PCCharge DevKit

Electronic Payment Processing Software
User's Manual

Copyright Dec-10, VeriFone, Inc.
Version 5.9.2 sp1

Notice

VeriFone, Inc. provides this publication “as is” without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of non-infringement, merchantability or fitness for a particular purpose. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. VeriFone may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Copyright © December 10 VeriFone, Inc. All rights reserved. active-Charge, active-Charge SDK, **PC**Charge Payment Server, **PC**Charge Pro, **PC**Charge DevKit, Virtual-Charge, IP-Charge are trademarks and PC-Charge is a registered trademark of VeriFone, Inc.

Microsoft is a registered trademark and Windows is a trademark of Microsoft Corporation. Other brands and their products are trademarks or registered trademarks of their respective holders and should be noted as such.

© VeriFone, Inc.
8001 Chatham Center Drive, Suite 500
Savannah, Georgia 31405
Development Support: (877) 659-8983
E-Mail: devsupport@verifone.com
Technical Support: (877) 659-8981
E-Mail: support@verifone.com

Table of Contents

Table of Contents 3

Software License 8

Important Security Notice 9
- Introduction and Scope 9
- Application Best Practice 9
- Password and Account Settings 11
- Logging 12
- Wireless Networks 13
- Network Segmentation 13
- Secure Remote Software Updates 14
- Remote Access 15
- Encrypting Network Traffic 16
- Training and Communications Program 16
- PA-DSS Version Information 17

CHAPTER 1 -- PCCharge DevKit Introduction 18

Introduction 19
- PCCharge DevKit 19
- PCCharge DevKit Suite 19

Product Components 20
- PCCharge DevKit 20
- PCCharge DevKit Development Support 20
- PCCharge Pro and PCCharge Payment Server 20

CHAPTER 2 -- Getting Started 23

Getting Started 24
- 1) Install the DevKit and the PCCharge Products 24
- 2) Install the Test Merchant Accounts 24
- 3) Verify that PCCharge is Set Up Properly 26
- 4) Determine Which Integration Method Will Be Used 29
- 5) Determine Which PCCharge Product(s) Will Be Supported 36
- 6) Review Payment Processing Basics and Integration Information and Settings 40

CHAPTER 3 -- Payment Processing Basics 41

Credit Card Processing 42
- Host based and Terminal based Processors 42
- Credit Card Transactions 43
- A Normal Credit Card Processing Day 44
- Credit Card Transaction Rates 45

Debit Card Processing 46
- Debit Card Transactions 47
- A Normal Debit Card Processing Day 47

Check Processing 48
- Check Conversion 48
- Check Transactions 49
- A Normal Check Conversion Processing Day 49

EBT Processing 50
- A Normal Day of Processing EBT Transactions 51

Gift Card Processing 53
- A Normal Day of Gift Card Processing 53

CHAPTER 4 -- Integration Information and Settings 54
Warnings, Tips, and Guidelines 55
Timeouts 59
Understanding Timeouts 59
Transaction Delays 59
Dial-Up Modem Backup Settings 60
Setting the Integrated Application's Timeout Value 60
Handling Timeouts 61
Multi-User Support 62
Support for Multiple Workstations 62
Support for Simultaneous Transaction Requests 63
Setting up Multi-user support 64
Additional Users 64
Unlimited User License 65
Limitations of PCCharge's Multi-User Feature 66
Alternatives 66
Multi-trans Wait 67
Multi-Merchant Support 68
Multi-Merchant Integration 68
Use Default Processor 69
SSL Support 70
Introduction 70
SSL Configuration 71
SSL Integration Notes 72
Custom SSL Certificates 73
Manual SSL Certificate Install 74
Cashier Permissions 75
Running transactions with cashier credentials 83
Follow On Transactions 86
Overview 86
Examples 87
Implementing Follow On Transactions 90
Non-TroutD Post-Authorizations 91
REF_TROUTD 93
Reversals 94
Commercial Card Transactions 95
Overview 95
Supporting Commercial Card Transactions 95
Using Bin.mdb 95
Submitting Commercial Card Transactions 97
Example 98
Flexible Spending Accounts (FSA/HRA) Transactions 99
Overview 99
FSA/HRA Examples 101
Discretionary Data 104
Restaurant Transactions 105
Overview 105
Benefits of XML 105
Integration 105
Examples 106
Processor Specific Notes 109
Gift Card Transactions 110
VeriFone Stored Value API (GAPI) 112
Pre-Paid Credit Card Transactions 113
Partial Authorization (Partial Auth) 115

Citi© Transactions/ Credit Plan 116
Canadian (Interac) Debit Transactions 121
Overview 121
Definitions 121
Integration Notes 122
Integration 123
Process Flow when using the processor Global Payments East (NDC) 124
Process Flow when using the processor Chase Paymentech (GSAR) 127
Transaction Inquiry 128
Overview 128
Usage 128
Example 130
Batch Settlement 131
Overview 131
Simple Batch Request 131
Multiple Batches 132
Batch Inquire 133
Batch Inquire with Card Type 133
Amex Settlement 134
Gift Settlement 138
Batch Totals Storage 139
BatchTotals Table 139
Health Message Transaction 140
Command Line Switches 141

CHAPTER 5 -- DevKit Constants 142

DevKit Constants 143
Action Codes 143
Address Verification Response Codes 148
CVV2/CVC2/CID Response Codes 149
Credit Card Types 149
System Error Codes and Descriptions 150
SYS.PCC Codes and Descriptions 151
Cashier Permissions Constants 152
Processing Company Codes 154
Transaction Result Constants 156

CHAPTER 6 -- PCCharge Integration Methods 157

Pseudo-code 158
Credit Card Sale/Pre-Authorization – Retail / Card Present 159
Credit Card Sale/Pre-Authorization – Card Not Present 163
Level II (Commercial, Purchasing, etc.) Card Sale 167
Credit Card Void 171
Credit Card Sale/Pre-Authorization – Restaurant 174
Credit Card Gratuity – Restaurant 178
Debit Sale 181
Reports 185
OCX (ActiveX) Method 188
Charge.OCX 191
Debit.OCX 208
Check.OCX 217
EBT 224
GiftCard.OCX 225
Batch.OCX 235
Device.OCX 242
SC550.OCX 246

SC5X.OCX 255
SC5X.OCX Error Codes 259
Reporting 260
DLL (ActiveX) Method 264
Charge Class 265
Debit Class 280
Debit Class 280
Check Class 287
EBT 294
Gift Class 295
Batch Class 304
Offline Class 310
Reporting 312
OLE/COM Method 316
PccCharge Class 318
PCCDebit Class 332
PccCheck Class 339
PCCEBT Class 345
PCCGiftCard Class 352
PccBatch Class 360
PccSettle Class 363
PccSettleGift Class 366
PccPinPad Class 368
PccSC550 Class 371
PccBin Class 373
Reporting 374
Utility Related Classes 378
Setup Related Classes 382
Classes No Longer Supported / Internal Use Classes 480
File Method 481
Introduction 481
File Method Integration 481
File Layout Specifications 483
Credit File Layouts 486
Debit File Layouts 496
Check File Layouts 500
EBT File Layouts 504
Gift File Layouts 508
Batch File Layouts 514
Report File Layouts 517
Configuration File Layouts 519
Various Utility File Layouts 520
Cashier Configuration File Layout 522
TCP Interface 524
PAYware SIM - SIM.DLL (ActiveX) Method 525
PAYware SIM 525

CHAPTER 7 -- Code Sample Information 526

Code Samples 527
Java Client 527
Web-based Integration Samples 529
System Requirements 529
PCCharge Virtual Terminal Sample 530
ASP Sample 535
Cold Fusion 536
Java 537

General Troubleshooting 539
Permissions 539
User In Use 540
Appendix 541
Test Credit Cards and Merchant Accounts 541
Test Credit Card Numbers 542
Test Track Data 543
Test Merchant Account Information 544
Integration Troubleshooting 549
IODebug.log 549
Transaction Request Duplication 550
Communication Log 550
Error Log 551
Troubleshooting a Live Installation 551
Contacting Support 552
Distribution and Deployment 553
Distribution Methods 553
Demo Mode 554
Evaluation Mode 554
Warehousing/Block Inventory 554
Activation 555
Support Policy 556
Philosophy 556
Contact 556
More Information 556

Software License

1. **GRANT OF LICENSE.** VeriFone, Inc. grants you the right to use a single copy of this Software, including documentation, on one computer of your choice. You may physically transfer each License, without cost, to a different computer, providing it is removed from the first computer. Please remember that when you buy Software, you are actually buying the rights to use the Software on one computer at a time. It is against Federal laws to use this Software on more than one computer at a time.

2. **RESTRICTED USE.** The Program may not be copied except for backup purposes. Copying of the Manual and Interface Specifications is prohibited. You may not remove any product identification, copyright, or other proprietary notices from the Software or Documentation.

3. **WARRANTY.** VeriFone, Inc. warrants that the original compact disc is free from defects in material and workmanship for a period of 30 days from the date of purchase. If a defect occurs during this time, VeriFone, Inc. will replace your compact disc free of charge.

4. **DISCLAIMER.** The **PC**Charge DevKit package is Licensed on an "as is" basis. There are no warranties, expressed or implied, including, but not limited to, warranties of merchantability, of fitness for a particular purpose, and all such warranties are expressly and specifically disclaimed. VeriFone, Inc. shall have no liability or responsibility to you or any other person or entity with respect to any liability, loss, or damage caused or alleged to be caused directly or indirectly by the **PC**Charge DevKit. Use of the **PC**Charge DevKit system signifies agreement with this disclaimer and is subject to the License Agreement provided with the installation compact disc.

Important Security Notice

Introduction and Scope

PA-DSS / PCI Security Standards Council Requirements:
In June 2001, Visa mandated its Cardholder Information Security Program (CISP). This program progressed to the Payment Application Best Practices (PABP). PABP is a standard for securing cardholder data, wherever it is located. Visa requires current DSS / PCI compliance of all entities that store, process, or transmit Visa cardholder data.

Starting September 30, 2008 this program advances to the Payment Application Data Security Standard (PA-DSS). This includes all merchants, Integrators and Resellers who use **PC**Charge to process transactions. The Payment Card Industry (PCI) Security Standards Council is an open global forum for the ongoing development, enhancement, storage, dissemination and implementation of security standards for account data protection.

PCI offers a single approach to safeguarding sensitive data for all card brands. The information below gives you further detail on DSS / PCI requirements and what you must do to be compliant for the PA-DSS Payment Application Data Security Standard. These requirements also include that Microsoft Windows OS restore points be disabled when you are using a Microsoft Windows system to process transactions.

What is the Payment Application Data Security Standard?
The Payment Application Data Security Standard (PA-DSS) is a set of security standards that were created by the PCI Security Standards Council to guide payment application vendors in implementing secure payment applications.

Distribution and Updates
This VeriFone PA-DSS Implementation guide will be disseminated to all relevant application users including customers, merchants, resellers and integrators. It is updated with each version release and annually to reflect changes in the PA-DSS standard. The annual review and update will include new software changes (when applicable) as well as changes in the PA-DSS standard.
Updates to the PA-DSS Implementation guide can be obtained by going to the VeriFone website. In addition, VeriFone support will publish updates and update notifications as needed.

Versions
This PA-DSS Implementation guide references both the DSS and PCI requirements. The following versions were referenced in this guide:

- DSS version 1.4
- PCI PA-DSS version 1.1

Application Best Practice

Merchant Applicability
No sensitive Authentication data can be stored. It is the merchant's responsibility to remove any magnetic stripe data, card validation values or codes, PINs or PIN block data, cryptographic key material, or cryptograms stored by previous versions of the software. Removal of this prohibited historical data is required for PCI compliance.

Protect Stored Data

- 2.2.1 Mask Account Numbers when Displayed.

- All account numbers are masked when displayed after entry.
- 2.2.2 Render Sensitive Cardholder Data Unreadable.
- All sensitive cardholder data is rendered unreadable in PCCharge logs, database, history files and trace files.
- 2.2.3 Protect Encryption Keys against disclosure and misuse.
- Encryption keys are obfuscated to prevent detection and provide encryption.

Secure Deletion Instructions
The following instructions can be used to securely delete prohibited cardholder transaction and historical data. All sensitive data is masked. The data at deletion can have a null value written in place of the data and then deleted. Cryptographic data must be removed by a secure delete. Any data deleted from the database is done via a stored procedure using a SQL DELETE command. This includes the historical data that must be removed (magnetic stripe data, card validation codes, PINs, or PIN blocks stored by previous versions of the software). The removal of this data is required for PCI compliance. This policy is to be reviewed at least annually and updated as the environment changes.

The PA-DSS Requirements Reference:
1.1.4 Securely delete any magnetic stripe data, card validation values or codes, and PINs or PIN block data stored by previous versions of the software. (PA-DSS 1.1.4)
1.1.5 Securely delete any cryptographic key material or cryptogram stored by previous versions of the software. This could be a cryptographic key used for computation or verification of cardholder data or sensitive authentication data. (PA-DSS 1.1.5)

Reseller/Integrator Applicability
It is the responsibility of the reseller or integrator to follow the following guidelines:

- Resellers/integrators must collect sensitive authentication only when needed to solve a specific problem.
- Resellers/integrators must store such data only in specific, known locations with limited access.
- Resellers/integrators must collect only the limited amount of data needed to solve a specific problem.
- Resellers/integrators must encrypt sensitive authentication data while stored.
- Resellers/integrators must securely delete such data immediately after use.

PA-DSS Requirements Reference:
1.1.5c Securely delete any log files, debugging files, and other data sources received from customers for debugging or troubleshooting purposes, to ensure that magnetic stripe data, card validation codes or values, and PINS or PIN block data are not stored on software vendor systems. These data sources must be collected in limited amounts and only when necessary to resolve a problem, encrypted while stored, and deleted immediately after use. (PA-DSS 1.1.5.c)

Purging Cardholder Out of Date Data:

- The end user merchant will not hold out of date cardholder data. The retention period will be merchant defined.
- The **PC**Charge user's manual provides for customers, resellers and integrators guidance on data removal and the location of the data to be deleted.

The PA-DSS Requirements Reference:
2.1.a Review the Implementation Guide prepared by the vendor and verify the documentation includes the following guidance for customers and resellers/integrators. Cardholder data exceeding the merchant-defined retention period must be purged. All locations where the payment application stores cardholder data (so that merchant knows the locations of the data that needs to be deleted.

Password and Account Settings

Access Control
Merchants, resellers and integrators are advised to control access, via unique username and PCI DSS compliant complex passwords and the granting of restricted user access rights, to any PCs, servers, and databases with payment applications and cardholder data. All activity is logged. This covers all transactions attempted and completed. Audit reports should be run at the end of a shift / end of day to verify all valid and invalid user authentications

Passwords
The following guidelines should be followed:

- Customers and resellers/integrators are advised against using administrative accounts for application logins (e.g., don't use the "sa" account for application access to the database). (PA-DSS 3.1c)
- Customers and resellers/integrators are advised to assign strong passwords to these default accounts (even if they won't be used), and then disable or do not use the accounts. (PA-DSS 3.1)
- Customers and resellers/integrators are advised to assign strong application and system passwords whenever possible. (PA-DSS 3.1c)
- Customers and resellers/integrators are advised how to create PCI DSS-compliant complex passwords to access the payment application, per PCI Data Security Standard 8.5.8 through 8.5.15. (PA-DSS 3.1c)
- Customers and resellers/integrators are advised to control access, via unique username and PCI DSS-compliant complex passwords, to any PCs, servers, and databases with payment applications and cardholder data. (PA-DSS 3.2)
- Customers and resellers/integrators are advised that passwords are encrypted during both transmission and storage, using strong cryptography. (PA-DSS 3.3)
- Passwords should meet the requirements set in PCI DSS section 8.5.8 through 8.5.15, as listed here.
- Do not use group, shared, or generic accounts and passwords. The username must be unique. Passwords are to complex (min. of 7 characters including 1 capital letter, 1 number and 1 special character)
- Change user passwords at least every 90 days
- Require a minimum password length of at least seven characters
- Use passwords containing both numeric and alphabetic characters
- Do not allow an individual to submit a new password that is the same as any of the last four passwords he or she has used
- Limit repeated access attempts by locking out the user ID after not more than 6 attempts
- Set the lockout duration to thirty minutes or until administrator enables the user ID
- If a session has been idle for more than 15 minutes, require the user to re-enter the password to re-activate the terminal

PA-DSS Requirements Reference:
3.1 Application must require unique usernames and complex passwords for all administrative access and for all access to cardholder data.
3.2 Access to PCs, servers, and databases with payment applications must require a unique username and complex password.
3.3 Application must require password files to be encrypted during transmission and storage.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

Logging

Merchant Applicability
PCCharge meets all PA-DSS requirements for logging.

PCI Guidelines for Logging
Implement automated audit trails for all system components to reconstruct the following events:

- All individual accesses to cardholder data.
- All actions taken by any individual with root or administrative privileges.
- Access to all audit trails.
- Invalid logical access attempts.
- Use of identification and authentication mechanisms.
- Initialization of the audit logs.
- Creation and deletion of system-level objects.

Record at least the following audit trail entries for all system components for each event:

- User identification.
- Type of event.
- Date and time.
- Success or failure indication
- Origination of event.
- Identity or name of affected data, system component, or resource.

Configuring Log Settings
The **PC**Charge User's Manual provides guidance on Configuring the Log Settings.

PA-DSS Requirements Reference:
Testing Procedure 4.2b If application log settings are by default set to full logging. When this is changed the customer and resellers/integrators will no longer be in compliance.

Wireless Networks

Merchant Applicability
If the merchant implements the payment application into a wireless environment, or implements wireless networking into a payment application environment, configure the wireless environment to adhere to PCI requirements 1.3.9, 2.1.1, and 4.1.1.

PCI Requirements:

- (PCI 1.3.8, 1.3.9) Installation and configuration of personal firewalls on employee-owned computers with direct connectivity to the Internet which are used to access the organization's network
- (PCI 2.1.1) Modify default wireless settings, including:
- Change wireless equivalent privacy (WEP) keys
- Change default service set identifier (SSID)
- Change default passwords
- Change SNMP community strings
- Disable SSID broadcasts
- Enable WiFi protected access (WPA and WPA2) technology for encryption and authentication when WPA-capable.
- (PCI 4.1.1) For wireless networks transmitting cardholder data, encrypt the transmissions by using Wi-Fi protected access (WPA or WPA2) technology, IPSEC VPN, or SSL/TLS. Never rely exclusively on wired equivalent privacy (WEP) to protect confidentiality and access to a wireless LAN. Secure encryption rules apply.

If WEP is used, do the following:

- Use with a minimum 104-bit encryption key and 24 bit-initialization value
- Use ONLY in conjunction with Wi-Fi protected access (WPA or WPA2) technology, VPN, or SSL/TLS
- Rotate shared WEP keys quarterly (or automatically if the technology permits)
- Rotate shared WEP keys whenever there are changes in personnel with access to keys
- Restrict access based on media access code (MAC) address

PA-DSS Requirements Reference:
Testing Procedure 6.1.c If customer could implement the payment application into a wireless environment, examine PA-DSS Implementation Guide prepared by vendor to verify customers and resellers/integrators are instructed on PCI DSS-compliant wireless settings, per PCI Data Security Standard 1.3.8, 1.3.9, 2.1.1 and 4.1.1.

Network Segmentation

Merchant Applicability
Any machines that can store credit card data should not be directly connected to the Internet. For example, web servers and database servers should not be installed on the same server. A DMZ must be set up to segment the network so that only machines on the DMZ are Internet accessible.

PA-DSS Requirements Reference:
Testing Procedure 9.1b If customer could store cardholder data on a server connected to the Internet, examine DSS Implementation Guide prepared by vendor to verify customers and resellers/integrators are told not to store cardholder data on Internet-accessible systems (e.g., web server and database server must not be on same server.)

Secure Remote Software Updates

Merchant Applicability
VeriFone, Inc. uses remote connectivity to deliver patches to the **PC**Charge. Merchants should develop an acceptable use of critical employee-facing technologies policy as per the guidelines below. If merchant is receiving updates via modem, the modem should only be activated when downloads are needed.

Acceptable Use Policy
The merchant should develop usage policies for all modems, wired and wireless devices, as per PCI requirement 12.3. These usage policies should include:

- Explicit management approval for use
- Authentication for use
- A list of all devices and personnel with access
- Labeling the devices with owner
- Contact information and purpose
- Acceptable uses of the technology
- Acceptable network locations for the technologies
- A list of company approved products
- Allowing use of modems for vendors only when needed and deactivation after use
- Prohibition of storage of cardholder data onto local media when remotely connected

Personal Firewall
Any "always-on" connections from a computer to a VPN or other high-speed connection should be secured by using a personal firewall product per PCI Data Security Standard 1.3.9.
Remote Update Procedures
PCCharge version 5.8.0 does not have remote update functionality.

PA-DSS Requirements Reference:
10.1 If software updates are delivered via remote access into customers' systems, software vendors must tell customers to turn on modem only when needed for downloads from vendor, and to turn off immediately after download completes. Alternatively, if delivered via VPN or other high-speed connection, software vendors must advise customers to properly configure a personal firewall product to secure "always-on" connections.

Remote Access

Merchant Applicability
If Product can be accessed remotely, all network connectivity should be performed using, at minimum, two-factor authentication that supports encrypted communications per PCI requirement 8.3. This is for all remote access to the network by employees, administrators, and third parties. Use technologies such as remote authentication and dial-in service (RADIUS) or terminal access controller access control system (TACACS) with tokens; or VPN (based on SSL/TLS or IPSEC) with individual certificates.

NOTE: All servers and network devices, whether managed by employees or by third parties, must be built and deployed in accordance with this policy. Exemptions from this policy will be permitted only if approved in advance and in writing by the Chief Security Officer.

Remote Access Software Security Configuration

- Implement the following applicable security features for all remote access software used by the merchant, reseller or integrator:
- Change default settings in the remote access software (for example, change default Passwords and use unique Passwords for each customer)
- Allow connections only from specific (known) IP/MAC addresses.
- Use strong authentication or complex passwords for logins.
- Enable encrypted data transmission.
- Enable account lockout after a certain number of failed login attempts.
- Configure the system so a remote user must establish a Virtual Private Network ("VPN") connection via a firewall before access is allowed.
- Enable the logging function.
- Restrict access to customer Passwords to authorized reseller/integrator personnel.
- Establish customer Passwords according to PCI DSS requirements 8.1, 8.2, 8.4, and 8.5.

PA-DSS Requirements Reference:
11.2 Remote access must be authenticated using a two-factor authentication mechanism.
11.3 If vendors, resellers/integrators, or customers can access customers' applications remotely, the remote access software must be implemented securely.

Encrypting Network Traffic

Transmission of Cardholder data
Any transmission of cardholder data over public networks should be encrypted. Encryption solutions such as SSL/TLS or IPSEC should be used.

Email and Cardholder data
Product does not natively support the sending of email. As per PCI requirement 4.2, cardholder data should never be sent unencrypted via email.

Non-Console administrative access
All non-console administrative access of the application or operating system should be encrypted. Use technologies such as SSH, VPN or SSL/TLS where applicable.

PA-DSS Requirements Reference:
12.1 Use strong cryptography and security protocols such as secure sockets layer (SSL) / transport layer security (TLS) and Internet protocol security (IPSEC) to safeguard sensitive cardholder data during transmission over open, public networks.

Examples of open, public networks that are in scope of the PCI DSS:
- *The Internet*
- *Wi-Fi (IEEE 802.11x)*
- *Global System for Mobile communications (GSM)*
- *General Packet Radio Service (GPRS).*

12.2 The application must never send unencrypted PANs by e-mail.
13.1 Encrypt all non-console administrative access. Use technologies such as SSH, VPN, or SSL/TLS for web-based management and other non-console administrative access.

Training and Communications Program

Overview
VeriFone Inc. has created a training program, per PA-DSS requirement 14.2, to train resellers and integrators on how to implement **PC**Charge into a production environment in a PA-DSS compliant manner.

Vendor Responsibility
VeriFone, Inc. will review and update the training materials annually and after new software versions are released.

Materials
All parties interested in the reseller and integrator training program should contact VeriFone, Inc. Sales for further information.

PA-DSS Requirements Reference:
14.2 Develop and implement training and communication programs to ensure software resellers and integrators know how to implement the application software and related systems and networks in a DSS-compliant manner. Update the training on an annual basis and whenever new software versions are released.

