Your information has been entered incorrectly. Please verify and try again."	Maximum 3 retries, or Auto-void upon CVV fail response of N, or base void on established criteria as in >\$300.00 Note MercuryPay recommends coding provisions for Merchant level configurable attributes.
Verification Result codes: AVS	(AVS Varies by card brand, all single digit alpha characters) See AVS Response Codes for more details.	"Your information has been entered incorrectly. Please verify your retry."	Maximum 3 retries
Settlement Response Codes	OK, INV BAL/SETL, MUST BAL NOW		Merchant/Shopping Cart/Gateway level only

Error Handling by CmdStatus

- Every XML response includes a *CmdResponse* element with **ResponseOrigin**, **DSIXReturnCode**, **CmdStatus**, **TextResponse** and **UserTraceData**.
- Error handling in eCommerce environments can be triggered from four *CmdStatus* responses:
 - Approved**: transaction approved by payment processor
 - Declined**: transaction declined by payment processor
 - Success**: command completed successfully
 - Error**: error processing command. Check DSIXReturnCode and TextResponse for info on error

For this CmdStatus response	Return this message to the web application	Remarks
Approved	Transaction approved.	
Declined	Transaction declined.	
Declined and TextResponse includes Reason/Action of: DECLINED, OTHER NOT EXCEPTED, PIC UP, EXCEEDS MAX AMT, DECLINE TRY LATER	Transaction declined.	
Declined and TextResponse includes Reason/Action of: INVLD ACCT, INVLD EXP DATE, INVALID REFERENCE NUMBER, INVLD TRAN CODE, INVALID FIELD]	Your account information has been entered incorrectly. Please retry.	Maximum 3 retries
Declined and TextResponse includes Reason/Action of: CALL AE,	Transaction was not completed: please contact us at ###-###-#### to complete your order.	

For this CmdStatus response	Return this message to the web application	Remarks
CALL DISCOVER, CALL ND, CALL XXXX		
Success	Transaction completed successfully.	ADMIN response
Error	An error has occurred.	Log full response to merchant and Site Administrator

Appendix 15 – Electronic Signature Capture

Implementing an electronic signature capture device is a great way to enhance your merchant and customer experience. It is easy to dive in and get a quick idea of what is required from a coding perspective using our Github code repository link below.

This section provides you details about capturing, storing and regenerating signatures along with a summary of the specifications used in this process.

Overview

When integrating to a peripheral device that supports signature capture, the interaction with the device follows two distinct operations: (1) the transaction authorization, followed by (2) the signature request. Although the signature request is a separate event, it is dependent on an approved authorization response. When a signature is electronically captured for either EMV or standard magstripe credit card processing, that signature must be securely bound to the transaction record for which it was captured and must be able to be regenerated in full upon request.

- **Signature capture is typically required** for all credit transaction types and where signature is the cardholder verification method (CVM) unless the merchant qualifies for Quick Serve small-ticket Quick Payment Service (QPS). Signature capture IS NOT required for Prepaid/Gift, PIN based U.S. PIN Debit or EBT transactions. It is also not required for recurring transactions or transactions originating from card-on-file, although terms and conditions of recurring billing arrangement should be in place and agreed to by the cardholder.
- At the time of this writing, the card brands are updating their regulations to eliminate the need for signature collection on EMV enabled transactions. Having an EMV enabled device is key and will keep merchants the most protected. Signature will, in most cases, no longer be used as compelling evidence for disputes and its use will be at the discretion of the merchant. These regulations may vary by card brand. For example Visa will still be requiring signature on MSD contactless, swiped and keyed transactions if the merchant is enabled to only accept magnetic strip transactions

As with standard paper receipts, it is important to follow card-brand operational regulations and State or Federal laws when storing, printing and reproducing receipts that originate from an electronic signature capture device. As with standard paper receipts, the electronically captured signature indicates that the cardholder was present at the time of the transaction and agrees to payment. ***It is essential that the receipt and the associated signature be retrievable in the event of a chargeback dispute.*** It is important to verify that a signature was captured and that when it is regenerated, it clearly represents the original sales draft.