More Information

☛	For more information related to security, visit: http://www.pcisecuritystandards.org http://www.visa.com/cisp http://www.sans.org/resources http://www.microsoft.com/security/default.asp https://sdp.mastercardintl.com/

!	**WARNING:** Although VeriFone, Inc. has designed **PC**Charge to properly secure credit card cardholder information according to PCI guidelines; *it is ultimately the merchant's responsibility to secure the system which hosts **PC**Charge and the environment which hosts the Point of Sale.*

!	**WARNING:** VeriFone, Inc. takes security seriously therefore we have enhanced our integration methods to enhance security for sensitive data. If using **File Method** integration to PCCharge we strongly suggest you consider migrating towards a more secure integration method such as **TCP/IP SSL** socket integration.

PA-DSS Requirements Reference:
The following DSS Requirements were referenced in this document:

DSS Requirement	*DSS Sub-requirement(s)*
1	*1.1.4, 1.1.5, 1.1.6*
2	*2.1*
3	*3.1, 3.2, 3.3*
4	*4.2b*
6	*6.1*
9	*9.1*
10	*10.1*
11	*11.2, 11.3*
12	*12.1, 12.2*
13	*13.1*
14	*14.2*

PA-DSS Version Information

1.2
Last Modified Date: 11 November, 2008 Printed: 11 November, 2008

PA-DSS Document-Security Section History

Revision	Author	Date	Description
1.0	CC.	03/14/2008	Consolidate documents.
1.1	SP.	03/14/2008	Format revisions.
1.2	TH.	11/17/2008	PA-DSS Update.

CHAPTER 1 -- PCCharge DevKit Introduction

Introduction

Thank you for purchasing the **PC**Charge DevKit or the **PC**Charge DevKit Suite.

VeriFone, Inc. is committed to providing integrators and merchants with the most comprehensive electronic payment processing solutions available today. If you have any suggestions as to how we can improve our products, support, or documents, please call us at (877) 659-8983 or e-mail us at devsupport@verifone.com.

This chapter is an introduction to the **PC**Charge DevKit. It will familiarize you with the various components and typical uses of this bundle of products.

PCCharge DevKit

The **PC**Charge DevKit is a bundle of applications, tools, code examples, and documents used to enable electronic payment processing in third party applications. The **PC**Charge DevKit is designed to guide developers while they are building an interface in their application to **PC**Charge Payment Server and/or **PC**Charge Pro.

The **PC**Charge DevKit includes *evaluation* copies of **PC**Charge Pro and **PC**Charge Payment Server for use *only* during integration and testing.

Please note: The evaluation copies of **PC**Charge may *not* be used to perform live payment processing. Merchants *must* purchase a licensed copy of either **PC**Charge Payment Server or **PC**Charge Pro in order to perform live payment processing. Licenses for distribution to resellers and customers are available by contacting an authorized reseller or by calling (800) 725-9264.

PCCharge DevKit Suite

The **PC**Charge DevKit Suite is also a bundle of applications, tools, code examples, and documents used to enable electronic payment processing in third party applications. In fact, the manual, media, code examples, etc. are *identical* to those in the **PC**Charge DevKit. The only difference between the two products is that the **PC**Charge DevKit Suite contains the following licenses which allow live payment processing:

- One (1) live license for either **PC**Charge Payment Server or **PC**Charge Pro.
- One (1) Unlimited User license that can be activated within **PC**Charge Payment Server or **PC**Charge Pro.

Note: In this manual, both the **PC**Charge DevKit and **PC**Charge DevKit Suite are commonly referred to as the "**PC**Charge DevKit" or "DevKit".

Product Components

PCCharge DevKit

The **PC**Charge DevKit is this manual and a group of examples of the various interface methods available to integrate payment processing into applications. Sample code for FoxPro, VB.Net, VB6, C++.Net, C#.Net, Access, Java, Delphi, ASP, Java, and Cold Fusion is included to demonstrate the OCX (ActiveX), DLL (ActiveX), OLE (COM), TCP Interface, and *File Methods of integration.

*** VeriFone takes security seriously therefore we have enhanced our integration methods to enhance security for sensitive data. If using File Method integration to PCCharge we strongly suggest you consider migrating towards a more secure integration method such as TCP/IP SSL socket integration.**

PCCharge DevKit Development Support

The **PC**Charge DevKit includes 30 minutes of telephone developer support to be used within twelve months. After that, additional hours may be purchased in order to receive additional phone support. Telephone/e-mail developer support is provided at a charge of $75.00 per half hour billed in advance in half hour increments.

Consultative coding support is $1500 per day at the VeriFone, Inc. offices in Savannah, Georgia **or** $1500 per day plus travel and expenses at the developer's location.

PCCharge Pro and PCCharge Payment Server

The **PC**Charge DevKit includes copies of **PC**Charge Payment Server and **PC**Charge Pro. **PC**Charge Payment Server and **PC**Charge Pro are the transaction engines that handle all payment processing requests submitted by the third-party integrated application.

Both **PC**Charge Payment Server and **PC**Charge Pro include a GUI (Graphical User Interface) that is used for setup. This GUI can also be used for reporting, settlement, maintenance, and transaction processing if needed. When distributing an integrated application, either **PC**Charge Payment Server or **PC**Charge Pro must be installed and activated on the merchant's computer or on a computer located in the merchant's LAN in order to enable payment processing.

Note: Merchants should not attempt to process transactions or run reports directly from the **PC**Charge GUI while the integrated application is processing transactions or settling batches. If the merchant wishes to process transactions or run reports directly with **PC**Charge (instead of through the integrated application), it is highly recommended that they use the **PC**Charge Client. The **PC**Charge Client is included on the **PC**Charge Pro and Payment Server CD, and can be installed on the same (or on a different) computer as **PC**Charge. The **PC**Charge Client has payment processing and reporting capabilities.

The following outlines the features of **PC**Charge Pro and **PC**Charge Payment Server:

Payment Types Supported

Credit Cards -- **PC**Charge supports credit card transaction processing according to specifications of supported payment processors for the following major credit cards: VISA, MasterCard, Discover, American Express, Optima, Carte Blanche, Diner's Club, and some private label cards.

Debit Cards – **PC**Charge supports debit card transaction processing according to specifications of supported payment processors for both online (ATM) and offline debit cards.

Checks – **PC**Charge supports verification and conversion of a variety of types of paper-based checks including personal, government, payroll, travelers, and so forth. When performing verification transactions, primary and secondary forms of identification are accepted for each check type according to the specifications of the supported check processors.

EBT – **PC**Charge supports EBT transaction processing according to the specifications of the supported payment processors for both food stamp and cash benefit transactions.

Gift Cards – **PC**Charge supports gift card processing according to the specifications of the supported payment processors. **PC**Charge includes support for the following gift card programs (availability depending on processor): loyalty transactions, points-based transactions, multiple issuance, and standard transactions (Sale, Void, etc.).

Hardware Devices Supported

The **PC**Charge graphical user interface (GUI) includes support for many industry-standard hardware devices in addition to new devices that support the OPOS standard. Some of these devices are accessed via a standard COM port, whereas others may be connected to the keyboard (PS2) port. The devices include, but are not limited to the following:

- Report Printers
- Receipt Printers
- Magnetic Stripe Readers
- PINpads
- Check Readers

Note: In order for an integrated application to support these types of devices, the developer must create an interface to each hardware device that will be supported. Some devices simulate keyboard entry (such as Magnetic Stripe Readers) and do not require any special coding to support. However, some devices, such as PINpads, require a complete integration such as a serial port interface. VeriFone, Inc. provides an ActiveX control and an OLE/COM class that can be used to integrate to some PINpads. To support PINpads and other hardware that are not included in the control or the class, check with the various hardware providers for information on hardware integration.

For a current list of the hardware devices that are supported by the **PC**Charge GUI, visit the VeriFone, Inc. website at www.pccharge.com

Reporting

The following reports are available in **PC**Charge. All reports contain filters (such as date, card number, member name, and so forth), allowing applications to designate the specific information to appear on the reports. All reports can be viewed on the screen if using the **PC**Charge GUI. Integrated applications can request that reports be sent to a printer or written to a file.

Transaction Summary Report – This report summarizes information for credit, debit, check, EBT, and gift transactions that have been performed.

Detail Report – Each payment type (credit, debit, check, EBT, and gift) has a detail report available. The reports show line item detail for each transaction.

Batch Pre-Settle Report – This report lists transactions that have been authorized but not settled.

Batch Post-Settle Report – This report lists transactions that have been settled.

Security

Transaction Database Encryption -- **PC**Charge automatically encrypts account numbers within the program's database. Account numbers appearing will show only the first and last four digits of the account number.

PIN Encryption -- **PC**Charge supports PIN data entry devices (as listed in previous sections) for use with online debit card processing. Each PINpad supported by **PC**Charge supports one or more of the two following PIN encryption standards:

1. **Derived Unique Key Per Transaction (DUKPT)** -- Derives a transaction key for the current or next transaction from the previous key plus other data. The scheme generates keys based on a finite list known only to the host and PINpad. The key sequence is unique to each PINpad, resulting in a unique key per transaction. DUKPT is the standard in the U.S., with other DUKPT schemes internationally.

2. **Master/Session Key Management** -- In this method, a key is injected into the PINpad in a secure environment. This "master" key is not used to encrypt PINs. Instead, it is used to decrypt a session or working key that has been encrypted by the host (using the master key) then transmitted over the network to the PINpad. This session key is used to encrypt PINs. The session key can be changed as frequently as every transaction. This encryption method is being phased out in the U.S.

Database Support

PCCharge uses a Microsoft Access database to store and maintain transaction information. This open-architecture, industry-standard database is accessible through ODBC drivers, DAO or ADO.

Utilities

PCCharge includes a number of utilities to facilitate maintenance of the system. These include the ability to back up and restore data files, compact and repair the database, archive transaction history, as well as a number of modem detection and configuration options.

CHAPTER 2 -- Getting Started

Getting Started

It is recommended that several tasks be performed prior to coding. The DevKit and the **PC**Charge products will need to be installed, the test merchant accounts will need to be set up and tested**, the method of integration will need to be chosen, and the differences between PCCharge Payment Server and PCCharge Pro should be reviewed**.

Follow the steps below to set up PCCharge in a development environment:

1) Install the DevKit and the PCCharge Products

To install, insert the DevKit CD and follow the installation instructions. Once the DevKit installation has concluded, the installation program will prompt to install both **PC**Charge Payment Server and **PC**Charge Pro. It is recommended that both products be installed. The default installation options should be chosen while installing the DevKit and the **PC**Charge products.

2) Install the Test Merchant Accounts

The **PC**Charge DevKit comes with test merchant accounts that can be used while integrating credit card processing into third-party applications. The test merchant accounts should be utilized to avoid testing with live merchant accounts.

Follow the procedure below to install the test merchant accounts:

1) Remove any floppy disks from drive A:

2) Start **PC**Charge Payment Server (use the **PC**Charge Payment Server shortcut in **Start** | **Programs** | **Verifone** | **PCCharge Payment Server**). If prompted, enter serial number **1234-1234-123456-54**.

3) After a modem detection completes, the following message will appear:
   "Do you have a configuration disk?"

4) Click **Yes** and then **OK**.

5) When prompted to insert a disk in Drive A, click **Cancel**.

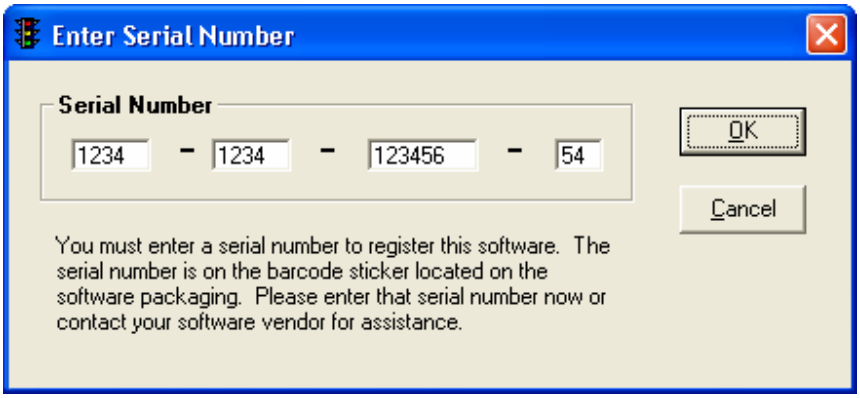
6) When the "Copy Configuration Files From…" box appears, browse to the following directory:
   `C:\Program Files\active-charge SDK\Test Merchant Info`

7) Several files ending in `.pcc` will appear. Click **Open**.

8) When prompted to overwrite files, click **Yes** each time.

9) **PC**Charge Payment Server will now shut down. Upon restart it will ask for a configuration disk again. This time click **No**.

10) When prompted to "Enter Serial Number", enter the serial number that was provided with the DevKit and click **OK**. Enter serial number **1234-1234-123456-54**.



11) Click **OK** to each box that appears.

PCCharge Payment Server is now set up for testing. Please repeat this process to setup **PC**Charge Pro.

Installing the Test Merchant Accounts Manually

The following is an alternative method for installing the test merchant accounts.

The test merchant accounts are included in text-based configuration files (`*.pcc`) that are located in the directory `C:\Program Files\active-charge SDK\Test Merchant Info`. To install the test merchant accounts manually, close **PC**Charge and then copy these files directly into the directories where **PC**Charge Payment Server and **PC**Charge Pro reside:

- `C:\Program Files\Active-Charge` (for **PC**Charge Payment Server)
- `C:\Program Files\PCCW` (for **PC**Charge Pro)

When prompted to overwrite files, choose **Yes**.

Note: Installing the test merchant accounts will overwrite any existing merchant configuration that was previously set up in **PC**Charge. A backup of the **PC**Charge directory is recommended if **PC**Charge was previously set up on the computer.

3) Verify that PCCharge is Set Up Properly

Prior to integrating, it is important to make sure that the test merchant accounts have been set up properly in **PC**Charge Payment Server and **PC**Charge Pro. This should be done by running test credit card transactions directly from the GUI of each program.

Follow the procedure below to run credit card transactions in either **PC**Charge Payment Server or **PC**Charge Pro:

1) Start **PC**Charge Payment Server or **PC**Charge Pro.

   Note: After the initial setup of **PC**Charge Payment Server, its GUI is hidden by default each time the program starts. Double-click the "dollar sign" icon in the system tray to open the GUI.

WARNING: As of **PC**Charge version 5.8.0, you will be required to create a **System** password at program launch. You will also be required to set up a cashier named **Manager. Manager** can perform any functions involving transactions, including setting up merchant accounts and devices, processing transactions, settling batches, and running reports within the user interface, **this will not affect any existing integration**; you do not have to send any cashiers when using an integrated application. **Manager** cannot access the cashier setup, perform the new key generation function or access the Data Manager.

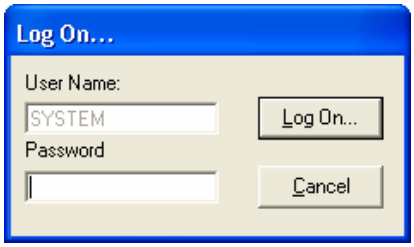
WARNING: PC Charge requires 'strong' passwords, and password variety is enforced. Passwords must be changed every 90 days. Reminder prompts will start when there are 14 days or less. A Password cannot be used if it has been one of the last four used. See next warning box for the **PC**Charge password requirements.

WARNING: If the **System** user attempts to login 5 times incorrectly, **PC**Charge will shutdown and remain locked for 30 minutes. The System user will be prompted with an option to call Tech Support for a password reset.

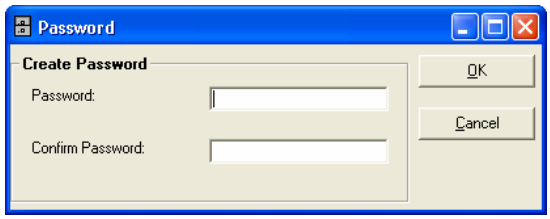
WARNING:
PCCharge requirements for 'strong' password:

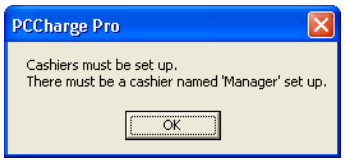
- Must be minimum 7 alpha/numeric characters in length.
- Must be case sensitive
- Must contain at least one upper case character
- Must contain at least one numeric character
- Must contain at least one special character (e.g., @, $, %, etc.)

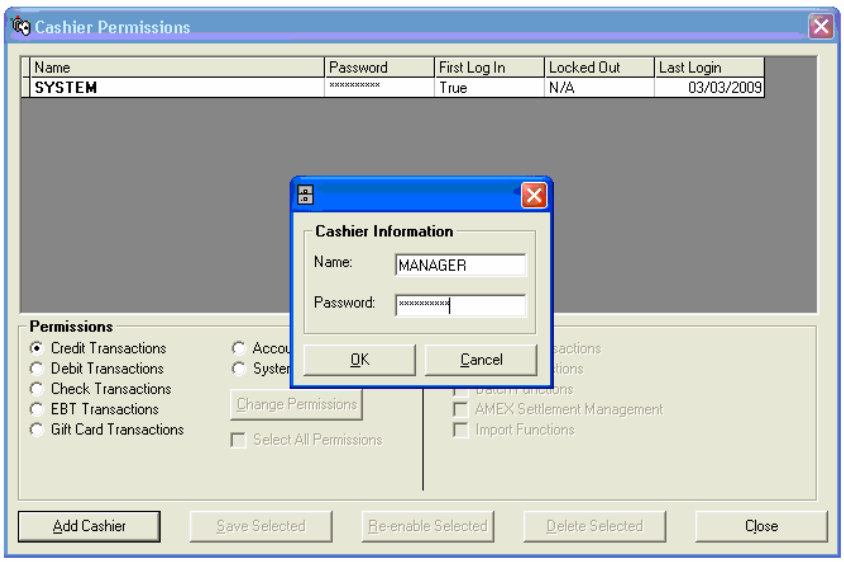
If a password becomes 'corrupt' as if someone has tried to tamper with the system, **PC**Charge will lock itself, and it may be necessary to call Dev Support.

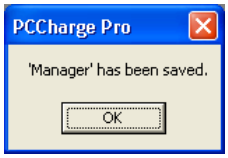


If you did not setup a password previously, Click **Log On**









!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

2) From **PC**Charge, select one of the processors from the drop-down list at the bottom of **PC**Charge.

   Note: All processors in this list support dial-up processing, and many of them support direct TCP/IP processing. If testing of dial-up processing is required, refer to the **PC**Charge Payment Server or **PC**Charge Pro manual for information on setting up a modem with **PC**Charge. Refer to the web-based certification list to determine which processors support dial-up and TCP/IP processing: http://www.pccharge.com/products/pccharge_certs.htm

   Note: Not all processors allow test merchant accounts to be distributed in the **PC**Charge DevKit. This drop-down list represents only a subset of the processors that **PC**Charge supports.

   Note: The sample Chase Paymentech merchant account included with the DevKit will not allow TCP/IP testing as-is. The password must be updated in the Chase Paymentech Advanded Options screen in **PC**Charge prior to using the NetConnect functionality. See the Chase Paymentech entry in the **Test Merchant Account Information** section (see page 544) for more information.

3) Click the Visa / MasterCard icon.

4) Key in a transaction using one of the test credit card numbers that appear in the **Test Credit Cards and Merchant Accounts** section of the **Appendix** (see page 541).

5) Use 12XX as the expiration date (where XX is the last two digits of the current year: e.g., 1208).

6) Use the amount of $1.00 (many processors will decline test transactions if the amount is different than $1.00).

7) Click **Process**. If prompted for additional information, click **No** to each prompt.

8) A result of "CAPTURED" and an auth code will be returned if **PC**Charge is set up properly. (**Note:** A "NOT CAPTURED" may also be returned with some sort of a declined response. This is also a valid response from the processor and means **PC**Charge is set up properly.)

   Note: Not every card works with every test merchant number. If errors or decline messages are returned by **PC**Charge, attempt running transactions using different credit card numbers.

4) Determine Which Integration Method Will Be Used

There are six **PC**Charge integration methods available. This section will help the developer determine which method(s) should be used. The five integration methods are:

1) OCX (ActiveX)
2) DLL (ActiveX)
3) OLE/COM
4) File Method
5) TCP Interface
6) PAYware SIM *

Note: Integration methods 1-3 (OCX, DLL, and OLE/COM) each create a text file that is sent to and processed by **PC**Charge. Thus, all three of these integration methods are essentially sending transaction information to **PC**Charge via the File Method. These three integration methods provide various classes, which include properties and methods to simplify integration. The properties and methods are very similar between the three integration methods, making it relatively easy to migrate from one to the other if needed.

Note: * Information regarding PAYware SIM can be found in the "SIM (Secure Integration Method)" folder in your Active-Charge SDK folder. This is typically found in *"C:\Program Files\Active-Charge SDK\SIM (Secure Integration Method)"*.

OCX Method

The OCX Method should be considered if programming will occur in a visual environment that supports ActiveX technology and all client machines that will process transactions are Windows-based.

The OCX controls are visual wrappers around code to create a flat text input file containing transaction data. Also, the OCX controls handle all of the file I/O (Input / Output) and automatically parse the output file that is created and returned by **PC**Charge. All of these operations are done by setting properties, calling methods, and monitoring events fired by the OCX controls. The use of events allows for asynchronous communication to **PC**Charge.

A choice of two different communication methods is provided with the OCX: file transfer or TCP/IP. If file transfer is selected, the OCX will build a file to be sent to the specified path. If TCP/IP is selected, the OCX will build a string, and send that string through sockets to the IP address specified. To make use of the TCP/IP functionality a dependency file called the DartSecure2.dll will need to be distributed with the OCX file and registered on the client machine. This file is installed with **PC**Charge Pro or Payment Server and can be found in the System32 folder of any machine that has **PC**Charge installed on it.

When using TCP/IP, it is imparitive that the port be opened in PCCharge. To do this, open the **PC**Charge interface and select Setup > Configure System. Click advanced and select "Use TCP/IP Connection". This is the port number setting that will be used in the OCX.

Future Versions of **PC**Charge will have the ability to accept TCP/IP transaction via SSL. This functionality is written into the OCX. For now, this functionality must be turned off. Please consult the OCX documentation for instruction on how to turn this functionality off.

Note: For the File Transfer method, in a client/server environment, the directory in which **PC**Charge resides *must* be shared so that the clients that are generating transactions have read, write, and execute permissions.

DLL Method

If programming will occur in a Windows programming environment that does not support the ActiveX OCX technology, then consider using the DLL Method (`PSCharge.dll`).

`PSCharge.dll` is a wrapper around code to create a flat text input file containing transaction data. Also, `PSCharge.dll` handles all of the file I/O (Input / Output) and automatically parses the output file that is created and returned by **PC**Charge. All of these operations are done by setting properties and calling methods that are provided by `PSCharge.dll`. `PSCharge.dll` performs processing in a synchronous manner.

A choice of two different communication methods is provided with the DLL: file transfer or TCP/IP. If file transfer is selected, the DLL will build a file to be sent to the specified path. If TCP/IP is selected, the DLL will build a string, and send that string through sockets to the IP address specified. To make use of the TCP/IP functionality a dependency file called the DartSecure2.dll will need to be distributed with the DLL file and registered on the client machine. This file is installed with **PC**Charge Pro or Payment Server and can be found in the System32 folder of any machine that has **PC**Charge installed on it.

When using TCP/IP, it is imparitive that the port be opened in PCCharge. To do this, open the **PC**Charge interface and select Setup > Configure System. Click advanced and select "Use TCP/IP Connection". This is the port number setting that will be used in the DLL.

Future Versions of **PC**Charge will have the ability to accept TCP/IP transaction via SSL. This functionality is written into the DLL. For now, this functionality must be turned off. Please consult the DLL documentation for instruction on how to turn this functionality off.

Note: For the File Transfer method, in a client/server environment, the directory in which **PC**Charge resides *must* be shared so that the clients that are generating transactions have read, write, and execute permissions.

Note: If the application will be web-based (e.g., a shopping cart, eCommerce-enabled website, etc.), and will be hosted on a Windows-based web server, consider using the DLL Method. Web-based languages such as ASP and Cold Fusion support referencing DLLs.

OLE/COM Method

If programming will occur in a Windows programming environment that allows directly referencing the exposed classes of an executable, then consider the OLE/COM method of integration.

The OLE/COM Method makes it possible to completely hide the **PC**Charge interface from the user. All aspects from setup and configuration to processing transactions can be done programmatically. It is possible to make calls to set properties or show **PC**Charge forms by accessing the classes exposed through OLE/COM. All processing is done by setting properties, calling methods, and monitoring events. The use of events allows for asynchronous communication to **PC**Charge. The OLE/COM method also supports synchronous communication to **PC**Charge.

Note: In order to use the OLE/COM Method, **PC**Charge must reside on the same computer as the integrated application. The OLE/COM Method will not typically work in a client/server environment (i.e., multi-user or multi-station).