You should build in a provision to immediately prompt the sales clerk to reverse a transaction if a signature was *not* captured and the cardholder opts to cancel the operation, or if the signature request times out.

Merchants implementing electronic signature capture device should:

- Only store and reproduce a signature that is associated with the transaction for which the signature was obtained.
- Only reproduce a signature upon specific written request from the Acquirer or in response to a Retrieval Request (chargeback dispute).
- Have proper controls in place to securely store signatures and other cardholder data, in accordance with the Payment Card Industry Data Security Standard (PCI DSS).

GetSignature Specification

Element	Req	Min	Max	Description
GetSignature	O	1	2048	Digitized signature

dsiEMVUS/PDCX GetSignature Request XML

```
<?xml version="1.0"?>
<Tstream>
```

```
<Transaction>
<HostOrIP>127.0.0.1</HostOrIP>
<IpPort>9000</IpPort>
<MerchantID>755847007</MerchantID>
<OperatorID>User</OperatorID>
<UserTrace>Loaner</UserTrace>
  <TranCode>GetSignature</TranCode>
  <SecureDevice>EMV_ISC250_MERCURY</SecureDevice>
  <ComPort>9</ComPort>
  <SequenceNo>0010010010</SequenceNo>
</Transaction>
</Tstream>
```

dsiEMVUS/PDCX GetSignature Response Data

The signature response data is a string of up to 2048 characters returned as a series of "strokes".

- Within each stroke is a series of X,Y coordinate points.
- X and Y coordinates are separated by a comma and end with a colon.
- The first point of a stroke is always a "move to" position.
- All subsequent points within a stroke are "draw to" positions.
- The "end stroke point" is simply #,: which is followed by the first point of the next stroke.

Below is an example of a dsIEMVUS/PDCX *GetSignature* response with the X,Y coordinates. Note that the response includes the maximum height (*SignMaximumX*) and maximum width (*SignMaximumY*). The graphical representation of this signature data below is Joe Worldpay from FIS.

```
<?xml version="1.0"?>
<RStream>
    <CmdResponse>
        <ResponseOrigin></ResponseOrigin>
        <DSIReturnCode>000000</DSIReturnCode>
        <CmdStatus>Success</CmdStatus>
        <TextResponse>Success</TextResponse>
        <SequenceNo>0010010010</SequenceNo>
        <UserTrace>Loaner</UserTrace>
    </CmdResponse>

    <Signature>50,40:48,41:45,42:43,43:42,44:40,45:38,46:36,47:34,48:32,49:30,49:28,50:23,50:21,49:19,49:17,48:16,47:14,45:13,44:12,42:11,40:10,37:10,31:21,31:22,29:14,27:17,28:16,29:17,27:22,28:23,29:25,30:26,31:28,33:29,35:30,37:32,39:33,42:34,43:34,45:32,56:36,68:35,73:35,77:34,81:33,85:#=10,85:11,80:13,76:15,72:17,69:19,66:22,63:24,62:26,59:29,58:31,57:34,56:38,56:40,57:42,57:44,58:45,59:47,60:48,61:49,62:50,64:51,66:#=70,37:69,38:68,40:67,42:66,44:66,45:65,47:65,55:66,57:68,58:70,57:70,49:69,48:68,46:67,45:66,44:65,43:63,43:64,45:65,46:69,46:71,45:73,45:74,44:76,43:78,42,40,41:81,40:83,39:84,38:85,37:86,36:87,35:87,34:86,35:85,36:84,38:83,40:81,42:80,44:79,46:78,48:77,50:77,56:78,57:79,59:80,60:82,61:84,62:86,63:93,63:#=#=129,35:128,34:128,31:127,30:127,23:128,22:129,21:130,22:131,23:132,25:133,27:133,29:134,31:134,33:135,36:135,56:134,58:134,64:135,62:136,61:137,59:138,57:139,54:140,51:142,48:143,45:145,41:147,37:149,33:151,30:152,27:154,24:155,22:157,20:158,18:159,17:161,16:162,18:#=#=168,40:166,41:165,42:164,43:163,44:161,45:160,47:159,49:157,50:156,52:155,54:154,55:153,57:152,58:151,59:151,62:153,61:155,60:156,59:157,58:159,57:160,55:162,54:163,52:164,51:166,49:167,48:168,46:169,44:170,43:171,42:170,43:169,45:168,46:167,47:166,48:165,50:165,51:164,53:163,55:163,57:162,59:163,61:163,63:167,63:168,62:170,61:171,59:173,57:174,56:175,55:176,53:177,51:178,50:179,48:180,46:179,47:179,51:180,52:180,54:181,55:182,56:183,57:187,57:188,55:190,54:191,53:192,53:191,51:195,50:194,50:198,51:199,53:200,54:201,56:204,56:205,54:#=#=229,10:229,12:228,13:228,15:227,17:227,18:226,20:222,5:23,22:25,24:223,23:223,32:222,32:222,37:224,40:221,44:220,46:220,52:218,55:#=#=200,42:202,41:204,40:206,39:208,38:210,38:213,37:215,36:218,35:221,35:224,34:227,33:229,33:232,32:242,32:#=#=248,35:247,37:246,39:245,41:244,43:244,45:243,46:242,47:242,50:241,51:241,55:243,54:244,53:245,52:246,51:247,49:248,48:249,46:251,45:252,43:253,42:254,41:255,39:256,38:257,37:257,39:256,40:256,42:255,44:254,46:254,54:255,56:256,57:257,58:25,59:264,58:266,57:268,56:270,55:271,54:272,52:274,51:275,50:276,49:277,48:278,47:279,46:280,45:281,44:282,43:284,42:284,40:285,38:#=#=257,10:255,10:254,10:#=#</Signature>
    <SignMaximumX>295</SignMaximumX>
    <SignMaximumY>95</SignMaximumY>
</RStream>
```

GetSignature code on Github

A Worldpay from FIS IP/MercuryPay Github Code repository is available to assist your development efforts. Use the link below to download, compile and run this code sample:

<https://github.com/MercuryPay/dsiEMVX.CSharp/blob/master/dsiEMVX.CSharp/dsiEMVX.CSharp/frmDsiEMVX.cs#L120>

Appendix 16 – Supported Transactions

TranType	TranCode	Swipe	Manual
Credit	Sale	✓	✓
	Return	✓	✓
	PreAuth	✓	✓
	PreAuthCapture	✓	✓
	Void Sale	✓	✓
	Void Return	✓	✓
	VoiceAuth	✓	✓
	Adjust	✓	✓
	ZeroAuth	✓	✓
EMV	EMVSale	EMVVoidReturn	inserted
	EMVReturn	EMVVoiceAuth	
	EMVVoidSale	EMVZeroAuth	
	EMVPreAuth		
Tokenization	Initial request for Credit Sale, PreAuth and Return TranCodes requesting Token Record and Frequency	✓	✓
	All “ByRecordNo” token requests for EMV and Credit TranCodes with the exception of ZeroAuth; Also includes IncrementalAuthByRecordNo and PartialReversalByRecordNo.		Token
US PIN Debit	Sale (Allows Cash Back)	✓	
	Return	✓	
Canadian EMV	Credit and Debit supported—See Canadian EMV Integration Guide for full details	✓	✓
FSA	FSASale	✓	✓
	CardLookup	✓	✓
	ReverseFSASale	✓	✓
EBT	Foodstamp Sale	✓	✓
	Foodstamp Return	✓	✓
	Foodstamp – Balance	✓	✓
	Cash Sale (Allows Cash Back)	✓	✓
	Cash Balance	✓	✓
	Foodstamp - Voucher		✓
Admin	BatchSummary		
	BatchClear (Supported only for Batch capture close method)		
	BatchClose		
StoreCard/Gift	Issue and Virtual Issue	✓	✓
	Sale	✓	✓
	Return	✓	✓
	Void Issue	✓	✓
	Void Sale	✓	✓
	Void Return	✓	✓
	Balance	✓	✓
	NoNSFSale	✓	✓
CardLookup	CardLookup	✓	✓

APPENDIX 17 – MercuryPay Platform, Datacap Errors

Error, Exception, and Status Response Messaging

DSI Server Specific (TCP/IP)