Note: When new versions of **PC**Charge are released (and the OLE/COM Method is used to integrate with **PC**Charge), the integrated application *must* be re-compiled for it to support these new versions. This is a limitation of OLE/COM, not of the **PC**Charge products.

File Method

The File Method is typically used by integrators who prefer to handle the creating, reading, and parsing of flat text files themselves. All of the required file I/O and polling must also be handled by the integrator. The widest variety of programming languages support the File Method. Integration via the File Method is required if programming will occur in an environment that does not support ActiveX, OLE/COM, or socket communications. When integrating via the File Method, the message format used to communicate with **PC**Charge is XML.

The primary benefits to using the File Method are:

- It is operating system independent.
- It can be used in any programming language that supports file I/O.

In addition, the **PC**Charge File Method is very similar to the File Method for RiTA Server. RiTA Server, also designed by VeriFone, Inc., is an enterprise level payment processing product. If the integrator feels that an integration to the RiTA Server product might occur in the future, migration to the RiTA product is easier if the File Method is used when integrating with **PC**Charge.

Note: Although the File Method is operating system independent, **PC**Charge must be running on a Windows machine somewhere on the network—only the client machines may run on other operating systems.

Note: In a client/server environment, the directory in which **PC**Charge resides *must* be shared so that the clients that are generating transactions have read, write, and execute permissions.

Note: If the application will be web-based (e.g., a shopping cart, eCommerce-enabled website, etc.), the File Method may be used. Many Web based languages support file I/O.

TCP Interface

Any integrator using a programming language that supports the use of TCP/IP communication should consider utilizing the TCP Interface method.

The primary advantage of using the TCP Interface method is that it is not file-based. This provides several benefits to the integrator:

- When **PC**Charge is used in a client/server environment, the TCP Interface *does not* require that the **PC**Charge directory be shared on the network. All other integration methods *require* that the **PC**Charge directory be shared on the network in a client/server environment.
- The TCP Interface utilizes the operating system's TCP/IP stack and does not require any additional controls or additional object overhead to perform payment processing.
- The TCP Interface is operating system independent.

In addition, the **PC**Charge TCP Interface method is very similar to the TCP Interface method for RiTA Server. RiTA Server, also designed by VeriFone, Inc., is an enterprise level payment processing product. If the integrator feels that an integration to the RiTA Server product might occur in the future, migration to the RiTA product is easier if the TCP Interface is used when integrating with **PC**Charge.

Note: Although the TCP Interface method is operating system independent, **PC**Charge must be running on a Windows machine somewhere on the network—only the client machines may run on other operating systems. In a client/server environment, the client machines must have TCP connectivity to the Windows-based computer on which **PC**Charge resides.

Note: If the application will be web-based (e.g., a shopping cart, eCommerce-enabled website, etc.), consider using the TCP Interface. Web based languages usually support socket communication.

When integrating via the TCP Method, the message format used to communicate with **PC**Charge is XML.

Charts of Integration Methods

The following chart summarizes several of the characteristics of each integration method. Characteristics that are bold-faced are considered benefits of the integration method.

Integration Method	Platforms supported	Supports client/server communication	Supports Web-based integration	Requires PCCharge directory to be shared for client/server communication	Requires bundling controls with the integrated application	Properties and methods provided / results parsed automatically
OCX	*Windows*	**YES**	*no*	*yes*	*yes*	**YES**
DLL	*Windows*	**YES**	**YES**	*yes*	*yes*	**YES**
OLE/COM	*Windows*	*no*	*no*	*n/a*	*yes*	**YES**
File Method	**All**	**YES**	**YES**	*yes*	**NO**	*no*
TCP Interface	**All**	**YES**	**YES**	**NO**	**NO**	*no*

The following chart lists the various Payment Types and functions that can be integrated along with the integration methods that support them.

Payment Types / Functions	Integration Method
Credit Card Transactions	OCX (Charge.OCX) DLL (PSCharge.dll Charge Class) OLE/COM (PccCharge Class) File Interface TCP Interface
Debit Card Transactions	OCX (Debit.OCX & Device.OCX) DLL (PSCharge.dll Debit Class) OLE/COM (PCCDebit & PccPinPad Classes) File Interface TCP Interface
Check Transactions	OCX (Check.OCX) DLL (PSCharge.dll Check Class) OLE/COM (PccCheck Class) File Interface TCP Interface
EBT Card Transactions	OCX (Debit.OCX & Device.OCX) DLL (PSCharge.dll Debit Class) OLE/COM (PCCEBT & PccPinPad Classes) File Interface TCP Interface
Gift Card Transactions	OCX (GiftCard.OCX) DLL (PSCharge.dll Gift Class) OLE/COM (PCCGiftCard Class) File Interface TCP Interface
Batch Settle/Close	OCX (Batch.OCX) DLL (PSCharge.dll Batch Class) OLE/COM (PccBatch & PccSettle Classes) File Interface TCP Interface
Reporting	OCX (Charge.OCX) DLL (PSCharge.dll Charge Class) OLE/COM (PccCharge Class) File Interface TCP Interface
Configuration	OLE/COM (Various Classes)
Utilities (Transaction Inquiry, etc.)	OCX (Charge.OCX) DLL (PSCharge.dll Charge Class) OLE/COM (PCCDebit Class) File Interface TCP Interface

5) Determine Which PCCharge Product(s) Will Be Supported

The DevKit includes copies of both **PC**Charge Payment Server and **PC**Charge Pro. Both products are provided so that the integrator can "kick the tires" on each product while integrating and testing. This section is designed to assist integrators in deciding which **PC**Charge product(s) to support in their applications.

Common Questions

*"What is the difference between **PC**Charge Payment Server and **PC**Charge Pro?"*

"Which product should I integrate with?"

*"What are the advantages of **PC**Charge Payment Server over **PC**Charge Pro and vice versa?"*

These are common questions that are often asked by integrators and merchants. Unfortunately, there is not one correct answer to each of these questions. However, it is important for integrators and merchants to know the differences between the two products so that they can determine which one (or both) will meet their needs.

Differences and Similarities

The following are the differences between PCCharge Payment Server and PCCharge Pro:

Directory / Executable

- **PC**Charge Pro is installed in the `c:\Program Files\PCCW` directory by default. **PC**Charge Pro's executable is named `Pccw.exe`.
- **PC**Charge Payment Server is installed in the `c:\Program Files\Active-Charge` directory by default. **PC**Charge Payment Server's executable is named `Active-Charge.exe`.

GUI (Graphical User Interface)

- **PC**Charge Pro has a full-featured GUI that loads automatically.
- **PC**Charge Payment Server has a hidden GUI. The GUI can be accessed manually by the user by double-clicking a "dollar sign" icon in the system tray.

Error Message Suppression

- **PC**Charge Pro *does not* support Error Message Suppression. If the **PC**Charge Pro GUI pops up an error message, processing will halt until the error message is cleared manually by an end-user.
- **PC**Charge Payment Server supports Error Message Suppression. If the **PC**Charge Payment Server GUI is hidden (the default setting), any error messages generated by **PC**Charge Payment Server will automatically be logged to an error file, thus allowing payment processing to proceed normally.

Customer Database and Recurring Billing

- **PC**Charge Pro has a Customer Database and Recurring Billing feature available via the GUI. **Note:** These features *cannot* be integrated.
- **PC**Charge Payment Server does not support the Customer Database or Recurring Billing.

Reporting (in the GUI)

- Both **PC**Charge Pro and **PC**Charge Payment Server have reporting capabilities available via the GUI. However, **PC**Charge Pro has more reports than **PC**Charge Payment Server. **PC**Charge Payment Server has fewer reports, mainly because all reports related to the Customer Database and Recurring Billing have been omitted.

Batch Management

- Both **PC**Charge Pro and **PC**Charge Payment Server support Batch Management features. These features cannot be integrated.
    - With PCCharge Payment Server or **PC**Charge Pro, these features can be accessed from the GUI via the "Batch" menu or via a separate program that is included on the PCCharge Payment Server or **PC**Charge Pro installation CD.

Note: For more information on the Customer Database, Recurring Billing, Reporting (in the GUI), and Batch Management please refer to the **PC**Charge Pro or **PC**Charge Payment Server manuals.

The following are the similarities of PCCharge Payment Server and PCCharge Pro:

Transaction Processing Engine

- *Identical* in both **PC**Charge Payment Server and **PC**Charge Pro

API (Application Programming Interface)

- *Identical* in both **PC**Charge Payment Server and **PC**Charge Pro

Processor Certifications

- *Identical* in both **PC**Charge Payment Server and **PC**Charge Pro

Database (Microsoft Access database)

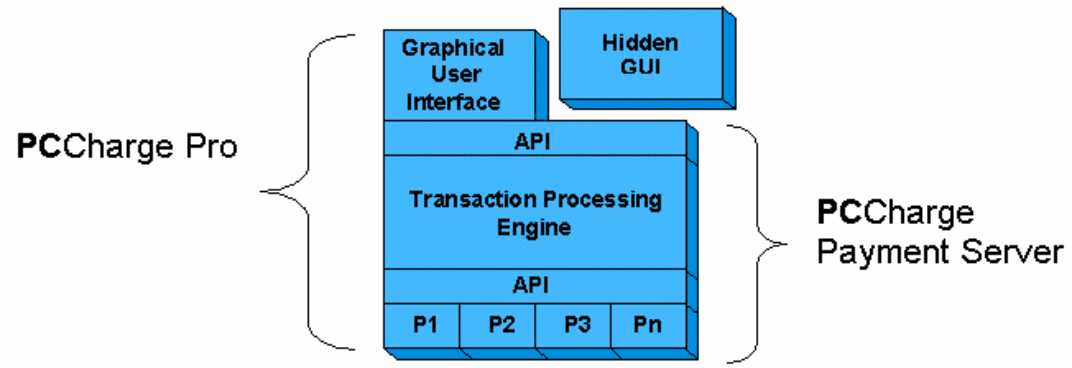
- *Identical* in both **PC**Charge Payment Server and **PC**Charge Pro

Multi-User and Multi-Merchant Number Support

- *Identical* in both **PC**Charge Payment Server and **PC**Charge Pro

Illustration of Basic Differences and Similarities

The following graphic shows the basic differences and similarities of **PC**Charge Payment Server and **PC**Charge Pro:



Description of the diagram from the bottom up:

P1, P2, P3, Pn and **API:**

These sections represent the processor interfaces (aka Processor Certifications) that VeriFone, Inc. has coded into the **PC**Charge products. The processors provide VeriFone, Inc. with an API to code to in order for **PC**Charge to communicate with the processor's network. Notice that there is only one set of processor interfaces. The processor interfaces are identical for both **PC**Charge Payment Server and **PC**Charge Pro. **Note:** When a processor interface is added and certified or updated by VeriFone, Inc., it is added or updated in all of the **PC**Charge products.

Transaction Processing Engine and **API:**

These sections represent the main **PC**Charge product and interface. This includes the **PC**Charge API, database, utilities, etc. **Note:** The API is *identical* for both **PC**Charge Payment Server and **PC**Charge Pro.

Graphical User Interface:

PCCharge Pro includes a full-featured GUI. This GUI appears when **PC**Charge Pro is started on a computer. When a user does transaction processing via the **PC**Charge Pro GUI, the GUI actually communicates to the Transaction Processing Engine via the API using the File Method of integration. In essence, the **PC**Charge Pro GUI is an integration to the Transaction Processing Engine.

Hidden GUI:

PCCharge Payment Server also includes a GUI. However, this GUI is not loaded by default and does not appear when **PC**Charge Payment Server is started on a computer. After **PC**Charge Payment Server has loaded, the user has the option of loading and viewing the GUI if they wish. Like **PC**Charge Pro's GUI, the **PC**Charge Payment Server GUI also communicates to the Transaction Processing Engine via the API using the File Method of integration.

The Choice

Now that the differences and similarities have been outlined, merchants and integrators should now consider which product(s) will be used and supported.

PCCharge Payment Server

The following features of **PC**Charge Payment Server make it an appealing choice for most integrators:

- Hidden GUI
- Error Message Suppression

The Hidden GUI allows the integrator to hide the **PC**Charge Payment Server interface from the merchant completely—making it appear as if all transaction processing is occurring solely in the third-party integration. This is desirable to many integrators.

Also, Error Message Suppression is ideal if **PC**Charge Payment Server is running in an unattended environment (i.e., running in a data center, running in an office behind locked doors, or processing payments on a 24/7 eCommerce website, etc.). If an error message was to pop up in this type of environment, and no operator was immediately available to clear the message, payment processing would halt if Error Message Suppression was not available.

Because of the above features, most integrators choose to integrate and support **PC**Charge Payment Server.

PCCharge Pro

Many integrators and merchants have found that they require some (or all) of the following features of **PC**Charge Pro that are not available in **PC**Charge Payment Server:

- Customer Database and Recurring Billing (only available via the GUI)

If these features are required, it may be necessary to integrate to and support **PC**Charge Pro. Keep in mind, though, that **PC**Charge Pro *does not* support the Hidden GUI and Error Message Suppression features of **PC**Charge Payment Server.

Both

Based on past experience and customer feedback, VeriFone, Inc. has found that most integrators decide to support both products, and then allow their merchants to decide which product works best based on their needs. Because of this, both **PC**Charge Pro and **PC**Charge Payment Server are included with the DevKit.

It is suggested that developers integrate to and support both products. Because the APIs are identical, an integration to **PC**Charge Payment Server will also work with **PC**Charge Pro and vice versa. If using the TCP Interface method, there are no differences in the integration whatsoever. With the other integration methods, the main difference between integrating with **PC**Charge Pro and **PC**Charge Payment Server is the target directory. An integration that supports both products would provide a user-selectable option to select the target directory for **PC**Charge rather than hard-coding the target directory.

6) Review Payment Processing Basics and Integration Information and Settings

The next chapter, **CHAPTER 3 -- Payment Processing Basics**, includes information about the basics of processing payments and a description of each payment type supported by **PC**Charge. This chapter should be reviewed by integrators before any coding begins. The integration of **PC**Charge will go much smoother if the integrator has an understanding of how transaction processing occurs on a day-to-day basis.

It is also recommended that integrators review the information contained in **CHAPTER 4 -- Integration Information and Settings** (see page 54). This chapter includes warnings, integration tips, and guidelines that should be followed while integrating, and includes important information about Users, Merchant Accounts, Timeouts, Follow On transactions, etc.

After **PC**Charge has been set up and these chapters have been reviewed, the integration may begin. Refer to the **DevKit Constants** section (see page 143) and **CHAPTER 6 -- PCCharge Integration Methods** (see page 157) for information on integrating payment processing using **PC**Charge.

CHAPTER 3 -- Payment Processing Basics

Credit Card Processing

This section provides a general overview of credit card processing and how processing differs between Host based and Terminal based Processors.

Host based and Terminal based Processors

The most important concept for integrators to understand is what Host based and Terminal based processors are and what the differences are between them. Depending on which type of processor will be used by the merchant, certain tasks must be performed by the merchant either directly from the **PC**Charge GUI or from the integrated application to enable them to receive funds from their credit card transactions.

Processing credit card transactions is a two-step process. When a credit card sale transaction is performed by the merchant, the customer's "limit to buy" on their credit card account is reduced and the transaction is placed in a batch. A batch is defined as a collection of transactions that are approved but have not yet been submitted for end of day settlement. Before funds can be deposited into the merchant's bank account, a second step, re-transmission, must occur.

When using **PC**Charge, the re-transmission process is called Batch Settlement or Batch Close. Terminal based processors have batches that are settled, and Host based processors have batches that are closed. Regardless, all credit card transactions in a batch have to be re-submitted to the processing company. With certain processing companies, this step takes place automatically. With others, the process must be performed manually.

So, in this context, there are two types of processing companies:

1) **Terminal based** – If the merchant is using a Terminal based processor, **PC**Charge stores the batch of transactions in a file in the **PC**Charge directory. This file is commonly known as a Settlement File. With Terminal based processors, batch settlement must be done manually, usually at the end of each business day. This procedure can be performed, by the merchant, from the **PC**Charge GUI. This procedure can also be accessed programmatically by the integrator.

2) **Host based** - With a Host based system, the batch is stored on the credit card processor's computer system. With a Host based system, merchants can be set up in one of two ways:

    a. **Auto (Time Initiated) Close** – At a certain time during the day, the Host based system will scan its computer system. If a merchant has an open batch of transactions, the system will automatically close the batch. Usually, the merchant can specify the time when the batch will be closed each day. If a merchant is set up for Auto Close with their Host based processor, there are no steps needed to initiate the batch close process.
    b. **Manual Close** – Credit card transactions will sit in an open batch indefinitely. The merchant is responsible for indicating to the Host based processor that the batch should be closed. This step can be performed by the merchant from the **PC**Charge GUI. This procedure can also be accessed programmatically by the integrator.

WARNING: Without settling or closing the batch, the merchant will not receive any funds from credit card transactions!

Credit Card Transactions

There are several types of credit card transactions. The different types of transactions are referred to as actions in **PC**Charge. The following is a list of the various actions with general descriptions.

Sale - This action reduces the cardholder's limit to buy, and places the transaction in the open batch. This action is commonly used in retail and restaurant environments.

Void Sale - This action removes a sale transaction from the open batch. No funds will be deposited into the merchant's bank account at settlement/close. The Void Sale action is typically used for same day returns or to correct cashier mistakes. This action can only be performed before the batch is settled/closed (this usually means the action has to be performed on the same day as the sale).

Credit - This action is used to refund money to the cardholder. This action is typically used after the batch that contains the Sale (or Post-Authorization) transaction has been settled or closed. This action will increase the cardholder's limit to buy once the batch containing the credit has been settled.

Void Credit - This action removes a credit transaction from the batch. This action can only be performed before the batch is settled/closed (this usually means the action has to be performed on the same day as the credit). This transaction is not available with all processing companies. If the Void Credit action is not available, use the Void Sale action.

Pre-Authorization - This action only reduces the cardholder's limit to buy. It does not place a transaction in the open batch. A Pre-Authorization can be considered the first half of a sale. A Pre-Authorization reduces the limit to buy for only a predetermined amount of time, usually 7-10 days. The credit card's issuing bank determines the amount of time. To place the transaction in the open batch, a follow-on transaction (called a Post-Authorization) must occur. This action is commonly used in MOTO (Mail Order / Telephone Order) and eCommerce environments.

Voice-Authorization – A Voice-Authorization is similar to a Pre-Authorization, except that it is not a **PC**Charge action. A Voice-Authorization is a way for a merchant to receive an approval and an authorization code from their processing company via a telephone operator or an automated system. This type of authorization is necessary if the processing company's Internet interface or modems are down, or if the merchant has experienced a power failure, computer crash, or other issue that does not allow them to process electronically. A Voice-Authorization may also need to occur if a credit card transaction is flagged as fraudulent by the credit card issuer. In order to place a Voice-Authorization in the open batch, a follow-on transaction, called a Post-Authorization, must occur.

Post-Authorization - This action places an approved Pre-Authorization transaction into the open batch. This action can be considered the second half of a sale. This follow-on transaction must occur before a Pre-Authorization can be settled/closed. This Post-Authorization may also be used to place an approved Voice-Authorization in the batch.

Void Post-Authorization - This action removes a Post-Authorization transaction from the open batch. This action can only be performed before the batch is settled/closed (this usually means the action has to be performed on the same day as the Post-Authorization). This transaction is not available with all processing companies. If Void Post-Authorization is not available, use the Void Sale action.

Commercial Card Sale - This action reduces the cardholder's limit to buy, and places the transaction in the open batch. This action is similar to a standard credit card sale, but is

typically used if the card tendered is a procurement, purchasing, business, government, or commercial card. Two additional values, the Tax amount and customer code, must be passed with this type of card in order for the merchant to qualify for the lowest transaction rate. **Note:** Global East (NDC), terminal based, requires the customer code be all upper case.

Commercial Card Credit - This action is typically used after the batch that contains the Procurement Card Sale (or Procurement Card Post-Authorization) transaction has been settled or closed. This action is similar to a standard credit card credit, but is typically used if the card tendered is a procurement, purchasing, business, government, or commercial card. This action will increase the cardholder's limit to buy once the batch containing the credit has been settled. Two additional values, the Tax amount and customer code, must be passed with this type of card in order for the merchant to qualify for the lowest transaction rate. **Note:** Global East (NDC), terminal based, requires the customer code be all upper case.

Commercial Card Post-Authorization - This action places an approved Pre-Authorization transaction into the open batch. This action is similar to a standard credit card Post-Authorization, but is typically used if the card tendered is a procurement, purchasing, business, government, or commercial card. This action can be considered the second half of a sale. This follow-on transaction must occur before a Pre-Authorization can be settled/closed. This Post-Authorization may also be used to place an approved Voice-Authorization in the batch. Two additional values, the Tax amount and customer code, must be passed with this type of card in order for the merchant to qualify for the lowest transaction rate. **Note:** Global East (NDC), terminal based, requires the customer code be all upper case.

Sale with Gratuity – Used only in a restaurant environment, this action allows a server to authorize the amount of the meal plus the gratuity and place the entire amount in the batch. When settled, the amount plus the gratuity is deducted from the cardholder's account.

Gratuity – Used only in a restaurant environment, this action adds or adjusts the gratuity amount on an existing Sale transaction. This action must be performed prior to batch settlement/close.

Note: There is no action to void a Pre-Authorization. This is because Pre-Authorization cannot be voided.

A Normal Credit Card Processing Day

This section discusses a normal day of processing credit card transactions. This section assumes that merchants will have access to the reports provided in the **PC**Charge GUI and that these reports can be compared to reports available in the integrated application.

Processing credit card transactions consists of six steps:

1) By performing the first credit card transaction of the day, the batch is opened.
2) Run transactions throughout the day.
3) After processing has been completed for the day, it is time to prepare for settlement:

    a. Look at the **PC**Charge Daily Transaction Summary Report for the day.
    b. Compare the **PC**Charge Daily Transaction Summary Report with summary data provided by the integrated application.
    c. Does the report look correct and do the totals match up with the integrated application's totals?

4) If the reports indicate discrepancies, use the **PC**Charge Credit Card Detail and Batch Pre-Settle Reports (if the processor is Terminal based) to investigate and resolve any discrepancies.
5) If the processor is Host based, settlement will occur automatically.
6) If the processor is Terminal based, settlement must be done manually. Start the settlement process now.

Repeat these six steps each day of processing credit card transactions.

Credit Card Transaction Rates

It is important for integrators to understand credit card transaction rates. The card associations (Visa, MasterCard, etc.) require certain data to be submitted with each transaction in order for the merchant to qualify for the lowest transaction fees. While integrating, it is important that the application be designed to allow merchant to pass any and all "rate-qualifying" data while performing transactions. Below is a general list of information that should be submitted in order for the merchant to receive the best rates:

Best Rates -- Card Swiped
Fair Rates -- Card Not Present & Ticket Number & Street Address & Zip Code
Worst Rates -- Card Not Present & No Address Information

Transaction rates are based on risk. They vary based on the size of the merchant organization and that merchant's ratings, markets, and how transactions occur. For instance, a very large traditional walk-in, multi-store retail organization with check out lanes will have the lowest rates. A new mail order or eCommerce merchant, with no retail facilities and no face-to-face or "card present" transactions, will typically have to pay the highest transaction rates.

Based on some of these variables, each transaction can have qualifiers for higher or lower rates. A properly written integrated application can help MOTO or eCommerce merchants get better rates (and reduce fraud) by using the Address Verification Service (AVS). AVS captures ZIP codes and addresses of the cardholder's billing address. AVS information is then compared to the cardholder's ZIP code and address that the card issuer has on file. Address and ZIP code mismatches help the merchant to decide whether or not they wish to go through with the transaction, thus helping to reduce fraud.

Commercial Cards Note

If a merchant will accept procurement, purchasing, business, government, or commercial cards, the integrated application must be written to allow merchants to pass the tax amount and customer code for these types of cards. This information helps the cardholder by allowing them to receive this detail on their monthly statements and also benefits the merchant by allowing them to qualify for the lowest rates when accepting these types of cards. Commercial card transactions that do not include this additional information will be downgraded. **Note:** Global East (NDC), terminal based, requires the customer code be all upper case.

Debit Card Processing

Debit cards were designed to be used more like a check than a credit card. Unlike credit cards with a line of credit, the debit card causes the amount of the purchase to be deducted from the debit cardholder‘s checking account. Like ATM cards, debit cards can be used to withdraw money from a bank account, debiting the account immediately. However, unlike debit cards, ATM-only cards cannot be used as cards for purchases.

Some debit cards, called “check cards”, can be used as debit, ATM, and credit cards. If used as a debit or ATM with the PIN, transactions are considered to be online and debited immediately from the cardholder’s account. Debit check cards used as credit cards are considered offline transactions and must go through the credit settlement process; thus, taking several days to reach the bank and be debited from the cardholder’s account.