Code	Description
002000	Password Verified
002001	Queue Full
002002	Password Failed – Disconnecting
002003	System Going Offline
002004	Disconnecting Socket
002006	Refused 'Max Connections'
002008	Duplicate Serial Number Detected
002009	Password Failed (Client/Server)
002010	Password failed (Challenge/Response)
002011	Internal Server Error – Call Provider

DSIClientX Specific (TCP/IP)

Code	Description	Code	Description
001001	General Failure	004019	TStream Type Missing
001003	Invalid Command Format	004020	Could Not Encrypt Response- Call Provider
001004	Insufficient Fields	009999	Unknown Error
001006	Global API Not Initialized	100201	Invalid Transaction Type
001007	Timeout on Response	100202	Invalid Operator ID
001011	Empty Command String	100203	Invalid Memo
003002	In Process with server	100204	Invalid Account Number
003003	Socket Error sending request	100205	Invalid Expiration Date
003004	Socket already open or in use	100206	Invalid Authorization Code
003005	Socket Creation Failed	100207	Invalid Authorization Code
003006	Socket Connection Failed	100208	Invalid Authorization Amount
003007	Connection Lost	100209	Invalid Cash Back Amount
003008	TCP/IP Failed to Initialize	100210	Invalid Gratuity Amount
003009	Control failed to find branded serial (password lookup failed)	100211	Invalid Purchase Amount
003010	Time Out waiting for server response	100212	Invalid Magnetic Stripe Data
003011	Connect Cancelled	100213	Invalid PIN Block Data
003012	128 bit CryptoAPI failed	100214	Invalid Derived Key Data
003014	Threaded Auth Started Expect Response Event (Note it is possible the event could fire before the function returns.)	100215	Invalid State Code
003017	Failed to start Event Thread.	100216	Invalid Date of Birth
003050	XML Parse Error	100217	Invalid Check Type
003051	All Connections Failed	100218	Invalid Routing Number
003052	Server Login Failed	100219	Invalid TranCode
003053	Initialize Failed	100220	Invalid Merchant ID
004001	Global Response Length Error (Too Short)	100221	Invalid TStream Type
004002	Unable to Parse Response from Global (Indistinguishable)	100222	Invalid Batch Number
004003	Global String Error	100223	Invalid Batch Item Count
004004	Weak Encryption Request Not Supported	100224	Invalid MICR Input Type
004005	Clear Text Request Not Supported	100225	Invalid Driver's License
004010	Unrecognized Request Format	100226	Invalid Sequence Number

Code	Description	Code	Description
004011	Error Occurred While Decrypting Request	100227	Invalid Pass Data
004017	Invalid Check Digit	100228	Invalid Card Type
004018	Merchant ID Missing		

MToken Error Responses (CmdStatus = ERROR)

The **DSIReturnCode** identifies the class of error, and the **TextResponse** indicates the specific error.

ReturnCode	TextResponse	Comments
004116	Merchant Setting Requires RecordNo and Frequency	The request does not contain <i>RecordNo</i> and <i>Frequency</i> .
004116	Merchant Setting Requires RecordNo	The request does not contain <i>RecordNo</i> .
004117	Merchant Setting Does Not Accept RecordNo and Frequency.	The merchant is not setup for tokenization.
004118	Decryption Service Unavailable. – Status Cod Not 200 – 503.	The token decryption service is unavailable.
004118	Decryption Service Unavailable. – 200 Response with Status:Failure Message:Decrypted data invalid	The token decryption service is unavailable.
004118	Decryption Service Unavailable. – 200 Response with Status:Failure Message: Failed to decrypt	The token decryption service is unavailable.
004119	Token Service Unavailable. 200 Response with Status: Failure message: Parse token failure.	Inconsistent use of token frequency - initial transaction included one frequency (e.g., “OneTime”) and subsequent transaction used a different frequency (e.g., “Recurring”) or there is corrupted data in the <i>RecordNo</i> or the token has expired
004119	Token Service Unavailable. – 200 Response with Status:Error Message:Error generating token.	The token service is unavailable.
004119	Token Service Unavailable. – Wait Message Failed. Timeout Waiting for Verifiable Packet from server (ReceiveAndDecrypt)9999	Timeout error.
004119	Token Service Unavailable. – Socket Error Trying to Connect Error 3006	Socket error.
004119	200 Response with Status:Failure Message:Create token failure.	The token service is unavailable.
004119	200 Response with Status:Failure Message:Token invalid	Invalid token.
004120	RecordNo must only be supplied on ByRecordNo Transaction	A PreAuth was attempted to be captured by a non- <i>RecordNo</i> transaction or <i>VoidSale</i> was requested with <i>AcctNo</i> instead of <i>RecordNo</i>
100256	Invalid Field-Frequency	The request does not contain <i>Frequency</i> or there is corrupted data in the <i>Frequency</i>

Appendix 18—HostedCheckout CSS Functionality

Cascading Style Sheets

CSS (Cascading Style Sheets) provide extensive styling capability for HostedCheckout pages. CSS uses a separate method to submit, and have Worldpay store, the merchant's CSS. This process is separate from the initialize payment process. Below are examples of the calls needed to support CSS.

HostedCheckout Service: HCService

Development URL:	https://hc.mercurycert.net/hcws/hcservice.aspx
Production URL:	https://hc.mercurypay.com/hcws/hcservice.aspx

Below are listed the UploadCSS, DownloadCSS and RemoveCSS parameter and examples.

UploadCSS

Parameter	Type	Length	Description	Required
MerchantID	Numeric	24	The Merchant's account or <i>MerchantID</i> that payments will be processed under. The typical format for a HostedCheckout merchant ID is: 88430087469 Please contact your developer integrations analyst for the test MerchantID.	Yes
Password	String	16	The web services password generated by the merchant or dealer/developer on the MercuryView portal and entered into the POS configuration screen to allow access to the web service. Please contact your developer integrations analyst for the test password.	Yes
Css	String	20,000 max	Merchant specific CSS .	Yes

Upload Request Code Example

```

HCService.CssUploadRequest request = new HCService.CssUploadRequest();
request.MerchantID = "778825001";
request.Password = "KR568r@1spyC13,T";
request.Css = ".divTotalDefault{color: #000000;background-color: red;}

<!--Call the web service to initialize the UploadCSS request.-->
HCService.HCService hcWS = new HCService.HCService();
HCService.CssAdminResponse response = new HCService.CssAdminResponse();
response = hcWS.UploadCSS(request);

if (response != null)
{
    this.txtCode.Text = response.ResponseCode.ToString();
    this.txtResponseMsg.Text = HttpUtility.HtmlDecode(response.Message);
}
else
<!--something seriously wrong. probably couldn't connect to the web service at all-->
    this.txtResponseMsg.Text = "Response from UploadCss was Null";

```

DownloadCSS

Parameter	Type	Length	Description	Required
MerchantID	Numeric	24	The Merchant's account or <i>MerchantID</i> that payments will be processed under. The typical format for a HostedCheckout merchant ID is: 88430087469 Please contact your developer integrations analyst for the test MerchantID.	Yes
Password	String	16	The web services password generated by the merchant or dealer/developer on the MercuryView portal and entered into the POS configuration screen to allow access to the web service. Please contact your developer integrations analyst for the test password.	Yes
Formatting	String	3	Values are "on" and "off". A value of "on" will include carriage returns and formatting to make the CSS more readable. Default is "off".	No

Download Request Code Example

```

HCSERVICE.CssDownloadRequest request = new HCSERVICE.CssDownloadRequest();
request.MerchantID = "778825001";
request.Password = "KR568r@1spyC13,T";
request.Formatting = "on";

<!--Call the web service to initialize the DownloadCSS request.-->
HCSERVICE.HCSERVICE hcWS = new HCSERVICE.HCSERVICE();
HCSERVICE.CssDownloadResponse response = new HCSERVICE.CssDownloadResponse();
response = hcWS.DownloadCSS(request);

if (response != null)
{
    this.txtCode.Text = response.ResponseCode.ToString();
    this.txtResponseMsg.Text = response.Message;
    this.txtCSS.Text = HttpUtility.HtmlDecode(response.Css);
}
else
<!--something seriously wrong. probably couldn't connect to the web service at all-->
    this.txtResponseMsg.Text = "Response from DownloadCss was Null";