Although merchants accept either credit cards or debit check cards, most merchants are unaware of the distinction by just looking at the card. However, credit cards normally have a percentage of the transaction as the fee while debit transactions are normally a smaller set fee.

There are three important points to know about processing debit transactions:

1) There are two types of debit card transactions:

    a. **Online** - Online debit refers to debit card processing that requires a debit card to be swiped and a PIN to be entered when processing a transaction.
    b. **Offline** - Offline debit refers to debit card processing that allows a debit card to be swiped or keyed in and PIN entry does not occur. Essentially, these card are processed as normal credit cards. A debit card supports offline debit if it has a VISA or MasterCard logo on the front of it. Cards that support offline debit are commonly referred to as “check cards”. Refer to the previous section for general information on processing credit cards.

2) Online Debit Transactions may only be performed in a Retail or "face to face" environment. Mail Order/Telephone Order, and eCommerce businesses cannot perform online debit transactions.
3) Online debit transactions require that the card be swiped and that the customer enters their PIN on a PINpad. Each cash register or computer that will support Online debit transactions must have a card swipe device and PINpad connected to it.

As with credit card processing, debit processing is a two-step process. Debit processing requires re-transmission of information, referred to as Closing. Typically, Debit processors are Host based. That means the information to be re-transmitted is stored on the host computer systems or the processor's computer system. Merchants can be set up one of two ways:

> **Time Initiated Close** – At a certain time during the day (usually specified by the merchant), a Host based system will scan its database. If a merchant has an open batch (transactions that have not been deposited), the host system will automatically close the batch.
> **Manual Close** – Authorized transactions will sit in an open batch indefinitely. The merchant is responsible for indicating to the host processor that the batch should be closed.

If the merchant is set up for Manual Close, they will need to perform a Close to complete their debit transactions.

If the merchant is processing credit card transactions as well as debit transactions, the debit transactions will be closed at the time that the credit card transactions are settled.

Debit Card Transactions

There are several types of debit card transactions. The different types of transactions are referred to as actions in **PC**Charge. The following is a list of the actions with general descriptions.

Sale - This action deducts funds from the cardholder's bank account.

Return - This action deposits funds into the cardholder's bank account.

Void – This action voids a Sale or Return.
Note: Not all debit card processors in PCCharge support Debit Void functionality

A Normal Debit Card Processing Day

This section discusses a normal day of processing Debit transactions. This section assumes that merchants will have access to the reports provided in the **PC**Charge GUI and that these reports can be compared to reports available in the integrated application.

Processing Debit transactions consists of six steps:

1) By performing the first Debit card transaction of the day, the batch is opened.
2) Run transactions throughout the day.
3) After processing has been completed for the day, it is time to prepare for settlement:
    a. Look at the **PC**Charge Daily Transaction Summary Report for the day.
    b. Compare the **PC**Charge Daily Transaction Summary Report with summary data provided by the integrated integration.
    c. Does the report look correct and do the totals match up with the integrated application’s totals?
4) If the reports indicate discrepancies, use the **PC**Charge Debit Summary report to investigate and resolve any discrepancies.
5) If the processor is Host based, settlement will occur automatically.
6) If the processor is Terminal based, settlement must be done manually. Start the settlement process now.

Repeat these six steps each day of processing debit card transactions.

Check Processing

Check Guarantee and Check Verification services do not involve an electronic transfer of funds. The merchant performs Check Verification to determine that if check writer has an account and if the check writer has any current outstanding (bounced) checks. Check Guarantee services extend a guarantee that the merchant will get his money, even if the check bounces. Normally, a higher fee is charged for guarantee service (usually a fee similar to credit card processing fees).

Check Verification and Check Guarantee are one-step processes.

There are three types of Check transactions:

1) **Check Verification** - Verification allows the merchant to verify that the check writer has a checking account and does not have any outstanding bad checks.
2) **Check Guarantee** - Guarantee allows the merchant to verify that the check writer has an account and guarantees that the funds are available.
3) **Check Conversion** – Conversion allows the merchant to use the MICR information on a check to electronically deposit funds. This transaction eliminates the need to deposit a paper check.

Check Conversion

Check Conversion is a process by which a checking account is debited electronically.

Check conversion is a two-step process. As with credit cards, there is a secondary transmission of information needed to complete a transaction. The second step is called Truncation Close. The important thing to remember is that without re-transmission of the check information, the merchant will not receive their money. Every day that a merchant performs truncations, they should perform a Truncation Close after all transactions are complete.

Check Transactions

There are a few different types of check conversion transactions. The following is a list of actions with general descriptions:

Verify – This action allows the merchant to verify that a checking account exists for the customer and guarantees that the amount of the transaction is available. This action also allows the merchant to perform the first half of a sale transaction. This action does not make information available for re-transmission.

Sale – This action reduces the balance of the customer's checking account. A sale actually performs two functions. First, a sale will Verify / Guarantee a check. Second, it will make the transaction available for re-transmission.

Void – This action removes a Sale or Force transaction from the re-transmission information. The transaction will be deleted; no funds will be received from this transaction. The Void Sale action is used to correct mistakes and on same day returns. This action can only be performed before re-transmission.

Force – This action makes a verified check transaction available for re-transmission. A Verify followed by a Force is equivalent to a Sale.

A Normal Check Conversion Processing Day

This section discusses a normal day of processing check transactions. This section assumes that merchants will have access to the reports provided in the **PC**Charge GUI and that these reports can be compared to reports available in the integrated application.

Processing Check transactions consists of six steps:

1) By performing the first check conversion transaction of the day, the batch is opened.
2) Run transactions throughout the day.
3) After processing has been completed for the day, it is time to prepare for settlement:
    a. Look at the **PC**Charge Daily Transaction Summary Report for the day.
    b. Compare the **PC**Charge Daily Transaction Summary Report with summary data provided by the integrated integration.
    c. Does the report look correct and do the totals match up with the integrated application's totals?
4) If the reports indicate discrepancies, use the **PC**Charge Check Summary report to investigate and resolve any discrepancies.
5) If the processor is Host based, settlement will occur automatically.
6) If the processor is Terminal based, settlement must be done manually, start the settlement process now.

Repeat these six steps each day of processing check conversion transactions.

EBT Processing

Electronic Benefits Transfer (EBT) is a way of issuing and processing certain benefits electronically. The government issues Food Stamps and aid to families with dependent children on EBT cards that resemble credit or debit cards. The government is issuing Social Security payments, Disability payments, and many other government issued payments on these EBT cards. With EBT processing, these payments can processed just as a debit card would be processed. More and more state governments and the federal government will be moving to this form of payment processing to reduce paper work and control fraud.

With EBT transactions, keep these two simple rules in mind.

1) EBT transactions can only be performed in a Retail or "face to face" environment. If the merchant is a Mail Order type business, they cannot perform EBT transactions.
2) EBT transactions require that the merchant have a card swipe and PINpad attached to their computer.

As with debit card processing, EBT processing is a two-step process. EBT processors are typically Host based. As with every Host based System, merchants may have the ability to be set up for either Time Initiated (Automatic) Close or Manual Close.

There are four types of EBT card transactions. The types are based on the kind of benefit being processed. For instance, if the merchant will process a transaction with food stamps, they will want to use the transaction type called Food Stamps. The reason for the different types of transactions is that there are different rules governing each type of benefit.

1) **Cash** – Use this transaction type to process a transaction with an EBT card that was issued for a Social Security payment.
2) **Food Stamp** – Use this transaction type to process a food stamp transaction to deduct money from the EBT card.
3) **Food Stamp Credit** – Use this transaction type to process a food stamp transaction to credit money back onto the EBT card.
4) **Account Inquiry** – This transaction allows the merchant to inquire into a customer's account. The merchant is able to dial into the EBT processing company and view a customer's account. This transaction is not specific to a benefit type. Merchants will be able to perform an inquiry, regardless of the type of benefit.

In the next four sections, we will break down each transaction type and describe each action available.

Cash EBT Transactions

All transactions that are not food stamp related should be processed as Cash EBT Transactions. Cash EBT transactions are very similar to debit transactions because customers can receive cash back from transactions.

There are three types of cash transactions. To maintain consistency, they will be referred to as actions.

1) **Withdrawal** – This action deducts from the cardholder's limit to buy. Money will be deducted from the account.
2) **Post** – This action makes an approved Voice Authorization transaction available for re-transmission. This action is the second half of a withdrawal.
3) **Void** – This action removes a withdrawal transaction from the re-transmission information. The transaction will be deleted. Merchants will not receive funds from this transaction. Use the

Void action to correct mistakes and on same day returns. This action has to be performed on the same day as the original transaction.

Food Stamp EBT Transactions

When performing a sale transaction using food stamp benefits, use the Food Stamp EBT Transaction.

1) **Withdrawal** – This action deducts from the cardholder's limit to buy. Money will be deducted from the account.
2) **Post** – This action makes an approved Voice Authorization transaction available for re-transmission. This action is the second half of a withdrawal.
3) **Void** – This action removes a withdrawal transaction from the re-transmission information. The transaction will be deleted. Merchants will not receive funds from this transaction. Use the Void action to correct mistakes and on same day returns. This action has to be performed on the same day as the original transaction.

Food Stamp Credit EBT Transactions

When performing a return transaction using food stamp benefits, use the Food Stamp Credit EBT Transactions.

1) **Withdrawal** – This action increases the cardholder's limit to buy. Funds will be credited back to the account.
2) **Post** – This action makes an approved Voice Authorization transaction available for re-transmission. This action is the second half of a withdrawal.
3) **Void** – This action removes a withdrawal transaction from the re-transmission information. The transaction will be deleted. Use the Void Sale action to correct mistakes. This action has to be performed on the same day as the original transaction.

Account Inquiry EBT Transaction

This transaction type is intended only as a maintenance function. It is used when merchants need to verify that there is a certain amount in a customer EBT account. Merchants simply enter the card number and expiration.

A Normal Day of Processing EBT Transactions

This section discusses an average day of processing EBT transactions. This section assumes that merchants will have access to the reports provided in the **PC**Charge GUI and that these reports can be compared to reports available in the integrated application.

Processing EBT transactions consists of six steps:

1) By performing the first EBT transaction of the day, the batch is opened.
2) Run transactions throughout the day.
3) After processing has been completed for the day, it is time to prepare for settlement:

    a. Look at the **PC**Charge Daily Transaction Summary Report for the day.
    b. Compare the **PC**Charge Daily Transaction Summary Report with summary data provided by the integrated application.
    c. Does the report look correct and do the totals match up with the integrated application’s totals?

4) If the reports indicate discrepancies, use the **PC**Charge EBT Summary report to investigate and resolve any discrepancies.
5) If the processor is Host based, settlement will occur automatically.
6) If the processor is terminal based, settlement must be done manually. Start the settlement process now.

Repeat these six steps each day of processing EBT transactions.

Gift Card Processing

A typical gift card program allows the merchant to offer their customers' full-color, plastic electronic gift cards instead of traditional paper gift certificates. Customers purchase these cards (using cash, check, or credit card), and the card is activated on the spot with a simple 'swipe' through the card reader. This activation records the starting dollar amount in a central location. The card's magnetic strip can also identify the cardholder, card number, and merchant location for which the card has been issued.

Gift cards are used just like a credit or debit card to purchase goods or services, and they are authorized using a system similar to that of the credit card processing systems. The dollar value of the purchase is subtracted from the account total in the database, the purchase is logged, and the new card value is recorded.

A Normal Day of Gift Card Processing

This section discusses a normal day of processing gift card transactions. This section assumes that merchants will have access to the reports provided in the **PC**Charge GUI and that these reports can be compared to reports available in the integrated application.

Note: Gift Card Processors are Host based. Gift Cards do not require settlement.

Processing Gift Card transactions consists of three steps:

1) Run transactions throughout the day.
2) After processing has been completed for the day, it is time to reconcile:

    a. Look at the **PC**Charge Daily Transaction Summary Report for the day.
    b. Compare the **PC**Charge Daily Transaction Summary Report with summary data provided by the integrated application.
    c. Does the report look correct and do the totals match up with the integrated application's totals?

3) If the reports indicate discrepancies, use the **PC**Charge Gift Card report to investigate and resolve any discrepancies.

Typically, these three steps should be repeated each day of processing Gift Card transactions.

CHAPTER 4 -- Integration Information and Settings

Warnings, Tips, and Guidelines

The following are warnings, integration tips, and guidelines to follow while integrating **PC**Charge. It is suggested that integrators adhere to all guidelines to keep integrated applications running smoothly and to keep merchants compliant with all payment processing rules and regulations.

- **XML Message Format** – It is strongly recommended that the XML message format is used when integrating to **PC**Charge. Starting with **PC**Charge version 5.6.0, the **PC**Charge products were updated to support the XML message format. The XML message format is recommended because:

    - Transaction requests and responses are formatted in a flexible, variable-width message format when using XML—thus allowing any number of fields to be passed back and forth to **PC**Charge. The legacy INP message format is proprietary and transaction requests are restricted to a fixed width (256 characters). When using the INP message format, many transaction requests require the re-use of fields.
    - All new features and enhancements to **PC**Charge require the use of the XML message format. Some of these features include Follow-On transactions, Transaction Inquiries, Gratuity Adjustments, and Canadian debit.
    - Starting with the 5.7 release of the DevKit, all documentation assumes that the XML message format is being used by the integrator.
    - When using the legacy INP format, some PCCharge reports may not update the transaction status for follow-on transactions. This is a limitation of the legacy INP format. This can be remedied by using the XML message format.
    - **The INP message format will eventually be phased out.**

    If an existing integration was written using the INP message format, it is highly recommended that the integration be updated to use the XML message format. The XML message format is available in all five integration methods. To enable the use of the XML message format when using the OCX, DLL, or OLE/COM methods of integration, set the message format parameter to "3" when calling the Send method.

    OCX / DLL: `.Send 3`
    OLE/COM: `.Send , 3`

    To use the XML message format when using the File Method or TCP Interface, simply follow the **File Layout Specifications** outlined in the **File Method** section (see page 483).

!	**WARNING:** VeriFone, Inc. takes security seriously therefore we have enhanced our integration methods to enhance security for sensitive data. If using **File Method** integration to PCCharge we strongly suggest you consider migrating towards a more secure integration method such as **TCP/IP SSL** socket integration.

    Note: The legacy INP message format is selected as default in the OCX, DLL, and OLE/COM Methods of integration for backwards compatibility reasons. If the message format parameter is not set to "3" when calling the Send method, the various controls and classes will default to the INP message format.

Note: Older copies of the DevKit that outline the INP message format are available for integrators upon request. Contact VeriFone, Inc. at 800-725-9264 to request a copy of an older DevKit manual.

- **Cardholder Information Security Program (CISP).** It is extremely important to always adhere to CISP guidelines while integrating. The following are the most pertinent to the integrator:

    - **Magnetic Stripe Data** – The integrated application should not store or print credit card magnetic stripe data. Make sure that all printouts, files, databases, logfiles, etc. do not contain this information. Refer to the **Important Security Notice** (see page 9) for more information.

    - **Card Verification Data** – The integrated application should not store or print the CVV2, CVC2, or CID data from the back or front of credit cards. Make sure that all printouts, files, databases, logfiles, etc. do not contain this information. Refer to the **Important Security Notice** (see page 9) for more information.

    - **Credit Card Numbers / Expiration Dates** – If the integrated application will store credit card account numbers and expiration dates, these values must be encrypted in all files, databases, logfiles, etc. Refer to the **Important Security Notice** (see page 9) for more information.

- **Receipt Printing** – If the integrated application will print receipts, it is a good idea to provide an option in the integrated application that allows masking of the digits of the credit card number and the removal of the expiration date on a receipt that will be given to a customer. The integrated application should not print the full card number and expiration date on a customer receipt. Many states have laws regarding the information that is printed on receipts. The merchant should familiarize themselves with the laws that pertain to them.

- **Transaction Processing** - Because of the single-threaded architecture of **PC**Charge, it is important that the integrated application refrains from submitting (or importing) transactions while **PC**Charge is performing the following functions:

    - Settlement / Batch Close
    - Database Repair / Compact
    - Backup or Restore of configuration files and database
    - Database Transaction archive

  If the integrated application submits transactions while any of the above functions are taking place, database corruption and/or lost transactions may occur. If integrating with the OCX, DLL, or OLE/COM methods, always use the `PccSysExists` method to check if these functions are being performed prior to submitting a transaction. If `PccSysExists` returns `TRUE`, do not submit the transaction. See **CHAPTER 6 – PCCharge Integration Methods** for more information on using the `PccSysExists` method. If using the File Method, always check for the presence of the `SYS.PCC` file in the **PC**Charge directory prior to submitting the transaction. If this file appears in the directory, the integrated application should not submit the transaction. The contents of the `SYS.PCC` file can be checked to determine what function is being performed by **PC**Charge. Refer to the section **SYS.PCC Codes and Descriptions** for more information. If using the TCP Interface, the `SYS.PCC` check is not necessary. When using the TCP Interface, if **PC**Charge is "busy" performing tasks, it will either queue up the transaction and process it once the task has been completed (such as in batch settlement) or it will return a "transaction cancelled, system busy" message if other tasks are running.

- **Direct Database Access** - If an integrated application will access the **PC**Charge database (`pccw.mdb`) directly, it is important that the integrated application refrains from accessing

the database while transactions are being processed or while batches are being settled. Because of the limitations of Microsoft Access, if an integrated application tries to access the database (even in read-only mode), **PC**Charge may not be able to read or update records in the database. This has been known to cause database corruption, settlement errors, and lost transactions.

- **Remote Access -** We currently **do not** support accessing **PC**Charge via Windows Remote Desktop, Virtual Private Network (VPN), or any other remote interface application. Programs like these allow **PC**Charge to initiate multiple instances of itself, causing lost transactions, duplicate charges, and database corruption. Instead, we recommend that you use either **PC**Charge Client or **PC**Charge DevKit (via integration) to remotely access **PC**Charge.

- **Terminal Service environment - PC**Charge is not supported in a Terminal Service environment. Terminal Services allows multiple instances of the same application to run simultaneously. **PC**Charge cannot be supported in such an environment. Running multiple instances of **PC**Charge often results in duplicate charges and lost transactions.

- **PCCharge GUI** - Merchants should not attempt to process transactions or run reports directly from the **PC**Charge GUI while the integrated application is processing transactions or settling batches. If the merchant wishes to process transactions or run reports directly with **PC**Charge (instead of through the integrated application), it is highly recommended that they use the **PC**Charge Client. The **PC**Charge Client is included on the **PC**Charge Pro and Payment Server CD, and can be installed on the same (or on a different) computer as **PC**Charge. The **PC**Charge Client has payment processing and reporting capabilities.

- **Default User Name** - The default user name, `User1`, should not be renamed under any circumstances. **PC**Charge relies on this user name to perform several internal functions. If different user names are desired, they should be added in the **PC**Charge "Users" setup screen. See the section **Multi-User Support** (see page 62) for more information on users.

- **Clearing Variables** - If the OCX, DLL, or OLE/COM methods of integration will be used, always destroy the object (or use the `Clear` or `ClearVariables` methods) after the transaction has been processed, the results have been retrieved, and the `DeleteUserFiles` method has been called. Running the next transaction without clearing the properties and methods from the previous transaction has been known to cause duplicate transactions and double-charges.

- **Duplicate Transactions** – It is very important that merchants always enable the "`Require Duplicate Transactions to be Forced`" option in the **PC**Charge configuration screen and that the integrated application handles duplicate transactions properly. A duplicate transaction is defined as a transaction that contains the same account number, the same expiration date, and the same amount on the same day. If the forced duplicates option is enabled and a duplicate transaction is submitted to **PC**Charge, **PC**Charge will respond immediately with the error response "`Duplicate Trans; F+Card# to Force`". Enabling the forced duplicates option ensures that customers are not double charged because of errors (either human error, an error in the integrated application, or an error in **PC**Charge). If the duplicate transaction is legitimate, the integrated application should provide the end-user the ability to "force" the transaction—which is defined as adding the character "F" to the beginning of the card number. This character will override the forced duplicates option on a per-transaction basis. **Note:** The forced duplicates option is enabled by default in **PC**Charge, but many merchants disable it when the "`Duplicate Trans`" error message occurs—thus potentially allowing any transaction to be duplicated.

- **Enhanced Transaction Force** – **PC**Charge supports an enhanced transaction force feature. This feature allows forcing a duplicate transaction unless the ticket number is the same as a previous transaction. By placing an "E" in front of the card number, **PC**Charge is instructed to force the transaction unless all of the standard fields for a duplicate transaction (account number, expiration date, and amount) PLUS the ticket number are identical to a previous transaction. For example, if the "E" is included in front of the account number and the ticket number is different, the transaction will be accepted. However, if the ticket number is the same, the transaction will be considered a duplicate and will return the error response "`Duplicate Trans; F+Card# to Force`".

- **FDMS South / NaBanco Special Note** - FDMS South / NaBanco (NB) will occasionally return certain pieces of transaction information that contain spaces. **PC**Charge changes each of these spaces to an asterisk (*) to make the resulting information more legible. Previous versions of **PC**Charge did not convert these spaces to asterisks. Therefore, if upgrading from a previous version of the DevKit and the integrated application supports FDMS South / NaBanco, it is recommended that the application be updated to correctly parse the information with asterisks. Unexpected results may occur if the application is not updated.

- **Global East/NDC Special Note** – When processing manual (non-swiped) Visa or Discover card transactions where the CVV or CID values are not supplied, the following data must be passed in the CVV2 parameter:

    - **0** – Deliberated bypassed
    - **2** – CVV value illegible
    - **9** – Card has no CVV value

- **CES Special Note** – When processing manual (non-swiped) Visa or Discover card transactions where the CVV or CID values are not supplied, the following data must be passed in the CVV2 parameter:

    - **0** – Deliberated bypassed
    - **2** – CVV value illegible
    - **9** – Card has no CVV value

- **Elavon (NOVA) Special Note** – When processing manual (non-swiped) Visa or Discover card transactions where the CVV or CID values are not supplied, the following data must be passed in the CVV2 parameter:

    - **0** – Deliberated bypassed
    - **2** – CVV value illegible
    - **9** – Card has no CVV value

- **ADSI Special Note** – When processing manual (non-swiped) Visa or Discover card transactions where the CVV or CID values are not supplied, the following data must be passed in the CVV2 parameter:

    - **0** – Deliberated bypassed
    - **2** – CVV value illegible
    - **9** – Card has no CVV value

Timeouts

It is very important that integrated applications handle Timeout errors properly so that customers are not double-charged. Several of the **PC**Charge integration methods support options for handling Timeout errors.

Understanding Timeouts

The first step in handling Timeout errors properly is to understand why they occur. A Timeout error will occur if a transaction has not completed processing in **PC**Charge in the allotted time that the integrated application has provided. For example, if the integrated application has allotted 20 seconds for a transaction to be processed, the integrated application will only poll for a transaction response for 20 seconds. If a response has not been received from **PC**Charge in that amount of time, a Timeout error will occur in the integrated application. Even if a Timeout error occurs in the integrated application, **PC**Charge will continue to process the transaction. Imagine, in this example, the transaction was processed by **PC**Charge in 25 seconds, and was approved. Because the transaction took more than 20 seconds to complete, the integrated application assumes the transaction was not successful. However, **PC**Charge received an approved response and added the transaction to the batch. The problem this could cause is that a cashier will receive a message stating the transaction timed out will assume it was not approved--when it really was approved. The cashier will then attempt process the transaction again, thus potentially double-charging the customer's card and creating other reconciliation issues.

Transaction Delays

Transaction delays are the main reason why Timeout errors occur. Most delays are caused by payment processor issues, but some delays can be caused by modem issues, Internet connectivity issues, or configuration settings. The most typical causes for transaction delays are:

- **PC**Charge is attempting to process a transaction over the Internet, and the Internet connection or the payment processor's TCP/IP interface is down or is running slow.
- **PC**Charge is attempting to process a transaction using a dial-up modem and the modem is unable to connect, the payment processor's network is down, or the transaction is simply taking a long time to complete.
- **PC**Charge is attempting to process a transaction using a modem and the primary phone number is busy. **PC**Charge hangs up, pauses, and then dials the secondary number to receive an authorization.
- The Internet connection or the payment processor's TCP/IP interface is down, and **PC**Charge attempts to process the transaction using a modem (Dial-Up Modem Backup).
- **PC**Charge has several transactions waiting in the queue during a busy period, thus delaying transaction processing. See the section **Multi-User Support** (see page 62) for more information on transaction queuing.

To determine how much time should be allotted to **PC**Charge to process transactions, the merchant should take into consideration all of the reasons why transaction delays would occur.