```

RemoveCSS

Parameter	Type	Length	Description	Required
MerchantID	Numeric	24	The Merchant's account or <i>MerchantID</i> that payments will be processed under. The typical format for a HostedCheckout merchant ID is: 88430087469 Please contact your developer integrations analyst for the test MerchantID.	Yes
Password	String	16	The web services password generated by the merchant or dealer/developer on the MercuryView portal and entered into the POS configuration screen to allow access to the web service. Please contact your developer integrations analyst for the test password.	Yes

Remove Request Code Example

```

HCSERVICE.CssRemoveRequest request = new HCSERVICE.CssRemoveRequest();
request.MerchantID = "778825001";
request.Password = "KR568r@1spyC13,T";

```

```
<!--Call the web service to initialize the DownloadCSS request.-->
HCService.HCService hcWS = new HCService.HCService();
HCService.CssAdminResponse response = new HCService.CssAdminResponse();
response = hcWS.RemoveCSS(request);

if (response != null)
{
    this.txtCode.Text = response.ResponseCode.ToString();
    this.txtResponseMsg.Text = response.Message;
    this.txtCSS.Text = String.Empty;
}
else
<!--something seriously wrong. probably couldn't connect to the web service at all-->
this.txtResponseMsg.Text = "Response from RemoveCss was Null";
```

Note: A comprehensive CSS White List is available upon request from your Integration Consultant

Appendix 19 – List of Acronyms

Acronym	Definition	Acronym	Definition
AVS	Address Verification Service	PABP	Payment Application Best Practices
BID	Business ID	PA-DSS	Payment Application Data Security Standard
BIN	Bank Identification Number	PAN	Personal Account Number or Primary Account Number
CAT	Cardholder Activated Transaction (Kiosk)	PA-QSA	Payment Application Qualified Security Assessor
CISP	Cardholder Information Security Program	PCI	Payment Card Industry
CVV	Cardholder Verification Value	PCI DSS	Payment Card Industry Data Security Standard
DES	Data Encryption Standards	PCI PED	Payment Card Industry PIN Entry Device
DNS	Domain Name Server	PCI PTS	Payment Card Industry PIN Transaction Security
DUKPT	Derived Unique Key Per Transaction	PCI SSC	Payment Card Industry Security Standards Council
E2E	End-to-End Encryption (synonym for P2PE)	PED	PIN Entry Device
EBT	Electronic Benefits Cards: PIN-based food stamp and cash benefits cards	PIN	Personal Identification Number
EMV	Europay, MasterCard and VISA (Canada only)	POS	Point of Sale
FACTA	Fair and Accurate Credit Transaction Act of 2003	PSCS	Payment Systems Configuration Server (Datacap)
FNS	Food and Nutrition Services	PSP	Payment Service Provider
FSA	Flexible Spending Account	PTS	PIN Transaction Security
HID	Human Interface Device	PVV	Pin Verification Values
HIPAA	Health Insurance Portability and Accountability Act	QIR	Qualified Integrator and Reseller
HSA	Healthcare Spending Account	QPS	Quick Payment Service
IIAS	Inventory Information Approval System	QSA	Qualified Security Assessor
ISO	Independent Service Provider	QSR	Quick Service Restaurant
ISV	Independent Service Vendor (synonym for Developer)	RecordNo	MercuryPay's term for Token
JSON	JavaScript Object Notation	REST	Representational State Transfer
KB	Keyboard	RPC	Remote Procedure Call
P2PE	Point-to-Point Encryption	SIGIS	The Special Interest Group for IIAS Standardization
KBE	Keyboard Emulation	SNAP	Supplemental Nutrition Assistance Program
KSN	Key Serial Number	SOAP	Simple Object Access Protocol
MID	Merchant ID	TDES	Triple Data Encryption Standards
MMID	Multiple Merchant ID	TID	Terminal ID
MMT	Multiple-Merchant Tokenization	UAT	Unattended Acceptance Terminal (Kiosk)
MOTO	Mail Order/Telephone Order	VAR	Value Added Reseller
MSR	Magnetic Stripe Reader	VPN	Virtual Private Network
OOS	Out-of-scope	WSDL	Web Services Description Language
		XML	Extensible Markup Language