Dial-Up Modem Backup Settings

PCCharge provides a Dial-Up modem backup feature for many of the payment processors that support Internet connectivity. When activating this feature, it is important that the default value in the setting "Internet Authorization Timeout Value (seconds)" be reviewed. This setting is located in the Advanced settings of the processor's extended data screen.

This setting determines how long **PC**Charge will wait for an Internet transaction to complete prior to "failing over" to the modem to process the transaction. Because most Internet transactions take only a few seconds to complete, the default value of 60 seconds should be lowered. If not, a cashier would have to wait for at least a minute (or more) for each transaction to process. If using the Dial-Up Modem Backup feature, this setting's value will need to be factored in when determining the integrated application's Timeout value.

Setting the Integrated Application's Timeout Value

To allow integrators to handle Timeouts properly, the various integration methods support a Timeout variable. If using the OCX, DLL, or OLE/COM Methods of integration, integrators may use the `TimeOut` property to set this variable. The default Timeout setting in each control and class is 90 seconds. If using the TCP Interface, integrators may use the `TXN_TIMEOUT` tag to set this variable.

Note: There is no Timeout setting for the File Method. Integrators must build in their own support for timeouts when using the File Method (the `TXN_TIMEOUT` tag has no effect on transactions submitted via the File Method).

When setting the Timeout value, several factors should be taken into consideration:

1. The maximum amount of time it would take to process a transaction:

    a. A typical transaction processed over the Internet takes about 3-5 seconds; a typical transaction that is processed using a modem averages about 25 seconds, but can take longer.

    b. If using a modem to process transactions, **PC**Charge will attempt to dial the secondary number if the primary number is busy. This can add 25 seconds or more to the transaction.

    c. If an Internet transaction fails, **PC**Charge will attempt to process the transaction using the modem if the Dial-Up Modem Backup feature is activated. The "Internet Authorization Timeout Value (seconds)" setting must be factored in as well as the dial time. (Again, if the primary phone number is busy, **PC**Charge will attempt to dial the secondary number.)

2. How long the merchant is willing to wait for a response after the "maximum amount of time" has elapsed.

Setting the Timeout to a high number (90 seconds, for example) would typically take the above factors into consideration, thus lowering the number of Timeout errors that would occur. However, setting this value too high could cause problems. For example, if a payment processor's network was down, cashiers would have to wait 90 seconds each time they submitted a transaction to receive an error response.

Setting the Timeout to a low number (10 seconds, for example) would alleviate the cashier wait time, but a greater number of Timeout errors would occur. Most likely, none of the transactions that used the Dial-Up Modem backup feature would register as approved in the integrated application. Cashiers would surely attempt to process the transaction again, thus double-charging a customer's card. However, if Duplicate Transaction checking is enabled, the cashier would immediately receive a "Duplicate Trans" message. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information.

The best way to determine how many seconds should elapse before a Timeout error occurs is to test the integrated application thoroughly. Testing will help to determine the normal amount of time that it takes to process transactions.

Handling Timeouts

OCX, DLL, or OLE/COM

If using the OCX, DLL, or OLE/COM methods of integration, the Timeout "countdown" begins when the `Send` method is called. Once the number of seconds set in `Timeout` has elapsed, an error event will fire or a response with a Result of "`Error`" will be returned. An error code of "6" will also be set (this can be retrieved using the `.GetErrorCode` method) to indicate a Timeout error.

Once a Timeout error occurs, it is suggested that the `Cancel` or `Abort` methods (if available) be called immediately to cancel the transaction. Because there is no guarantee that the `Cancel` or `Abort` methods will successfully cancel the transaction, it is suggested that a message be provided to the user of the integrated application that indicates a Timeout error has occurred. It is also suggested that the user review the **PC**Charge reports to determine whether the transaction was canceled properly or successfully processed.

Note: If using the OLE/COM Method of integration, the Timeout value will only be used if the application is written to process transactions in an asynchronous manner. If programming synchronously, **PC**Charge will ignore the Timeout value and always use pre-set timeouts when processing transactions. Changing the Timeout value when programming in a synchronous manner will not work due to the internal workings of **PC**Charge.

TCP Interface

If using the TCP Interface and the `TXN_TIMEOUT` flag, the "countdown" begins once **PC**Charge receives the transaction request. Once the number of seconds set in `TXN_TIMEOUT` have elapsed, **PC**Charge will return a `RESULT` of "`NOT CAPTURED`" and an `AUTH_CODE` of "`Timeout`" and will attempt to cancel the transaction in progress. Because there is no guarantee that the transaction will be successfully canceled, it is suggested that a message be provided to the user of the integrated application that indicates a Timeout error has occurred. It is also suggested that the user review the **PC**Charge reports to determine whether the transaction was canceled properly or successfully processed.

File Method

Although a tag named `TXN_TIMEOUT` appears in the File Layout section of the File Method, this tag may only be used when processing transactions using the TCP Interface. The integrated application must provide its own Timeout handling.

!	**WARNING:** VeriFone, Inc. takes security seriously therefore we have enhanced our integration methods to enhance security for sensitive data. If using **File Method** integration to PCCharge we strongly suggest you consider migrating towards a more secure integration method such as **TCP/IP SSL** socket integration.

Multi-User Support

Multi-user support can be defined as **PC**Charge's ability to:

- Support multiple workstations or registers
- Handle simultaneous transaction requests

PCCharge is able to handle multiple workstations and simultaneous request through the use of "Users". Users are unique **PC**Charge user names that must be purchased and configured in **PC**Charge by the merchant. Users must also be supported by the integrated application. The following describes how and why the Multi-User feature of **PC**Charge should be supported.

Note: Both **PC**Charge Pro and **PC**Charge Payment Server support multiple users. All **PC**Charge integration methods support multiple users.

Support for Multiple Workstations

In order for a integrated client/server application to work properly with **PC**Charge, **PC**Charge and each of the integrated client stations must support and be configured to use unique **PC**Charge user names.

In a typical client/server environment, **PC**Charge will be running on a Windows-based computer in a data center (or in the back office of a restaurant). Each employee or clerk (or waiter/waitress) that needs to process payments will typically have the integrated client running on their own workstation (or restaurant station). When a credit card, debit card, check, etc. is swiped (or keyed in), the integrated client will communicate to **PC**Charge over the LAN. In order for the integrated client and **PC**Charge to know which transaction response matches each workstation request, the integrated client must be uniquely identified by a user name.

When coding to support this scenario, make sure that the user name property can be set for each integrated client by the end-user. Typically, this user name setting would appear in a setup menu or properties page provided in the client.

When configuring each client station, the merchant should assign a unique user name to each client station (such as "`User1`", "`User2`", etc.). Once configured, this setting determines what is set in the "User" property of the Active X controls, OLE/COM classes, or TCP Interface. If using the File Method, the property will determine the name of the text file created by the client. In order to communicate to **PC**Charge using the user names that have been set up at each client station, each user name must be added in **PC**Charge.

User names are added by utilizing the User settings screen in the **PC**Charge GUI. This screen can be accessed from the Setup | Users menu in **PC**Charge. Alternatively, the Unlimited User License can be added rather than adding individual user names – see below for more information.

Note: Do not rename the default user name (`User1`). **PC**Charge uses this user name for several internal functions.

Support for Simultaneous Transaction Requests

PCCharge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. However, **PC**Charge can be configured to handle multiple simultaneous transaction requests. Once configured to support multiple simultaneous transaction requests, **PC**Charge still only processes these requests one at a time. **PC**Charge handles multiple requests through the use of queuing. The queuing feature of **PC**Charge processes the requests in a "first in first out" (FIFO) manner.

Simultaneous transaction requests usually occur in the following environments:

- Client/Server (i.e., multiple terminals, cash registers, or workstations)
- eCommerce enabled web-sites
- Multi-threaded applications

Support for simultaneous transaction requests is very similar to multiple workstation support. In fact, once multiple workstation support has been configured (see above), the ability to submit simultaneous transaction requests is already supported. However, the use of multiple workstations is not necessarily required to support simultaneous transaction requests.

To support simultaneous transaction requests, Multi-user support must be activated in **PC**Charge. At least two users or the Unlimited User License must be set up. When coding an application to send multiple requests, each request must have a unique user name. For example, "`User1`", "`User2`", etc. Again, **PC**Charge will "queue up" these transactions and will process them in the order that they are received.

Keep in mind that no two transaction requests can be submitted at the same time with the same user name. To reuse the same user name, the integration must wait until the transaction has been processed completely by **PC**Charge, the client has received the response, and deleted the response file (except for the socket method, there is no response file). Only then can the user name be reused to send another transaction request.

Known Conflict with Simultaneous Transaction Requests

PCCharge has a known issue with processing simultaneous transactions that have certain characteristics. "Invalid TroutD" errors may be returned when multiple transactions are submitted to **PC**Charge and have all the following characteristics:

- **The transactions are submitted simultaneously**. Specifically, multiple transaction requests are all submitted to **PC**Charge within 0.5 seconds (0.5 seconds is the default "Queue Timer Interval" setting in **PC**Charge. This is how often **PC**Charge polls for incoming transaction request files.)
- **The transactions that are submitted use the same processing company but have different merchant accounts with different communication settings**. For example, if TSYS will be used as the processing company, one transaction will use a TSYS merchant account that is set up to process via modem and another transaction will use a TSYS merchant account that is set up to process via the Internet.

  or

  Some transactions are processed by the processing company and others are processed via Split-dial directly to AMEX. For example, if Split-Dial is activated, a Visa credit card transaction will be processed by TSYS, and an American Express credit card transaction will be processed via Split-dial directly to American Express.

Again, keep in mind that all of the above characteristics must be met in order for this issue to occur—and this error only occurs when submitting simultaneous transactions. Currently, the only workaround is to avoid submitting simultaneous transactions that have all of the above characteristics. If the error occurs, the transaction should be re-submitted.

The issue occurs because of the design of the queuing module in **PC**Charge. VeriFone, Inc. is currently working on updating the queuing module to handle simultaneous transaction requests properly. This update will be available in a future release of **PC**Charge.

Setting up Multi-user support

Each copy of **PC**Charge Payment Server and **PC**Charge Pro ships, by default, with a single user license activated. If the integration will be multi-threaded, will be used in a client/server environment, or will somehow send more than one transaction at a time to **PC**Charge, support for additional users or unlimited users is required.

The integrator has the option to decide whether the application supports additional users or unlimited users or both.

Note: Every single transaction that **PC**Charge processes has a user name assigned to it. If the transaction originates in the **PC**Charge GUI or is submitted through the file interface (the OCX, DLL, OLE/COM, or File Methods), the file name that is used is the **PC**Charge user name. If the transaction is sent through the TCP Interface, the <USER_ID> field assigns the user name to the transaction once the transaction has begun processing in **PC**Charge.

Note: If using the File Method, the name of the file must match the <USER_ID> tag contained in the transaction request message.

Additional Users

Typically, merchants would add additional users if they have a static number (and usually a small amount) of workstations or client machines that will need to access **PC**Charge. For example, if a merchant had three "cash registers" or workstations, they would need to purchase two additional users (**PC**Charge already includes the first user license), and then set up the three registers or workstations with the three unique user names. Now, when the registers or workstations submit transactions, **PC**Charge can identify each register or workstation by its unique user name. Also, if transactions are sent from more than one register simultaneously, **PC**Charge can now queue these transactions. In essence, this merchant has enabled "queuing" and the queue can hold up to three transactions.

Note: It is recommended that the **PC**Charge naming convention for users (User + X) be used when adding additional users. **PC**Charge ships by default with a single user named "User1". When adding users, VeriFone, Inc., suggests that merchants use "User2", "User3", etc. It is not a requirement that this naming convention be followed, however, if there are problems in the future, using the default naming convention may decrease the amount of time needed on a technical support call. Regardless, all user names must be eight characters or less and can be alphanumeric, no spaces.

Note: Do not rename the default user name (User1). **PC**Charge uses this user name for several internal functions.

If additional users are not added to **PC**Charge and a client is set up to send transactions with a user name other than "User1", the transactions will fail. Specifically, if any client attempts to process

transactions with user names that do not appear in the **PC**Charge user screen, the transaction requests will be immediately rejected by **PC**Charge and the message "User Not Found" will be returned.

If the integrated application will support additional users, merchants will need to purchase additional user licenses and activate them in **PC**Charge to take advantage of the Multi-user features of the application. Once purchased, additional user licenses may be activated by contacting VeriFone, Inc.'s technical support department or by submitting an additional user request via the Support section of VeriFone, Inc.'s website: http://www.pccharge.com.

For information on how to add additional users to **PC**Charge Pro or **PC**Charge Payment Server, see the product manuals for each product. For information on pricing, merchants my contact an authorized VeriFone, Inc. reseller or call 800-725-9264.

Unlimited User License

Typically, merchants would opt for an unlimited user license if they have a variable number and/or a large number of workstations or client machines that will need to access **PC**Charge.

For example, if a merchant had twenty "cash registers" or workstations, it is recommended that they purchase and activate an Unlimited Users license. This would allow the merchant to set up as many registers or workstations as needed. The only requirement for Multi-User processing with an Unlimited User License is that all registers or workstations would need to be set up with unique user names. This allows **PC**Charge to identify each register or workstation by its unique user name. Also, if transactions are sent from more than one register simultaneously, **PC**Charge can now queue these transactions. In essence, this merchant has enabled "queuing" and the queue can hold an unlimited number of transactions. If the merchant needs to add registers or workstations in the future, there are no additional costs or configuration in **PC**Charge that would need to occur. The merchant would simply set up any new client machines with unique user names and begin processing.

An example of an environment that would require Unlimited Users would be a real-time eCommerce-enabled website. The Unlimited User License allows multiple customers to submit payments at the same time. Typically, real-time web applications would use a user naming scheme involving `SessionIDs` (a variable used by web servers).

Note: Some developers prefer integrating to support only the Unlimited User License. This type of integration assumes that the Unlimited User License will be activated in **PC**Charge. The developer can programmatically create unique user names on the fly, thus simplifying the coding and configuration aspects of a client/server installation or multi-threaded application.

Note: By adding the Unlimited User license to **PC**Charge, the restriction of the client application's user name having to match a user name set up in **PC**Charge is removed. Any valid alphanumeric eight character or less user name can be used by the client. However, each user name, when submitted must be unique—it cannot be the same as another transaction's user name. Once the transaction has been processed, however, the user name can be reused.

If the integrated application will support unlimited users, merchants will need to purchase the unlimited user license and activate it in **PC**Charge to take advantage of the Multi-user features of the application. Once purchased, the unlimited user license may be activated by contacting VeriFone, Inc.'s technical support department or by submitting an additional user request via the Support section of VeriFone, Inc.'s website: http://www.pccharge.com/. For information on how to add additional users to **PC**Charge Pro or **PC**Charge Payment Server, see the product manuals for each product. For information on pricing, merchants my contact an authorized VeriFone, Inc. reseller or call 800-725-9264.

Limitations of PCCharge's Multi-User Feature

Although there are no technical or programmatic limitations of how many users are supported or how many transactions can be processed by **PC**Charge, there are realistic limitations.

Because **PC**Charge is a single-threaded application, it can only process one transaction at a time. This could cause throughput issues if a large number of transactions are submitted simultaneously. In addition, if the merchant will be processing transactions via modem, or if **PC**Charge is used to connect via modem to multiple processors, delays can become severe.

For example, if 50 transactions are submitted simultaneously to **PC**Charge (and **PC**Charge is configured to support that many simultaneous transactions), **PC**Charge will queue up all 50 transactions and process them one at a time. If the Internet will be used to connect to the processing company, processing will average about four seconds per transaction. In this scenario, it will take almost 3.5 minutes for **PC**Charge to process all of the transactions in real-time (meaning the client that submitted the 50^{th} transaction will have to wait almost 3.5 minutes to get a response.) If a customer or clerk is waiting on the response, this amount of time may not be acceptable. Further, if **PC**Charge is required to dial different processors during this type of scenario, all of the dial and disconnect time would be factored in as well. This could greatly increase the amount of time it takes for the operation to complete.

Aside from throughput issues, the way **PC**Charge stores transaction data should also be taken into consideration. **PC**Charge stores all transaction data in a Microsoft Access database. Access databases are designed for use in a single-user or small workgroup environment. Access can easily handle a moderate amount of workstations and several hundred transactions per day. But, because of the characteristics of Microsoft Access, **PC**Charge is not designed to be implemented in an enterprise environment. If **PC**Charge is expected to store thousands of transactions a day; reporting, archival, settlement, and other operations may become difficult to perform or may take a long time to complete.

Alternatives

If a single copy of **PC**Charge is being used to support many workstations (or even multiple store locations), consider implementing multiple copies of **PC**Charge. Adding additional copies of **PC**Charge to a location or simply installing a copy of **PC**Charge at each store may alleviate throughput and data storage issues. Only one copy of PCCharge should be installed on a single machine. Keep in mind that merchants will need to purchase additional copies of **PC**Charge and may incur additional expenses for supporting multiple merchant accounts in this type of scenario. Merchants should check with their processing company or merchant services provider for more information on additional merchant account fees.

If using multiple copies of **PC**Charge is not an option, consider integrating and supporting RiTA Server or **IP**Charge. **IP**Charge is VeriFone's new gateway product. The **IP**Charge DevKit is included with the **PC**Charge DevKit. RiTA Server is a scaleable, multi-threaded application that is designed to support high transaction volumes and unlimited merchant numbers and users. If the File Method or TCP Interface method is used for the **PC**Charge integration, integration to RiTA Server would be very similar.

Multi-trans Wait

Enabling the Multi-trans Wait feature allows **PC**Charge to attempt to keep the modem connected to the processor for a certain amount of time (usually a few seconds) after each transaction is processed. If **PC**Charge will communicate to the processing company via dial-up modem, the Multi-trans Wait feature of **PC**Charge can greatly increase throughput during busy periods or when performing batches of transactions. As long as transactions are submitted to **PC**Charge within a short period of time (or if transactions are currently queued in **PC**Charge), each transaction will be processed on the same call and will not require a re-dial to the processing company. If Multi-trans Wait is not supported by the processor, a call will need to be made to the processing company for each transaction that is submitted.

Note: If the merchant will connect via the Internet to do payment processing, the Multi-trans Wait feature has no effect on processing.

Although **PC**Charge supports Multi-trans Wait, whether or not this feature will actually hold the modem line open depends completely on the processing company. If the processor doesn't support Multi-trans Wait, the processor will send a hang up command to **PC**Charge after each transaction. Also, the length of time that the call will stay connected also depends on the processing company. The length of time can range from half a second to twenty seconds or more.

Some processors support Multi-trans by default. If Multi-trans does not appear to be working after the Multi-trans Wait setting has been enabled in **PC**Charge, the merchant should contact their processing company or merchant services provider to request that the feature be activated for their merchant account(s). In some cases, merchants can also request how long the call will stay connected after each transaction.

By default, Multi-trans Wait is disabled in **PC**Charge. This feature can be enabled in two ways:

1) **Through the GUI**: The merchant should check the "Multi-trans Wait" option in the GUI of **PC**Charge Payment Server or **PC**Charge Pro (Setup | Configure System | Advanced).

2) **Through the API**: To enable this feature programmatically, set the "`Multi`" property of the OCX, DLL, or OLE/COM methods or the "`MULTI_FLAG`" property in the File Method or TCP Interface to "1" when submitting a transaction. If this setting is sent programmatically, it will override the GUI's Multi-trans Wait setting.

Multi-Merchant Support

PCCharge is designed to handle multiple merchant accounts. Each **PC**Charge product (**PC**Charge Pro and **PC**Charge Payment Server) includes a license for a single merchant account. In order for merchants to use more than one merchant account in **PC**Charge, they must first purchase and then activate additional merchant accounts.

Multi-Merchant Integration

In order for an integrated application to support multiple merchant accounts, the integrated application must provide the end-user the ability to choose which merchant account will be used to process transactions and allow this information to be passed to **PC**Charge. This can be accomplished by providing a setup option that allows the end-user to specify their merchant account information and then populating the proper properties or XML tags with that information. The merchant's account information is defined as the merchant account number and a valid processor code.

For example, assume an end-user has three merchant accounts set up in **PC**Charge. Two of the merchant accounts are with FDMS South (NB): 67888882701 & 67888882702, and the other merchant account is with Chase Paymentech (GSAR): 999999999999519. In order for the end-user to designate which merchant account should be used to process the transaction, the end-user must have a way to indicate to the integrated application to send the proper merchant account number and the processor code to **PC**Charge. In this example, if the merchant will use the second FDMS South merchant account to process the transaction, the integration must send the "67888882702" account number and the "NB" processor code to **PC**Charge. To pass the account number and processor code using the OCX, DLL, or OLE/COM Methods of integration, use the `MerchantNumber` and `Processor` properties. To pass the account number and processor code using the File Methods or TCP Interface, use the <PROCESSOR_ID> and <MERCH_NUM> tags.

Note: Choosing the processor code will be easier (and less error-prone) for the end-user if the integrated application provides a list that allows the end-user to choose their processor code rather than requiring them to type it in. A list of valid processor codes is located in the section **DevKit Constants** (see page 143). Also, the processor drop-down lists found in the various setup screens in **PC**Charge serve as accurate lists of the available processor codes.

GetMerchantInfo Method

The OCX, DLL, and OLE/COM Methods of integration provide a method named "`GetMerchantInfo`". This method is located in `Charge.OCX`, in the `Batch` class of `PSCharge.dll`, and in the `PccConfig` OLE/COM class. This method provides the integrated application with a string that includes a list of merchant numbers and processor codes that are currently set up in **PC**Charge. In addition, it will also return a character indicating if the processor is Host or Terminal based. Specifically, this method accesses the `tid.pcc` file that resides in the **PC**Charge directory, and will return all merchant accounts, processor codes, and terminal/host indicators that are set up in the file. The following is an example of a string that will be returned if three merchant numbers are set up in **PC**Charge:

```
<STX>CES <FS>000000927996296767<FS>T<GS>GSAR<FS>999999999999519<FS>T<GS>VISA
<FS>999999999911<FS>T<ETX>
```

Once retrieving this information at run-time, the integrated application could then display a user-friendly list of all of the merchant accounts that are set up in **PC**Charge. The end-user could then choose the proper merchant account from the list. This list would eliminate the need for an end-user to key in merchant account numbers and would reduce errors caused by typos.

OCX / DLL Method Note: PC Charge does not have to be running in order for the `GetMerchantInfo` method to return the string. However, a valid `Path` variable must be set prior to calling the `GetMerchantInfo` method.

Use Default Processor

Many merchants that use **PC**Charge require only a single merchant account. If only a single merchant account will be set up in **PC**Charge, integrators should consider supporting the "Use Default Processor" option. Enabling this option in **PC**Charge allows the integrator to omit the merchant number and processor code from transaction requests. If the option is enabled and the merchant number and processor code are omitted, **PC**Charge will automatically use the active merchant account set up in **PC**Charge to process the transaction.

The "Use Default Processor" option is available in the **PC**Charge Configure System setup screen. This option is disabled by default.

SSL Support

Introduction

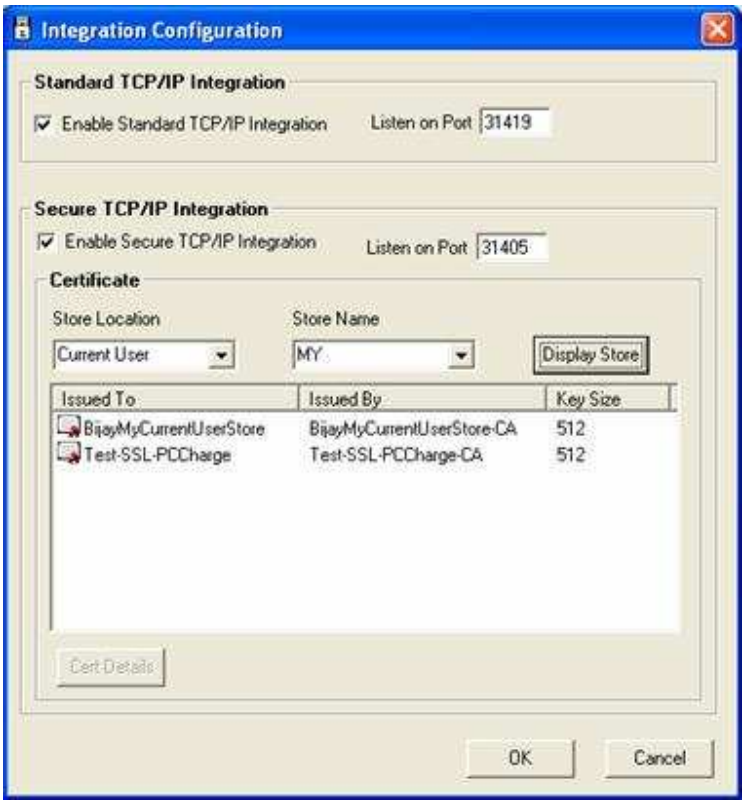
PCCharge now offers a new method of integration. Secure Socket Layer (SSL) via TCP/IP integration allows a developer to send transactions over a secure protocol. Due to its secure nature, this is the method of integration we highly recommend. The SSL protocol is commonly used for managing the security of messages sent over a local network and the Internet. SSL is supported thru the OCX: TCP/IP, the DLL: TCP/IP as well as TCP/IP methods of integration.

SSL Configuration

This section will provide steps to install the local SSL Certificate and enable SSL within PCCharge.

Follow these steps to select the certificate within PCCharge and enable SSL:

1) Open PCCharge.
2) Go to **Setup -> Configure System**.
3) Click **Advanced** on the bottom of the window.



4) Check **"Enable Secure TCP/IP Integration"**.
5) Use the **"Store Location"** dropdown box and select **"Current User"**.
6) Use the **"Store Name"** dropdown box and select **"MY"**.
7) Click **"Display Store"**.
8) Highlight the correct certificate.
    i) The current certificate being installed by PCCharge is called **"PCChargeDefaultCertificate-CA"**.
    ii) Prior to the final build of PCCharge 5.8.0 the certificate was called **"Test-SSL-PCCharge"**.
9) This should complete the setup for SSL integration with PCCharge.

SSL Integration Notes

SetSSLCertificate

The method "SetSSLCertificate": User can set the local certificate to authenticate the SSL TCP/IP request. This routine has four parameters as follows.

SSLCertStoreName : this is the name of the store. **Example: "MY"**

SSLCertificateLocation : this is the location of the store the certificate was installed into. Valid values are "**0**" for Local Machine or "**1**" for Current User.

SSLCertIssuedBy : Issuer of the certificate. Found by looking at certificate details in PCCharge. **Example: "PCChargeDefaultCertificate-CA"**

SSLCertSerialNumber : This is the serial number of the certificate; this is also found in the certificate details in PCCharge. **Example: "07427b6a70e37f3e"**

Syntax Example:

```
SetSSLCertificate (SSLCertStoreName As String, SSLCertificateLocation As String, SSLCertIssuedBy As String, SSLCertSerialNumber As String)
```

With the following values:

```
SSLCertStoreName          : MY
SSLCertificateLocation    : 0
SSLCertIssuedBy           : PCChargeDefaultCertificate-CA
SSLCertSerialNumber       : 07427b6a70e37f3e
```

Example:

```
SetSSLCertificate ("MY", "0", "PCChargeDefaultCertificate-CA", "07427b6a70e37f3e")
```

This will create certificate and authenticate before sending request to SSL Server.

EnableSSL

This value must be set to True to enable SSL Integration using the OCX or DLL methods of Integration.

Custom SSL Certificates

The default PCCharge certificate **"PCChargeDefaultCertificate-CA"** can be used on the PCCharge server and deployed with a POS on a workstation as well. PCCharge is also capable of using a certificate created and issued by the integrator. The developer must provide a **.PFX (PKCS#12)** formatted certificate that meets specific requirements. Here is a list of the requirements that must be met by the certificate for use by PCCharge:

Certificate Properties	Description
Version Number	The version of the X.509 standard to which the certificate conforms.
Serial Number	A number that uniquely identifies the certificate and is issued by the certification authority.
Certificate Algorithm Identifier	The names of the specific public key algorithms that the certification authority has used to sign the digital certificate.
Issuer Name	The identity of the certification authority who actually issued the certificate.
Validity Period	The period of time for which a digital certificate is valid and contains both a start date and expiration date.
Subject Name	The name of the owner of the digital certificate.
Subject Public Key Information	The public key that is associated with the owner of the digital certificate and the specific public key algorithms associated with the public key.
Issuer Unique Identifier	Information that can be used to uniquely identify the issuer of the digital certificate.
Subject Unique Identifier	Information that can be used to uniquely identify the owner of the digital certificate.
Extensions	Additional information that is related to the use and handling of the certificate.
Certification Authority's Digital Signature	The actual digital signature made with the certification authority's private key using the algorithm specified in the certificate algorithm identifier field.

Manual SSL Certificate Install

PCCharge will install the certificate on the server automatically, but not on the POS workstation. If an integrator needs to manually install the default PCCharge SSL certification or a certificate they've created, these are the steps required:

1) Click **Start** then **Run**.
2) Type **"MMC"** in the **Run** dialog box and hit **Enter** or click **OK**.
3) Select **Add/Remove Snap-in** from the **File** menu.
4) Click **Add** and select **Certificates** from the **Add Standalone Snap-in** dialog.
5) Click **Add** and select **Computer Account** from the **Certificates Snap-in** dialog.
6) Select **Local Computer** on the **Select Computer** window.
7) Click **Finish** and then **Close**.
8) Click **OK** to close the **Add/Remove Snap-in** window.
9) Expand the **Certificates (Local Computer)** folder and right-click **Personal** under the **Certificates** tree.
10) Select **Import.**
11) The **Import Wizard** should start, click **Next** one time.
12) The 2nd screen should allow you to **browse** to your certificate. You'll need to change the **file type** to **All files (*.*)**.
13) Once you have found your certificate click **Next**, without making further changes, until you get to **Finish**.

Cashier Permissions

To take advantage of the new PCI Cashier Updates, an Integrator will now have to submit the following two tags with every transaction. **NOTE:** If you do not wish to send anything new, send neither tag.

- CASHIER_NAME
- PASSWORD

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

***NOTE**: If the developer is integrating to PCCharge Pro and will be using cashier ids and passwords, their cashiers will have to log into the Pro user interface and change their password before they will be able to process in the POS.*

The CASHIER_NAME of "**System**" only functions as PCCharge administrator and does not have full production permissions. A "**System**" user will not be able to process transactions or settle batches, but will have full access to make changes to other users, merchant setup information, and device setup.

A Cashier will not need to log in to the PCCharge user interface as the above tags allow the cashier to basically "log in" during each transaction. If a user attempts to send the CASHIER_NAME of "**System**" for a financial transaction, they will be blocked just as if they were operating in the PCCharge User Interface.

New Cashier Transactions

Action Code	Description
ZP	Change Password
ZU	Add Cashiers
ZD	Delete Cashier
ZR	Re-enable Cashier
ZL	Log out Cashier
ZZ	Get Expiration

Just like within the UI, the Add, Delete and Re-enable commands can only be performed by the 'System' cashier. The '**System**' credentials must be passed with these commands in the request.

Additional XML Tags

For all commands **except ZL**: `<CASHIER_NAME>, <PASSWORD>`

For **ZU**: `<ADD_CASHIER_1>, <ADD_CASHIER_2>, <ADD_CASHIER_3>, <ADD_CASHIER_4>, <ADD_CASHIER_5>`

For **ZD** and **ZR**: `<TARGET_CASHIER_NAME>`

For **ZP**: `<NEW_CASHIER_PASSWORD>, <NEW_CASHIER_CONFIRM_PASSWORD>`

For **ZZ**: `<PWD_EXPIRY>`

Note: *The <PWD_EXPIRY> tag will not be seen in any INX that is sent through integration. However, you will see these both in the INX and OUX portion of the IODebug and it will be sent to the integrator in the OUX. This tag contains the cashier's password expiration date.*

Format Examples

Change Password – ZP Command

This allows a user to setup a password after installing PCCharge. This is required prior to submitting transactions.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

Sample

INX

```
<XML_REQUEST>
      <USER_ID>User1</USER_ID>
      <CASHIER_NAME>Eric</CASHIER_NAME>
      <PASSWORD>Test123%</PASSWORD>
      <NEW_CASHIER_PASSWORD>New123%</NEW_CASHIER_PASSWORD>
      <NEW_CASHIER_CONFIRM_PASSWORD>New123%</NEW_CASHIER_CONFIRM_PASSWORD>
      <COMMAND>ZP</COMMAND>
</XML_REQUEST>
```

Response Code	Description
51	SUCCESS
52	"Cashier Not Found" – Cashier is not in the database.
53	"Invalid Password" – Password is incorrect.
54	"New Password not Strong" – New password does not meet minimum standards.
55	"New Password not Confirmed" – New password does not match confirm password.
56	"Password Previously Used" – New password is one of the previous passwords.
57	"Password is in use" – New Password is the same as the current password.
58	"Invalid Information" – Either the Cashier ID or the Password are empty.

Add Cashier – ZU Command

This allows integrators to setup cashiers without entering the PCCharge UI. It is possible to pass up 5 new cashiers at a time. The information is used in a name value pair setup within each tag/field.

Note: *Cashier names are not case-sensitive. PCCharge will UCase all cashier names in the background, but mixed case can be used by integrators. The Name Value Pair names ARE case-sensitive and must be submitted in all upper case*

FORMAT	**EXAMPLE**
CASHIER_NAME <Name>	CASHIER_NAME ANGIE1
PASSWORD <Password>	PASSWORD P@ssw0rd
PERMISSIONS <Permissions>	PERMISSIONS ||||||

Note: The Name Value Pairs with each ADD_CASHIER tag must be separated by spaces. No carriage returns are allowed. The transaction will not be processed if carriage returns are present within the ADD_CASHIER tags.

Permissions Bitmap

The Permissions Bitmap will have a 0 (not allowed) or a 1 (allowed) for each permission. There should be a total of 8 sections, each section is separated by a pipe character "|" for a total of 7 pipes (do not add an additional "|" at the end).

If a permissions value is left out, it is defaulted to 0 (not allowed). This means a PERMISSIONS of "|||||||" that is sent defaults a cashier to no permissions. A table of permissions is located in **Chapter 5-- DevKit Constants under the section Cashier Permissions Constants**.

EXAMPLE:

```
<XML_REQUEST>
	<USER_ID>User1</USER_ID>
	<CASHIER_NAME>System</CASHIER_NAME>
	<PASSWORD>Test123%</PASSWORD>
	<ADD_CASHIER_1>
		CASHIER_NAME Eric
		PASSWORD P@ssw0rd
		PERMISSIONS 11110|000|111|100|101111000011|1100|001111|01
	</ADD_CASHIER_1>
	<ADD_CASHIER_2>
		CASHIER_NAME System
		PASSWORD P@ssw0rd
		PERMISSIONS 11110|000|111|100|001111000011|1100|001111|01
	</ADD_CASHIER_2>
	<ADD_CASHIER_3>
		CASHIER_NAME Phil
		PASSWORD P@ssw0rd
		PERMISSIONS 11110|000|111|100|001111000011|1100001111|01
	</ADD_CASHIER_3>
	<ADD_CASHIER_4>
		CASHIER_NAME Beatrice
		PASSWORD P@ssw0rd
		PERMISSIONS 11110|000|111|100|001111000011|1100|001111|01
	</ADD_CASHIER_4>
	<ADD_CASHIER_5>
		CASHIER_NAME YourMother
		PASSWORD Passw0rd
		PERMISSIONS 11110|000|111|100|001111000011|1100|001111|01
	</ADD_CASHIER_5>
	<COMMAND>ZU</COMMAND>
</XML_REQUEST>
```

This example should come back with a response of "**03451**".
0 = Successful – ADD_CASHIER_1 was successfully added.
3 = Invalid Username – ADD_CASHIER_2 'System' is an invalid username.
4 = Invalid Permissions - ADD_CASHIER_3 had invalid or incorrectly formatted permissions.
5 = Illegal Permissions - ADD_CASHIER_4 was unsuccessful due to permissions conflict.
1 = Password is not Strong - ADD_CASHIER_5 failed, the password did not meet strong requirements.

NOTE: The response will come back in the auth field. It will either be "**Success**" if all 5 cashiers are successfully added, or it will be a five-digit number. Each digit corresponds to the corresponding cashier. It is theoretically possible to have a successful **NOT CAPTURED**. This can occur if some of the cashiers are successfully setup and some aren't. Therefore, the integrator should pay close attention

to the response. If, however, the "**unsuccessful**" cashiers are just ones that aren't sent in, then the transaction will still be **CAPTURED**.

Example: Two cashiers are sent in (and they are successful) a response of "00999" will come back. This will be **CAPTURED**. The three "9"s at the end signify that nothing was sent in this case.

Example: Three cashiers are sent in, and the third one was an attempt to set up System, then the response would be "00399". This would be **NOT CAPTURED**. This would tell the integrator that at least one of the sent in cashiers was unsuccessful.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

Cashier Transaction Response	Description
0	SUCCESSFUL
1	Password is not Strong
2	Password is Previous
3	Invalid Username (SYSTEM was passed in)
4	Invalid Permissions (Less than 7 pipes in the map)
5	Illegal Permissions (Debit Voids cannot be setup without Debit Transactions for example)
9	No Cashier Sent (Cashier Name was empty.)

Add Cashier Example

Example of "Add Cashier" Request:

```
<XML_REQUEST>
      <USER_ID>User1</USER_ID>
      <RESP_TYPE>3</RESP_TYPE>
      <COMMAND>ZU</COMMAND>
      <CASHIER_NAME>System</CASHIER_NAME>
      <PASSWORD>System123%</PASSWORD>
      <ADD_CASHIER_1>CASHIER_NAME CreditR PASSWORD CreditR123% PERMISSIONS
      10000|000|000|000|000000000000|0000|000000|00</ADD_CASHIER_1>
      <ADD_CASHIER_2>CASHIER_NAME CreditV PASSWORD CreditV123%  PERMISSIONS
      01000|000|000|000|000000000000|0000|000000|00</ADD_CASHIER_2>
      <ADD_CASHIER_3>CASHIER_NAME CreditB PASSWORD CreditB123% PERMISSIONS
      00100|000|000|000|000000000000|0000|000000|00</ADD_CASHIER_3>
      <ADD_CASHIER_4>CASHIER_NAME CreditX PASSWORD CreditX123% PERMISSIONS
      00010|000|000|000|000000000000|0000|000000|00</ADD_CASHIER_4>
      <ADD_CASHIER_5>CASHIER_NAME CreditI PASSWORD CreditI123% PERMISSIONS
      00001|000|000|000|000000000000|0000|000000|00</ADD_CASHIER_5>
</XML_REQUEST>
```

Example of "Add Cashier" Response:

```
<XML_REQUEST>
      <USER_ID>User1</USER_ID>
      <RESULT>PROCESSED</RESULT>
      <AUTH_CODE>Success</AUTH_CODE>
      <INTRN_SEQ_NUM>1892</INTRN_SEQ_NUM>
</XML_REQUEST>
```

Change Cashier Permissions (ZU)

The ZU (Add) command is also used to change an existing cashier's permissions. You would submit the same XML tags as you would for an actual Add Cashier, but can omit the PASSWORD Name Value Pair in the request.

Just like with regular Add Cashier, you can change the permissions of up to 5 cashiers at a time. It follows all the same conventions.

```
INX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>ZU</COMMAND>
   <CASHIER_NAME>System</CASHIER_NAME>
   <PASSWORD>P@ssw0rd1</PASSWORD>
   <ADD_CASHIER_1>CASHIER_NAME EricH PERMISSIONS
11110|111||||||</ADD_CASHIER_1>
</XML_REQUEST>

OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <RESULT>CAPTURED</RESULT>
   <AUTH_CODE>09999</AUTH_CODE>
   <INTRN_SEQ_NUM>13079</INTRN_SEQ_NUM>
</XML_REQUEST>
```

Change Password – Forgotten Password (ZU)

The ZU command can also be used to reset a cashier's password in situations where the cashier has forgotten the original password. The entire set of Name Value Pairs must be sent with the request just as if this was adding a new cashier. The integrator will have to know the PERMISSIONS bitmap for that particular cashier – if they send seven pipes (||||||), it will reset the permissions of that cashier to base permissions (only credit card sales/preauths allowed).

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

Note: This command should only be used if the cashier has forgotten their original password. If they know their original password, a ZP command should be used to change their password.

```
INX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>ZU</COMMAND>
   <CASHIER_NAME>System</CASHIER_NAME>
   <PASSWORD>Test123$</PASSWORD>
   <ADD_CASHIER_1>CASHIER_NAME EricH PASSWORD Test123% PERMISSIONS
   |||||||</ADD_CASHIER_1>
</XML_REQUEST>

OUX:
<XML_REQUEST>
```

```
   <USER_ID>User1</USER_ID>
   <RESULT>CAPTURED</RESULT>
   <AUTH_CODE>09999</AUTH_CODE>
   <INTRN_SEQ_NUM>13136</INTRN_SEQ_NUM>
</XML_REQUEST>
```

Change cashier password – ZP Command

Example of "Change Cashier password" Request:

```
<XML_REQUEST>
      <USER_ID>User1</USER_ID>
      <COMMAND>ZP</COMMAND>
      <CASHIER_NAME>CreditR</CASHIER_NAME>
      <PASSWORD>CreditR123%</PASSWORD>
      <NEW_CASHIER_PASSWORD>CreditR789%</NEW_CASHIER_PASSWORD>
      <NEW_CASHIER_CONFIRM_PASSWORD>CreditR789%
      </NEW_CASHIER_CONFIRM_PASSWORD>
</XML_REQUEST>
```

Example of "Change Cashier password" Response:

```
<XML_REQUEST>
      <USER_ID>User1</USER_ID>
      <RESULT>PROCESSED</RESULT>
      <AUTH_CODE>Success</AUTH_CODE>
      <INTRN_SEQ_NUM>1882</INTRN_SEQ_NUM>
</XML_REQUEST>
```

Delete cashier – ZD Command

Example of "Delete Cashier" Request:

```
<XML_REQUEST>
      <USER_ID>User1</USER_ID>
      <COMMAND>ZD</COMMAND>
      <CASHIER_NAME>System</CASHIER_NAME>
      <PASSWORD>System123%</PASSWORD>
      <TARGET_CASHIER_NAME>CreditR</TARGET_CASHIER_NAME>
</XML_REQUEST>
```

Example of "Delete Cashier" Response:

```
<XML_REQUEST>
      <USER_ID>User1</USER_ID>
      <RESULT>PROCESSED</RESULT>
      <AUTH_CODE>Cashier deleted</AUTH_CODE>
      <INTRN_SEQ_NUM>1897</INTRN_SEQ_NUM>
</XML_REQUEST>
```

Re-enable cashier – ZR Command

Example of "Re-enable Cashier" Request:

```
<XML_REQUEST>
```

```
      <USER_ID>User1</USER_ID>
      <COMMAND>ZR</COMMAND>
      <CASHIER_NAME>System</CASHIER_NAME>
      <PASSWORD>System123%</PASSWORD>
      <TARGET_CASHIER_NAME>CREDITR</TARGET_CASHIER_NAME>
</XML_REQUEST>
```

Example of "Re-enable Cashier" Response:

```
<XML_REQUEST>
      <USER_ID>User1</USER_ID>
      <RESULT>PROCESSED</RESULT>
      <AUTH_CODE>Cashier Reenabled</AUTH_CODE>
      <INTRN_SEQ_NUM>1937</INTRN_SEQ_NUM>
</XML_REQUEST>
```

Get Password Expiration Date – ZZ Command

Example of Get Expiration Request:

This command will allow an integrator to pass in just the cashier credentials and get back when the cashier's password will expire. The OUX will return a "Processed" response. The password expiration date will be returned in the <PWD_EXPIRY> tag.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

```
INX:
      <XML_REQUEST>
            <USER_ID>User1</USER_ID>
            <COMMAND>ZZ</COMMAND>
            <CASHIER_NAME>Shelly</CASHIER_NAME>
            <PASSWORD>Test123%</PASSWORD>
      </XML_REQUEST>
OUX:
      <XML_REQUEST>
            <USER_ID>User1</USER_ID>
            <RESULT>PROCESSED</RESULT>
            <PWD_EXPIRY >MMDDYY</ PWD_EXPIRY >
      </XML_REQUEST>
```

Log out Cashier - ZL Command

The **ZL** command is used behind the scenes by Client to log out the logged in cashier on the server application. This command is triggered if a cashier attempts to process unsuccessfully in Client (possible hack attempt). It is possible for this same cashier id to be logged into the server application. Therefore, when a cashier lockout occurs in Client, the **ZL** command is sent to the server to log out whoever is logged in. This is a fraud protection. This forces 'System" to log in and re-enable that cashier id. An integrator can submit this command to log out the cashier if they suspect tampering as well.

```
INX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>ZL</COMMAND>
```

```
</XML_REQUEST>
```

If no one is currently logged into PCCharge, the integrator will receive the following response.

```
OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <RESULT>PROCESSED</RESULT>
   <AUTH_CODE>Nobody Logged In</AUTH_CODE>
   <INTRN_SEQ_NUM>13087</INTRN_SEQ_NUM>
</XML_REQUEST>
```

Successful logout returns the following response.

```
OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <RESULT>PROCESSED</RESULT>
   <AUTH_CODE>Logged Out</AUTH_CODE>
   <INTRN_SEQ_NUM>13089</INTRN_SEQ_NUM>
</XML_REQUEST>
```

Running transactions with cashier credentials

PCCharge will support sending cashier credentials with the transaction data as well as not sending these credentials (for backwards compatibility). If the integrator chooses to send the cashier credentials, these credentials must be for valid cashier/password combinations.

Note: It should be stressed to integrators that in order to be PCI compliant to the PABP Guidelines, they need to be setting up cashiers and sending cashier credentials with their transaction requests.

Sample of Incorrect Password being sent:

```
INX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>1</COMMAND>
   <CASHIER_NAME>EricH</CASHIER_NAME>
   <PASSWORD>P@ssw0rd1</PASSWORD>
   <PROCESSOR_ID>VISA</PROCESSOR_ID>
   <MERCH_NUM>888000001005</MERCH_NUM>
   <ACCT_NUM>5439750001500347</ACCT_NUM>
   <EXP_DATE>1210</EXP_DATE>
   <MANUAL_FLAG>0</MANUAL_FLAG>
   <TRANS_AMOUNT>1.00</TRANS_AMOUNT>
</XML_REQUEST>

OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <TROUTD>13074</TROUTD>
   <RESULT>Error</RESULT>
   <AUTH_CODE>Invalid Login Info</AUTH_CODE>
   <REFERENCE>Invalid Password</REFERENCE>
   <INTRN_SEQ_NUM>13074</INTRN_SEQ_NUM>
```

```
</XML_REQUEST>
```

Sample of Invalid Cashier Name being sent:

```
INX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>1</COMMAND>
   <CASHIER_NAME>BettyJ</CASHIER_NAME>
   <PASSWORD>P@ssw0rd1</PASSWORD>
   <PROCESSOR_ID>VISA</PROCESSOR_ID>
   <MERCH_NUM>888000001005</MERCH_NUM>
   <ACCT_NUM>5439750001500347</ACCT_NUM>
   <EXP_DATE>1210</EXP_DATE>
   <MANUAL_FLAG>0</MANUAL_FLAG>
   <TRANS_AMOUNT>1.00</TRANS_AMOUNT>
</XML_REQUEST>

OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <TROUTD>13075</TROUTD>
   <RESULT>Error</RESULT>
   <AUTH_CODE>Invalid Login Info</AUTH_CODE>
   <REFERENCE>CashierID Not Found</REFERENCE>
   <INTRN_SEQ_NUM>13075</INTRN_SEQ_NUM>
</XML_REQUEST>
```

Note: Unlike the UI, there is no cashier locking in the event of incorrect credential submission with a transaction sent through integration. The integrator will continue to receive the "Invalid Password" response until such time as they submit the correct credentials for that cashier.

Example of successful transaction sent with cashier credentials.

```
INX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>1</COMMAND>
   <CASHIER_NAME>Shelly</CASHIER_NAME>
   <PASSWORD>Test123$</PASSWORD>
   <PROCESSOR_ID>VISA</PROCESSOR_ID>
   <MERCH_NUM>888000001005</MERCH_NUM>
   <ACCT_NUM>5439750001500347</ACCT_NUM>
   <EXP_DATE>1210</EXP_DATE>
   <MANUAL_FLAG>0</MANUAL_FLAG>
   <TRANS_AMOUNT>1.00</TRANS_AMOUNT>
</XML_REQUEST>

OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <TROUTD>13090</TROUTD>
   <RESULT>CAPTURED</RESULT>
   <AUTH_CODE>TAS756</AUTH_CODE>
   <REFERENCE>813021502892</REFERENCE>
   <AVS_CODE>0</AVS_CODE>
```

```
   <TRANS_DATE>050908</TRANS_DATE>
   <INTRN_SEQ_NUM>13090</INTRN_SEQ_NUM>
   <TRANS_ID>0000123456789</TRANS_ID>
   <TICODE>CK</TICODE>
   <MSI>A</MSI>
   <PEM>5</PEM>
   <TIM>170652</TIM>
   <ACI>A</ACI>
   <PROC_RESP_CODE>00</PROC_RESP_CODE>
   <CMRCL_TYPE>0</CMRCL_TYPE>
   <PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
   <CMRCL_FLAG>N</CMRCL_FLAG>
   <CARD_ID_CODE>M</CARD_ID_CODE>
   <ACCT_DATA_SRC>T</ACCT_DATA_SRC>
</XML_REQUEST>
```

Follow On Transactions

Overview

When integrating to **PC**Charge, developers must make a decision on whether or not to support "follow on" transactions in their application. While a few integrators decide to support only basic transactions such as Sales or Pre-Authorizations in their application, most integrators choose to support the majority of available follow on transactions, such as Voids, Reversals, Post-Authorizations, and Gratuities. **PC**Charge requires that follow on transaction requests include the TroutD (*T*ransaction *ROUT*ing I*D*) from the original Sale or Pre-Authorization when they are submitted. The TroutD is a **PC**Charge-assigned unique number associated with a single transaction--or, in the case of follow on transactions, a TroutD can be associated with an interrelated series of transactions.

For simplicity, **PC**Charge usually requires that only the TroutD of the original Sale or Pre-Authorization transaction and the action code be set in order to perform follow on transactions. The TroutD enables **PC**Charge to pull all needed information (merchant number, card number, expiration date, auth code, etc.) from the transaction record in the **PC**Charge database in order to submit the follow on transaction.

In some cases, additional values may be sent with the follow on transaction. For example, if performing a Post-Authorization for a different amount than the original Pre-Authorization, the different amount may be sent in with the Post-Authorization. This is also true with Reversals. An integrator may send an amount with the troutd. This will alert PCCharge to Reverse the amount specified instead of the entire transaction amount.

Note: Support for TroutD "follow on" transactions was added starting with **PC**Charge version 5.6.0. Follow on transaction support is available in all integration methods.

Examples

The following examples show common uses for follow on transactions:

File Method / TCP Interface Examples:

This example demonstrates performing a Void on a Sale using the File Method or TCP Interface. Refer to the section **File Method** (see page 481) for more information on the File Method or TCP Interface API.

1. A Sale transaction is processed and **PC**Charge assigns a unique TroutD number (1024) to it.

Request:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<COMMAND>1</COMMAND>
		<PROCESSOR_ID>VISA</PROCESSOR_ID>
		<MERCH_NUM>999999999911</MERCH_NUM>
		<ACCT_NUM>5424180279791765</ACCT_NUM>
		<EXP_DATE>1211</EXP_DATE>
		<MANUAL_FLAG>0</MANUAL_FLAG>
		<TRANS_AMOUNT>10.00</TRANS_AMOUNT>
	</XML_REQUEST>
</XML_FILE>
```

Response:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<TROUTD>1024</TROUTD>
		<RESULT>CAPTURED</RESULT>
		<AUTH_CODE>124954</AUTH_CODE>
		<TRANS_DATE>1231</TRANS_DATE>
		<INTRN_SEQ_NUM>1024</INTRN_SEQ_NUM>
		<TRANS_ID>0412MCC364698</TRANS_ID>
		<RET>A014</RET>
		<ACI>P</ACI>
	</XML_REQUEST>
</XML_FILE>
```

2. Later, a second Sale transaction is processed and **PC**Charge assigns a unique TroutD number (1025) to it.

Request:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<COMMAND>1</COMMAND>
		<PROCESSOR_ID>VISA</PROCESSOR_ID>
		<MERCH_NUM>999999999911</MERCH_NUM>
		<ACCT_NUM>5424180279791765</ACCT_NUM>
		<EXP_DATE>1211</EXP_DATE>
		<MANUAL_FLAG>0</MANUAL_FLAG>
		<TRANS_AMOUNT>11.00</TRANS_AMOUNT>
	</XML_REQUEST>
</XML_FILE>
```

Response:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<TROUTD>1025</TROUTD>
		<RESULT>CAPTURED</RESULT>
		<AUTH_CODE>124956</AUTH_CODE>
		<TRANS_DATE>1231</TRANS_DATE>
		<INTRN_SEQ_NUM>1025</INTRN_SEQ_NUM>
		<TRANS_ID>0412MCC729964</TRANS_ID>
		<RET>A014</RET>
		<ACI>P</ACI>
	</XML_REQUEST>
</XML_FILE>
```

3. The merchant decides to process a Void transaction on the first Sale transaction. To Void the transaction, the integrated application sends the action code for a Void transaction (action code 3) and the TroutD number of the transaction to be voided (1024, in this example).

Request:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<COMMAND>3</COMMAND>
		<TROUTD>1024</TROUTD>
	</XML_REQUEST>
</XML_FILE>
```

Response:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<TROUTD>1024</TROUTD>
		<RESULT>VOIDED</RESULT>
		<INTRN_SEQ_NUM>1026</INTRN_SEQ_NUM>
		<PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
	</XML_REQUEST>
</XML_FILE>
```

4. **PC**Charge does not need any additional information to process the Void. Of course, if a follow on transaction will be performed that modifies some aspect of the original transaction (such as a Post-Authorization whose amount is less than the original Pre-Authorization), that information would need to be sent to **PC**Charge in addition to the action code and TroutD.

File Method / TCP Interface Restaurant Example:

Assume the merchant above is operating in a restaurant environment. This merchant still has one transaction left in the batch. Now, the merchant wishes to modify the second Sale transaction by adding a gratuity to it. (**Note:** The processor used must support restaurant transactions in order to add gratuities)

1. To add the gratuity to the second transaction, the application sends the action code for a Gratuity transaction (action code 13), the TroutD of the Sale transaction (1025, in this example), and the gratuity amount in the <GRATUITY_AMNT> tag ($2.51, in this example).

Request:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<COMMAND>13</COMMAND>
		<TROUTD>1025</TROUTD>
		<GRATUITY_AMNT>2.51</GRATUITY_AMNT>
	</XML_REQUEST>
</XML_FILE>
```

Response:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<TROUTD>1025</TROUTD>
		<RESULT>GRATUITY ADDED</RESULT>
		<AVS_CODE>0</AVS_CODE>
		<INTRN_SEQ_NUM>1027</INTRN_SEQ_NUM>
		<ACI>N</ACI>
		<CMRCL_TYPE>0</CMRCL_TYPE>
		<PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
		<CARD_ID_CODE>N</CARD_ID_CODE>
		<ACCT_DATA_SRC>T</ACCT_DATA_SRC>
	</XML_REQUEST>
</XML_FILE>
```

2. If, prior to settlement, the merchant wishes to adjust the gratuity amount of this transaction for some reason, the same information (Action code 13, TroutD 1025) for the transaction can be sent in with the new gratuity amount ($3.51, in this example).

Request:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<COMMAND>13</COMMAND>
		<TROUTD>1025</TROUTD>
		<GRATUITY_AMNT>3.51</GRATUITY_AMNT>
	</XML_REQUEST>
</XML_FILE>
```

The gratuity amount has been adjusted.

Implementing Follow On Transactions

In order to allow an integrated application to process follow on transactions effectively, it is recommended that the application:

1) Store the TroutD value
2) Store the status of the transaction:
    a. Whether it has been voided
    b. Whether it has been post-authorized
    c. Whether the gratuity has been added
    d. Whether it has been settled (and is not eligible for further follow on transactions)

The TroutD for each transaction can be retrieved either from the **<**`TroutD`**>** tag of the response (in the File or TCP Methods) or by using the `GetTroutD` method (available in the OCX, DLL, or OLE/COM methods).

Once this information is available to the application, lists or menu options should be provided by the application that allow merchants to perform follow on transactions easily.

Alternatively, the merchant can be instructed to view the **PC**Charge detail reports to acquire TroutD values and manually enter them in a screen provided by the application or even into the **PC**Charge GUI.

The following is a list of credit card transactions that can be processed using TroutD follow on support:

Credit Card Transactions:

- Void Sale
- Post-Authorization (amount is optional)
- Void Credit
- Void Post-Authorization
- Procurement Card Post-Authorization (amount is optional, tax and customer code should be passed for lowest rates). **Note:** Global East (NDC), terminal based, requires the customer code be all upper case.
- Gratuity (gratuity amount is required)
- Void Book

Non-TroutD Post-Authorizations

Post-Authorizations

In some cases, non-TroutD Post-Authorizations may need to occur. Specifically, if the merchant has called the processor's voice authorization center to receive an authorization code, the merchant will need to be able to manually enter the entire transaction as a Post-Authorization (action code 5). In this case, all fields that would typically be used for a standard Sale or Pre-Authorization should be entered plus the authorization code that the merchant received from the voice operator. Once this transaction is submitted to **PC**Charge, the transaction will be placed into the open batch.

Note: Voice-Authorizations will not qualify for the most favorable rates. The processor's Voice-Authorization system should be used only when absolutely necessary.

The following is an example of a Post-Authorization used to add a Voice-Authorization to the open batch:

Request:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<COMMAND>5</COMMAND>
		<PROCESSOR_ID>VISA</PROCESSOR_ID>
		<MERCH_NUM>999999999911</MERCH_NUM>
		<ACCT_NUM>5424180279791765</ACCT_NUM>
		<EXP_DATE>1211</EXP_DATE>
		<MANUAL_FLAG>0</MANUAL_FLAG>
		<TRANS_AMOUNT>6.00</TRANS_AMOUNT>
		<REFERENCE>123456</REFERENCE>
		<TICKET_NUM>999999999</TICKET_NUM>
		<CARDHOLDER>VERIFONE TEST 1</CARDHOLDER>
		<TOTAL_AUTH>6.00</TOTAL_AUTH>
		<AUTH_CODE>123456</AUTH_CODE>
	</XML_REQUEST>
</XML_FILE>
```

Response:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<TROUTD>1078</TROUTD>
		<RESULT>PROCESSED</RESULT>
		<AUTH_CODE>123456</AUTH_CODE>
		<TICKET>999999999</TICKET>
		<INTRN_SEQ_NUM>1078</INTRN_SEQ_NUM>
		<CMRCL_TYPE>0</CMRCL_TYPE>
		<PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
		<AUTH_SRC_CODE>E</AUTH_SRC_CODE>
		<CARD_ID_CODE>N</CARD_ID_CODE>
		<ACCT_DATA_SRC>@</ACCT_DATA_SRC>
	</XML_REQUEST>
</XML_FILE>
```

Notice that the AVS data (the street and zip code) and Card Verification data (CVC2) were omitted in this Post-Authorization. It is not necessary to pass this information because the transaction has already been authorized.

Note: If a post-authorization is sent with a TroutD and an authorization code, **PC**Charge will reject the transaction with an error of "Invalid TroutD or Auth Code". The format above for a voice-authorization must be followed if an authorization code is sent.
Note: When using the processor National Bankcard Services (NBS), Voice-Authorizations are not supported for Fuelman or Fleet One cards, an error will be returned. The transaction should be submitted as a standard Post-Authorization.

Stored Voice-Authorizations

Another type of Non-TroutD Post-Authorization is a "Stored Voice-Authorization". This type of transaction is similar to a Post-Authorization except that a stored Voice-Authorization is not added to the open batch. Instead, it essentially becomes a Pre-Authorization. A second transaction, a TroutD Post-Authorization, must be submitted to add the Stored Voice-Authorization to the batch. A Stored Voice-Authorization would be useful in a MOTO environment if an authorization was received from a voice operator the day a product was ordered, and product was not scheduled to ship until a few days later. The Stored Voice-Authorization would be processed the day the product was ordered, and the TroutD Post-Authorization would be processed the day that the product shipped. To submit a stored Voice-Authorization, use action code 5, and set the `Store` or `TRANS_STORE` flag. The following is an example of a Stored Voice-Authorization:

Request:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<COMMAND>5</COMMAND>
		<PROCESSOR_ID>VISA</PROCESSOR_ID>
		<MERCH_NUM>999999999911</MERCH_NUM>
		<ACCT_NUM>5424180279791765</ACCT_NUM>
		<EXP_DATE>1211</EXP_DATE>
		<MANUAL_FLAG>0</MANUAL_FLAG>
		<TRANS_AMOUNT>7.00</TRANS_AMOUNT>
		<REFERENCE>987654</REFERENCE>
		<TICKET_NUM>123456789</TICKET_NUM>
		<CARDHOLDER>VERIFONE TEST 1</CARDHOLDER>
		<TOTAL_AUTH>7.00</TOTAL_AUTH>
		<TRANS_STORE>1</TRANS_STORE>
		<AUTH_CODE>987654</AUTH_CODE>
	</XML_REQUEST>
</XML_FILE>
```

Note: Only some processors support stored Voice-Authorizations. If the processor does not support stored Voice-Authorizations, an error will be returned. The transaction should be submitted as a standard Post-Authorization.

REF_TROUTD

REF_TROUTD Transactions

An integrator may want to be able to take an existing transaction and reuse the card data to process a new transaction. In the case a TroutD from a successful transaction may be used to take the place of a Card Number and Expiration Date in a new, unrelated, transaction. This is supported only through the PCCharge API and will be allowed for transactions that are either settled or unsettled. As long as they were successful and still appead in the transaction database, they can be used.

- **Supported Tender Types:** Credit, Gift, Check, pinless Debit, and pinless EBT. However, PCCharge will allow the integrator to submit REF_TROUTD for any type of transaction. If the transaction requires track data or PIN data, then it will be rejected.

- The tender type must be the same as the original transaction. For example, PCCharge will not allow the integrator to attempt to pass in a REF_TROUTD Credit transaction using a TroutD from a Debit transaction.

Reversals

New mandates require that merchants have the ability to run Full and Partial Reversal transactions. Previously Reversals were tied to Pre-Paid Credit Cards and FSA/HRA transactions. Now, as long as we have certified for it and the processor supports it, the transaction can be reversed. This sends a transaction to **PC**Charge which actively goes to the processor in order to reverse the authorization hold placed on a credit card. The **PC**Charge GUI will have a new transaction tab marked "Reversal". Integrators can use the Action Code **P2** in order to process Reversals. Reversals should be treated like a Void, it should only be processed prior to End Of Day/Settlement/Batch Close. The Action Code **P2** has been available to use for Reversals since PCCharge version 5.8.1.

Processors that support Reversal transactions as of PCCharge version 5.9.1:

- BAMS FACTS (NPC)
- Elavon (NOVA) – Host and Terminal
- Fifth Third (BPS) – Terminal
- First Data North (CES)
- First Data Noth-Nash (FDNN)
- First Data Nashville (FDCN)
- Global Payments (NDC) – Host
- Heartland Payment Systems (HPTS)
- RBS WorldPay (LYNK)
- TSYS Acquiring (VISA)

Important Note: If the processor supports reversals and PCCharge has certified it, then merchants are being required to use a Reversal transaction in lieu of Void. This will allow the bank to remove the authorization hold on the Cardholder's available funds immediately rather than waiting for it to drop off naturally. Also, this is only supported with VISA, MasterCard, and Discover transactions. Any other card type attempted will receive a "Not Captured: Serv Not Allowed" response.

Example Full Reversal:

```
<XML_FILE>
      <XML_REQUEST>
            <USER_ID>User1</USER_ID>
            <COMMAND>P2</COMMAND>
            <PROCESSOR_ID>VISA</PROCESSOR_ID>
            <MERCH_NUM>999999999911</MERCH_NUM>
            <TROUTD>1024</TROUTD>
      </XML_REQUEST>
</XML_FILE>
```

Example Partial Reversal:

```
<XML_FILE>
      <XML_REQUEST>
            <USER_ID>User1</USER_ID>
            <COMMAND>P2</COMMAND>
            <PROCESSOR_ID>VISA</PROCESSOR_ID>
            <MERCH_NUM>999999999911</MERCH_NUM>
            <TROUTD>1024</TROUTD>
            <TRANS_AMOUNT>2.00</TRANS_AMOUNT>
      </XML_REQUEST>
</XML_FILE>
```

Commercial Card Transactions

Overview

A Commercial Card transaction is defined as any credit card transaction that requires the customer code and the tax amount of the transaction to be submitted. This information, commonly known as "Level II" data, is submitted for the customer's records and is required for the merchant to qualify for the lowest transaction rates. In general, if a customer is using one of the following types of credit cards, Level II data must be submitted:

- Commercial Cards
- Government Cards
- Purchasing Cards
- Procurement Cards
- Fleet Cards

If the end-user of an integrated application will accept Commercial Cards, the integrated application must be able to support the action codes and Level II data required to process Commercial Card transactions. Treating a Commercial Card transaction as a standard credit card transaction will cause the transaction to downgrade, thus increasing the fees the merchant must pay.

Note: Some processors do not support Level II data. A list of processors and what features they support can be found on VeriFone, Inc.'s website (http://www.verifone.com/). Click "**Developers**" and then "**PCCharge Certified Processors**" to view the list. The column labeled "**P-Card II**" indicates which processors support Level II data.

Supporting Commercial Card Transactions

The key to supporting Commercial Card transactions is the ability to identify a Commercial Card and then prompt for the Level II data. The first six digits of each credit card account number is called the Bank Identification Number (BIN) and the next three digits determine the card type. These first nine digits of credit card account numbers allow merchants to identify what type of credit card is being used. A database that includes valid Commercial Card BIN ranges is installed with each of copy of **PC**Charge. This database is named `Bin.mdb` and is installed automatically in the **PC**Charge directory. If the first nine digits of the credit card being submitted is within one of the ranges included in `Bin.mdb`, the integrated application should then prompt for Level II data.

Using Bin.mdb

OCX or DLL Method

If using the OCX or DLL Methods of integration, the `Charge.OCX` and the `PSCharge.dll` `Charge` class provide two methods that can be used to access the Commercial Card information contained in the `Bin.mdb` database.

The first method, CommercialCardType, is used to determine if a credit card is a commercial card. To use this method, set the Path variable, and then pass the credit card number as a parameter to the method. The method will return TRUE if the card is a Commercial Card, FALSE if it is not. For example:

```
If .CommercialCardType("Account Number") Then
  'Prompt for Level II data
Else
  'Do not Prompt for Level II data
End If
```

The second method, GetCommercialCardType, is used to determine the Commercial Card type. To use this method, set the Path variable, and then pass the credit card number as a parameter to the method. This method will return a character that indicates the Commercial Card Type. For example:

```
CommercialCardTypeChar = .getCommercialCardType("Account Number")
```

OLE/COM Method

If using the OLE/COM Method of integration, the PccBin class provides two functions that can be used to determine if a credit card is a commercial card and what type of commercial card it is.

The first function, CommercialCard, is a Boolean function that is used to determine if a credit card is a commercial card. To use this function, pass the credit card number as a parameter. This function will return TRUE if the card is a Commercial Card, FALSE if it is not. For example:

```
If .CommercialCard("Account Number") then
  'Prompt for Level II data
Else
  'Do not Prompt for Level II data
End If
```

The second function, CommercialCardType, is used to determine the Commercial Card type. The function does not require any parameters and must called after the CommercialCard function has been used. This function returns the commercial card type for the credit card number that was submitted via the CommercialCard function. This function will return a character that indicates the Commercial Card Type. For example:

```
CommercialCardTypeChar = .CommercialCardType
```

File Method or TCP Interface

If using the File Method or TCP Interface, integrated applications must access the Bin.mdb database directly. The following code sample is a Visual Basic 6 function that checks if a credit card account number falls into one of the Commercial Card BIN ranges in the Bin.mdb database.

Note: This code sample assumes that a reference to the "Microsoft DAO 2.5/3.51 Compatibility Library" has been set in the project.

```
Public Function CommercialCard (CreditCard As String) As Boolean

'Returns true if the credit card account number's BIN falls within a
```

```
'valid commercial card BIN range, else, returns false.

  Dim dbBin As Database
  Dim rsRange As Recordset
  Dim CCBIN, strQuery As String

  CCBIN = Left(CreditCard, 9) 'The Credit Card BIN is the first 9
                              'digits of the account number

  Set dbBin = OpenDatabase("C:\Program Files\Pccw\Bin.mdb")

  strQuery = "SELECT * FROM BIN WHERE LowRange <= " & CCBIN & _
             " AND HighRange >= " & CCBIN

  Set rsRange = dbBin.OpenRecordset(strQuery)

  If Not rsRange.EOF Then

  'if the account number BIN was found in the database, return True
    CommercialCard = True

  'Also, to determine the Commercial Card type, use the following
  'command to check the Type column: rsRange("Type").Value

  End If

End Function
```

Submitting Commercial Card Transactions

Once it has been determined that the credit card is a Commercial Card, the integrated application should perform the following steps:

1. Prompt for Level II data from the end-user and/or customer.

2. Create the Commercial Card transaction request. Pass the following values in the transaction request (refer to **CHAPTER 6 -- PCCharge Integration Methods** (see page 157) for information on the properties or tags that are used to pass the data):
   a. The request should contain all of the standard credit card Sale, Credit, or Post-Authorization values (card number, expiration date, amount, etc.)
   b. The request should contain Level II data (customer code and tax amount). The request should also indicate if the purchase is tax exempt (if applicable). The request should also include the Destination Zip Code if processing an American Express card and using American Express as your processor or via split dial. **Note:** Global East (NDC), terminal based, requires the customer code be all upper case.
   c. The request should contain the Commercial Card Flag character. (This is the character returned by the various methods above and is the character that appears in the Type column of the `Bin.mdb` database)
   d. Make certain that the request uses the proper Commercial Card action code. Valid Commercial Card codes are 8, 9, and 10. See the section **DevKit Constants** section (see page 143) for more information

3. Submit the transaction to **PC**Charge.

Example

The following is an example of a Commercial Card Sale using the File Method or TCP Interface.

Request:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<COMMAND>8</COMMAND>
		<PROCESSOR_ID>VISA</PROCESSOR_ID>
		<MERCH_NUM>999999999911</MERCH_NUM>
		<ACCT_NUM>4055011111111111</ACCT_NUM>
		<EXP_DATE>1211</EXP_DATE>
		<MANUAL_FLAG>1</MANUAL_FLAG>
		<TRANS_AMOUNT>5.30</TRANS_AMOUNT>
		<TRACK_DATA>4055011111111111=08121011000001234567</TRACK_DATA>
		<CUSTOMER_CODE>02</CUSTOMER_CODE>
		<TAX_AMOUNT>0.30</TAX_AMOUNT>
		<TICKET_NUM>123456789</TICKET_NUM>
		<CARDHOLDER>John Doe</CARDHOLDER>
		<CMRCL_FLAG>P</CMRCL_FLAG>
		<TAX_EXEMPT>0</TAX_EXEMPT>
	</XML_REQUEST>
</XML_FILE>
```

Response:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<TROUTD>1138</TROUTD>
		<RESULT>CAPTURED</RESULT>
		<AUTH_CODE>TSYS 4</AUTH_CODE>
		<REFERENCE>411921002550</REFERENCE>
		<AVS_CODE>0</AVS_CODE>
		<TRANS_DATE>042804</TRANS_DATE>
		<TICKET>123456789</TICKET>
		<INTRN_SEQ_NUM>1138</INTRN_SEQ_NUM>
		<TRANS_ID>000000000009548</TRANS_ID>
		<MSI>E</MSI>
		<PEM>5</PEM>
		<TIM>173952</TIM>
		<ACI>E</ACI>
		<PROC_RESP_CODE>00</PROC_RESP_CODE>
		<CMRCL_TYPE>0</CMRCL_TYPE>
		<PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
		<CARD_ID_CODE>@</CARD_ID_CODE>
		<ACCT_DATA_SRC>D</ACCT_DATA_SRC>
	</XML_REQUEST>
</XML_FILE>
```

Flexible Spending Accounts (FSA/HRA) Transactions

Overview

FSA - Flexible Spending Account
HRA - Health Reimbursement Arrangement
IIAS - Inventory Information Approval System

☛	**FSA/HRA** transactions are **now** supported with **PC**Charge **Payment Server** and **PC**Charge **Pro** version **5.8.1**.

An IIAS program is an inventory control system that helps to categorize saleable items. FSA/HRA deals with the healthcare-related categories. With this system in place, the merchant will know based on an item's SKU number whether or not it is FSA/HRA eligible. When a customer purchases items, the IIAS system will automatically flag the healthcare-related items so that the customer can purchase using an FSA/HRA card.

Note: PCharge *has nothing to do with IIAS. PCCharge assumes that there is an IIAS system in place and that the merchant will input the proper amounts for the various healthcare categories.*

Processors That Support FSA/HRA transactions.

Processor	PCCharge Availablity (In Version)	Partial Auth Supported
Alliance Data (ADSI)	Future Release	N/A
TANDEM BAMS (NPC)	5.8.1	Yes
Echo (ECHO)	Future Release	N/A
Elavon (NOVA)	5.8.1	Yes
Fifth Third St. Pete (BPS) Host	5.8.1	Yes
Fifth Third St. Pete (BPS) Terminal	5.8.1	N/A
First Data Atlanta/ Buypass (BPAS)	5.8.2	Yes
First Data Omaha (FDC)	Future Release	N/A
First Data South (NB)	Future Release	N/A
Global Payments East (NDC)	Future Release	N/A
Heartland Payments (HPTS)	5.8.1	Yes
First Data New Nashville (FDNN)	5.8.2	Yes
First Data North (CES)	5.8.1	Yes
First Data Nashville (FDCN)	Future Release	N/A
National Bankcard (NBS)	Future Release	N/A
Paymentech Tampa (GSAR)	5.8.1	Yes
RBS WorldPay (LYNK)	5.8.2	Yes
TSYS Acquiring (VISA)	5.8.1	Yes

Multi-purpose FSA Cards

FSA/HRA cards can be loaded for the purchase of just healthcare-related items or they can be loaded with other benefits as well (multi-purse).

- If the FSA/HRA card is **not multi-purse**, and the customer wants to purchase a combination of healthcare and non-healthcare items, a Partial Approval situation will occur. The processing platform will approve the healthcare amount, if available on the card, and the customer will be asked to pay for the remainder of the bill with some other tender type.

- If the FSA/HRA card is a **multi-purse**, the customer can purchase whatever items they choose using this card and a Partial Approval will only be triggered if there is not enough open to buy on the card to satisfy the bill.

FSA/HRA cards are similar to Pre-Paid credit cards. The only difference is that the FSA value on the card may only be used to for healthcare-related purchases. There are two ways to load money onto the cards:

- Payroll Deduction

- The employer can load the value directly to the card.

Note: HRA cards can only be loaded by the employer as this is reimbursement for healthcare expense.

IRS Mandated Guidelines

According to IRS guidelines, merchants will be mandated to participate in the IIAS program in order to accept FSA/HRA cards beginning in 2009 if they do not have a Drug Store or Pharmacy Merchant Category Code (MCC/SIC). The current MCC/SICs affected are:

MCC	Description
5411	Grocery Stores and Supermarkets
5310	Discount Stores
5300	Discount Goods – Wholesale Clubs
5499	Convenience Stores
5960	Online Pharmacies
5964	
5965	
5969	
5912	Drug Stores and Pharmacies (effective 01/01/09)
5122	Drugist/Druggist Proprietaries (effective 01/01/09)

Other MCC/SICs not listed above can also be affected as long as they are non-healthcare related MCCs which sell healthcare related items.

Split Tender

Due to the nature of the program, it will be possible for the cardholder to use an FSA/HRA card for a partial payment for eligible items and then use a different tender type or card for the remainder of the items that do not qualify as eligible. This partial payment can also occur if all items are eligible but there is not enough available on the card to pay for the entire purchase.

Eligible Purchases

Eligible items currently include:

- Prescriptions (including over the counter medications)
- Vision care
- Clinical (doctor visits, labs, etc)
- Dental

Note: The list of eligible medical expenses is part of the Internal Revenue Code. The merchant is responsible for the maintenance of all eligible SKUs so that when a purchase is made, the eligible items can purchased using a FSA/HRA card.

Card Types Currently Supported

FSA/HRA cards currently are only Visa/MC. There are specific BIN ranges for these cards.

Note: FSA/HRA transactions are NOT supported for the Restaurant industry. Therefore, any FSA settings/enablement will be greyed out in the setup for processing platforms that support both FSA and Restaurant when Restaurant is the selected industry. For example, if you go to the TSYS Extended Data fields and choose Retail, MOTO or Ecommerce as the industry, you can enable FSA/HRA. If you choose Restaurant as the industry, the FSA enablement will be greyed out and unavailable.

FSA/HRA Examples

Successful FSA transaction.

```
INX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>1</COMMAND>
   <PROCESSOR_ID>NPC</PROCESSOR_ID>
   <MERCH_NUM>000000000011</MERCH_NUM>
   <ACCT_NUM>5439750001500347</ACCT_NUM>
   <EXP_DATE>1210</EXP_DATE>
   <MANUAL_FLAG>0</MANUAL_FLAG>
   <TRANS_AMOUNT>5.00</TRANS_AMOUNT>
   <FSA>1</FSA>
   <AMOUNT_HEALTHCARE>5.00</AMOUNT_HEALTHCARE>
   <AMOUNT_PRESCRIPTION>1.00</AMOUNT_PRESCRIPTION>
   <AMOUNT_CLINIC>1.00</AMOUNT_CLINIC>
   <AMOUNT_VISION>1.00</AMOUNT_VISION>
   <AMOUNT_DENTAL>1.00</AMOUNT_DENTAL>
</XML_REQUEST>

OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <TROUTD>10540</TROUTD>
   <RESULT>CAPTURED</RESULT>
   <AUTH_CODE>TAS992</AUTH_CODE>
   <REFERENCE>830120501440</REFERENCE>
   <AVS_CODE>0</AVS_CODE>
   <TRANS_DATE>102708</TRANS_DATE>
   <INTRN_SEQ_NUM>10540</INTRN_SEQ_NUM>
   <TRANS_ID>0000123456789</TRANS_ID>
   <TICODE>CK</TICODE>
   <MSI>A</MSI>
   <RET>M</RET>
   <PEM>5</PEM>
   <TIM>155502</TIM>
   <ACI>A</ACI>
   <PROC_RESP_CODE>00</PROC_RESP_CODE>
```

```
   <CMRCL_TYPE>0</CMRCL_TYPE>
   <PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
   <CMRCL_FLAG>N</CMRCL_FLAG>
   <CARD_ID_CODE>M</CARD_ID_CODE>
   <ACCT_DATA_SRC>T</ACCT_DATA_SRC>
</XML_REQUEST>
```

Failed FSA/HRA transaction.

```
INX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>1</COMMAND>
   <PROCESSOR_ID>NPC</PROCESSOR_ID>
   <MERCH_NUM>000000000011</MERCH_NUM>
   <ACCT_NUM>4387753333333333</ACCT_NUM>
   <EXP_DATE>1210</EXP_DATE>
   <MANUAL_FLAG>0</MANUAL_FLAG>
   <TRANS_AMOUNT>5.00</TRANS_AMOUNT>
   <FSA>1</FSA>
   <AMOUNT_HEALTHCARE>4.00</AMOUNT_HEALTHCARE>
   <AMOUNT_PRESCRIPTION>1.00</AMOUNT_PRESCRIPTION>
   <AMOUNT_CLINIC>1.50</AMOUNT_CLINIC>
   <AMOUNT_VISION>1.25</AMOUNT_VISION>
   <AMOUNT_DENTAL>1.00</AMOUNT_DENTAL>
</XML_REQUEST>

OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <TROUTD>10495</TROUTD>
   <RESULT>Error</RESULT>
   <AUTH_CODE>Invalid Dollar Amount</AUTH_CODE>
   <REFERENCE>Health Care Amount must be greater than Sub Amounts</REFERENCE>
   <INTRN_SEQ_NUM>10495</INTRN_SEQ_NUM>
</XML_REQUEST>
```

Partial Approval of an FSA/HRA Transaction:

```
INX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>1</COMMAND>
   <PROCESSOR_ID>NPC</PROCESSOR_ID>
   <MERCH_NUM>000000000011</MERCH_NUM>
   <ACCT_NUM>4387753333333333</ACCT_NUM>
   <EXP_DATE>1210</EXP_DATE>
   <MANUAL_FLAG>0</MANUAL_FLAG>
   <TRANS_AMOUNT>5.00</TRANS_AMOUNT>
   <FSA>1</FSA>
   <AMOUNT_HEALTHCARE>4.00</AMOUNT_HEALTHCARE>
   <AMOUNT_PRESCRIPTION>1.00</AMOUNT_PRESCRIPTION>
   <AMOUNT_CLINIC>0.50</AMOUNT_CLINIC>
   <AMOUNT_VISION>1.25</AMOUNT_VISION>
   <AMOUNT_DENTAL>1.00</AMOUNT_DENTAL>
</XML_REQUEST>

OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <TROUTD>10496</TROUTD>
   <RESULT>CAPTURED</RESULT>
   <AUTH_CODE>TAS185</AUTH_CODE>
   <REFERENCE>829814501105</REFERENCE>
   <AVS_CODE>0</AVS_CODE>
   <TRANS_DATE>102408</TRANS_DATE>
   <INTRN_SEQ_NUM>10496</INTRN_SEQ_NUM>
   <TRANS_ID>000000000894349</TRANS_ID>
   <TICODE>CK</TICODE>
   <MSI>N</MSI>
   <RET>M</RET>
   <PEM>5</PEM>
   <TIM>104858</TIM>
   <ACI>N</ACI>
   <PROC_RESP_CODE>10</PROC_RESP_CODE>
   <CMRCL_TYPE>0</CMRCL_TYPE>
   <PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
   <CARD_ID_CODE>M</CARD_ID_CODE>
   <ACCT_DATA_SRC>T</ACCT_DATA_SRC>
   <AMOUNT_DUE>1.00</AMOUNT_DUE>
   <AUTH_AMOUNT>4.00</AUTH_AMOUNT>
</XML_REQUEST>
```

☛	**Note :** Integrators will use either the AMOUNT_DUE or the AUTH_AMOUNT tags to have the POS indicate the remaining purchase amount to the end user.

☛	**Note :** When total cost of the transaction is covered by the initial purchase PCCharge will not return AMOUNT_DUE and AUTH_AMOUNT tags.

Discretionary Data

This allows an integrator to submit transaction descriptions with certain transaction types and processors. Once enabled, PCCharge will store the data and submit it to the processor during settlement. This is typically enabled on the Extended Data setup screen within PCCharge.

Processors That Support Discretionary Data

Processor	Transaction Type	Tender Type
NBS	Sale, Credit, Pre-Auth	Credit
TANDEM BAMS (NPC)	Sale, Credit, Pre-Auth, Book, Voice-Auth	Credit, Debit, EBT

Example transaction with Discretionary Data :

```
INX:
<XML_REQUEST>
       <USER_ID>User1</USER_ID>
       <COMMAND>1</COMMAND>
       <PROCESSOR_ID>NPC</PROCESSOR_ID>
       <MERCH_NUM>999999999911</MERCH_NUM>
       <ACCT_NUM>4387753333333333</ACCT_NUM>
       <EXP_DATE>1212</EXP_DATE>
       <MANUAL_FLAG>0</MANUAL_FLAG>
       <TRANS_AMOUNT>2.00</TRANS_AMOUNT>
       <ZIP_CODE>85284</ZIP_CODE>
       <STREET>8320</STREET>
       <TICKET_NUM>11</TICKET_NUM>
       <CVV2>999</CVV2>
       <PRESENT_FLAG>1</PRESENT_FLAG>
       <PRODUCT_DETAIL_CODE_1>Line Item 1 - Test</PRODUCT_DETAIL_CODE_1>
       <PRODUCT_DETAIL_CODE_2>Line Item 2 - Test</PRODUCT_DETAIL_CODE_2>
       <PRODUCT_DETAIL_CODE_3>Line Item 3 - Test</PRODUCT_DETAIL_CODE_3>
</XML_REQUEST>

OUX:
<XML_REQUEST>
       <USER_ID>User1</USER_ID>
       <TROUTD>1176</TROUTD>
       <RESULT>CAPTURED</RESULT>
       <AUTH_CODE>TAS680</AUTH_CODE>
       <REFERENCE>833116503763</REFERENCE>
       <AVS_CODE>Y</AVS_CODE>
       <TRANS_DATE>112608</TRANS_DATE>
       <TICKET>11</TICKET>
       <INTRN_SEQ_NUM>1176</INTRN_SEQ_NUM>
       <TRANS_ID>000000000301523</TRANS_ID>
       <TICODE>CK</TICODE>
       <MSI>N</MSI>
       <PEM>5</PEM>
       <TIM>113023</TIM>
       <ACI>N</ACI>
       <PROC_RESP_CODE>00</PROC_RESP_CODE>
       <CVV2_CODE>M</CVV2_CODE>
       <CMRCL_TYPE>0</CMRCL_TYPE>
       <PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
       <CMRCL_FLAG>N</CMRCL_FLAG>
       <CARD_ID_CODE>M</CARD_ID_CODE>
       <ACCT_DATA_SRC>T</ACCT_DATA_SRC>
```

</XML_REQUEST>

Restaurant Transactions

Overview

Integrating **PC**Charge into a restaurant point-of-sale system is similar to integrating **PC**Charge into any other type of retail point-of-sale system. Once Restaurant is selected as the business type in the processor's extended data fields, **PC**Charge will be able to send "restaurant certified" transactions to the processor.

Note: Some processors do not support restaurant transactions. A list of processors and what features they support can be found on VeriFone, Inc.'s website (http://www.pccharge.com/). Click **"Support"** and then **"PCCharge DevKit"** and then **"PCCharge Certified Processing Companies"** to view the list. The column labeled **"Restaurant"** indicates which processors support restaurant transactions.

Benefits of XML

Starting with **PC**Charge version 5.6.0, the **PC**Charge products were updated to support restaurant integration using the XML message format. **If an existing integration was written using the INP message format, it is highly recommended that the integration be updated to use the XML message format**. Problems may occur with reports and when adding gratuities when using the INP message format—these problems have been resolved with the XML message format. In addition, the XML message format supports gratuity adjustments, the INP message format does not.

- **The INP message format will eventually be phased out.**

Note: The XML message format is available in all five integration methods. To activate the XML message format when using the OCX, DLL, or OLE/COM methods of integration, use the parameter "3" when calling the Send method. For example:

OCX / DLL: `.Send 3`
OLE/COM: `.Send , 3`

Integration

To properly send restaurant transactions from an integrated application to **PC**Charge, restaurant transactions should basically be treated as standard retail transactions. In addition, include the following information in restaurant transaction requests:

- **Estimated gratuity** – This amount may be sent with a Sale (action code 1) or a Pre-Authorization (action code 4) transaction. The amount that **PC**Charge sends to the processor for authorization is the sale amount + the estimated gratuity. Sending the estimated gratuity helps to assure that the customer has enough available credit to leave a tip. The amount that is recorded for settlement by **PC**Charge (or the processor, if Host based) is the sale amount only. **Note:** It is recommended to check with the processor or merchant service provider for guidance on what amount to set this value to. Incorrectly setting this value can result in downgrades.

- **Gratuity** – Sending the gratuity is the second step of a restaurant transaction. If the original transaction was a Sale, send the gratuity amount in a gratuity transaction (action code 13). A

Gratuity transaction adds the actual gratuity to the amount recorded for settlement. If the original transaction was a Pre-Authorization, a Post-Authorization (action code 5) should be used to complete the Pre-Authorization. This completion records the sale amount plus the gratuity for settlement. A Sale with Gratuity (action code 14) transaction authorizes and records the sale amount plus gratuity in one step. A Gratuity transaction (action code 13) may also be used to adjust an existing gratuity prior to settlement.

Server ID -- This value should be sent with all restaurant Sale (action code 1), Pre-Authorization (action code 4), or Sale with Gratuity (action code 14) transactions. This two digit ID is stored in the transaction record and is used in **PC**Charge's gratuity reports to determine the amount of Gratuity each server has received. This value is required by some processors. To force entry of this value for each transaction, the **Require Server ID** box should be checked in the processor's extended data fields in **PC**Charge. Use the MCSN tag or property to pass the Server ID.
Processor specific note: The Server ID is required for AMEX card transactions. Also required when using the processor NB and GSAR in restaurant business type.

Examples

The following are examples, using the File Method or TCP Interface, that show common transactions used in a restaurant environment. For more information on the File Method of integration, refer to the section **File Method** (see page 481).

Restaurant Sale

The following is an example of a restaurant sale for $6.00 with an estimated gratuity of $0.85. The total amount that will be authorized is $6.85. However, only $6.00 will be added to the settlement file. A gratuity transaction must be submitted later to finalize (add the actual gratuity amount to) this transaction. **Note about estimated gratuity:** It is recommended to check with the processor or merchant service provider for guidance on what amount to set this value to. Incorrectly setting this value can result in downgrades.

Request:

```
<XML_FILE>
       <XML_REQUEST>
              <USER_ID>User1</USER_ID>
              <COMMAND>1</COMMAND>
              <PROCESSOR_ID>GSAR</PROCESSOR_ID>
              <MERCH_NUM>999999999999519</MERCH_NUM>
              <ACCT_NUM>4012000033330026</ACCT_NUM>
              <EXP_DATE>1211</EXP_DATE>
              <MANUAL_FLAG>1</MANUAL_FLAG>
              <TRANS_AMOUNT>6.00</TRANS_AMOUNT>
              <TRACK_DATA>4012000033330026=08121011000001234567</TRACK_DATA>
              <TICKET_NUM>9999</TICKET_NUM>
              <CARDHOLDER>VERIFONE TEST 3</CARDHOLDER>
              <MCSN>02</MCSN>
              <PRESENT_FLAG>1</PRESENT_FLAG>
              <GRATUITY_AMNT_EST>0.85</GRATUITY_AMNT_EST>
       </XML_REQUEST>
</XML_FILE>
```

Response:

```
<XML_FILE>
       <XML_REQUEST>
              <USER_ID>User1</USER_ID>
              <TROUTD>1102</TROUTD>
              <RESULT>CAPTURED</RESULT>
              <AUTH_CODE>096899</AUTH_CODE>
              <REFERENCE>00000000</REFERENCE>
              <TRANS_DATE>0404</TRANS_DATE>
              <TICKET>9999</TICKET>
              <INTRN_SEQ_NUM>1102</INTRN_SEQ_NUM>
              <PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
       </XML_REQUEST>
</XML_FILE>
```

Because the Sale transaction was successful, the Restaurant Point-of-Sale should store the TroutD value (1102 for this transaction) to enable the server to add the gratuity.

Gratuity

The following is an example of a Gratuity Transaction used to finalize a sale. In this example, the Gratuity transaction is finalizing the Restaurant sale from above. To finalize the Sale, pass the Gratuity action code (13), the TroutD from the original Sale (1102 for this transaction), and the actual Gratuity amount ($1.00 for this transaction). A Result of "GRATUITY ADDED" indicates the gratuity was successfully added to the Sale.

Request:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<COMMAND>13</COMMAND>
		<TROUTD>1102</TROUTD>
		<GRATUITY_AMNT>1.00</GRATUITY_AMNT>
	</XML_REQUEST>
</XML_FILE>
```

Response:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<TROUTD>1102</TROUTD>
		<RESULT>GRATUITY ADDED</RESULT>
		<TRANS_DATE>0404</TRANS_DATE>
		<INTRN_SEQ_NUM>1103</INTRN_SEQ_NUM>
		<PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
	</XML_REQUEST>
</XML_FILE>
```

Gratuity Adjustment

A gratuity adjustment is defined as a transaction that modifies a gratuity that has already been processed. Typically, a gratuity adjustment would be used to correct order-entry errors. The syntax for a gratuity adjustment is exactly the same as a gratuity. Simply pass in the Gratuity action code (13), the TroutD from the original Sale (1102 for this transaction), and the new Gratuity amount ($2.00 for this transaction).

Request:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<COMMAND>13</COMMAND>
		<TROUTD>1102</TROUTD>
		<GRATUITY_AMNT>2.00</GRATUITY_AMNT>
	</XML_REQUEST>
</XML_FILE>
```

Response:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<TROUTD>1102</TROUTD>
		<RESULT>GRATUITY ADDED</RESULT>
		<TRANS_DATE>0404</TRANS_DATE>
		<INTRN_SEQ_NUM>1109</INTRN_SEQ_NUM>
		<PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
	</XML_REQUEST>
```

```
</XML_FILE>
```

Processor Specific Notes

The processors Global Payments East (NDC) and FDMS South (NB) have unique restrictions related to gratuity based transactions.

These two processors will not allow the actual total sale plus gratuity amount to exceed 120% of the originally authorized total sale plus gratuity amount.

Example:

1. The customer's bill is $10.00. The server estimates the gratuity will be $1.00 (10%). The restaurant application will then submit a Sale transaction (action code 1) with the amount of $10.00 and the estimated gratuity of $1.00. Therefore, **PC**Charge will request an authorization for $11.00. This authorization is approved by the processor.

2. After the meal has concluded, the customer decides to leave a gratuity of $5.00 This gratuity amount would result in a total amount of $15.00, which is greater than 120% of the originally authorized total of $11.00. The merchant could not, in this situation, add the gratuity by performing a Gratuity transaction.

3. Instead, the server will need to perform a Void Sale (action code 3) on the original transaction, and then enter a Sale with Gratuity (action code 14) with the amount of $10.00 and a gratuity amount of $5.00.

Gift Card Transactions

This section details the various action codes and transaction types that are applicable for each Gift Card processor. Although several of the transaction types are similar among all Gift Card processors (such as Balance or Redemption transactions), most Gift Card processors support very unique transaction types. It is recommended that the developer review the various screens in the **PC**Charge GUI to determine what values should be set for each transaction type. Also, many Gift Card processors also recommend that developers contact them directly to discuss the usage and integration of their services.

Note: VeriFone, Inc. does not provide test merchant accounts for any of the Gift Card processors. Developers should contact the various Gift Card processors directly to request test merchant accounts.

Givex (GVEX)

18 – Balance
25 – Redemption
26 – Register
27 – Increment
28 – Activate
29 – Cancel
0Q – Adjustment (amount may be either positive or negative)
0N – Points

Valuetec (VTEC)

18 – Balance
25 – Redemption (Loyalty) / Sale (Non-Loyalty)
26 – Replace
27 – Purchase (Loyalty) / Add Value (Non-Loyalty)
28 – Activation
29 – Void
0A – Deactivate
0C – Current Day Totals
0D – Previous Day Totals

ValueLink (VLNK)

18 – Balance
28 – Activate
25 – Redemption
27 – Reload
29 – Void
0H – (Restaurant OR E-Commerce) with Simulate Pre-Auth Set up: Balance with Lock
0I – (Restaurant OR E-Commerce) with Simulate Pre-Auth Set up: Redemption Unlock
0P – Balance Merge
0Q – Balance Adjustment
0R – Balance Transfer
0S – Report Lost/Stolen
0T – Cash Out

Chase Paymentech (GSAR)

18 – Balance Inquiry
25 – Redemption & Prior Redemption
27 – Issuance/Add Value
29 – Void
0A – Deactivate
0E – Reactivate

World (WRLD)

18 – Balance
25 – Sale
27 – Add Value
28 – Activate
0N – Redeem Points

Fifth-Third Bank-St Pete (BPS)

18 – Balance
25 – Redemption
27 – Reload
28 – Activate
29 – Void
0A – Close
0B – Refund
0Q – Unload
0V – Pre-Auth
0I – Post-Auth

RBS WorldPay (LYNK)

18 – Balance
25 – Redemption
27 – Add Value
29 – Void Add value
E1 – Gift Settlement

Datamark Gift Card (DMRK)

18 – Balance Inquiry
25 – Redemption & Prior Redemption
27 – Issuance/Add Value
29 – Void
0A – Deactivate
0E – Reactivate

Smart Transaction Systems (SMTS)

18 – Balance Inquiry
25 – Redemption
27 – Add Value

28 – Activation
29 – Void
0B – Credit
0C – Totals Inquiry (Current only)
0N – Redeem Points
0R – Balance Transfer
0U – Add Tip

Secure Payment Systems (SPS)

18 – Balance Inquiry
25 – Redemption
27 – Add Value/Points
28 – Activation
29 – Void
0B – Credit
0C – Totals Inquiry
0R – Balance Transfer
0N – Redeem Points
0A – Deactivate Points

Stored Value Systems (SVSI)

18 – Balance Inquiry
25 – Redemption
27 – Issuance
28 – Activation
29 – Void
0B – Refund
0I – Post-Auth
0Q – Recharge
0T – Cash Out
0U – Tip
0V – Pre-Auth

TSYS Acquiring Solutions (VISA)

18 – Balance
25 – Redemption
27 – Reload
0B – Refund
0A – Deactivate (Only Supported for Fifth Third RIID)
0Q – Unload (Only Supported for Fifth Third RIID)

VeriFone Stored Value API (GAPI)

The VeriFone Stored Value API (GAPI) is a proprietary specification that allows for stored value card processors to add themselves to **PC**Charge. Applications using GAPI can also integrate with **PC**Charge using the various integration methods. For more information on adding a stored value card processor to **PC**Charge, and how to obtain the VeriFone Stored Value API, please contact VeriFone sales at 1-800-725-9264.

Pre-Paid Credit Card Transactions

Pre-paid credit cards are similar to regular credit cards, the only difference is that they carry a fixed amount. **PC**Charge supports Visa, MasterCard, American Express and Discover pre-paid card. Pre-Paid Credit Card processing was added in *PCCharge version 5.7.1 release I sp6*, but not made configurable until *PCCharge version 5.7.1 release I sp9a*.

Action Codes:
P1 - Balance Inquiry
P2 - Partial Authorization Reversal
Regular Credit card action codes can be used.

P1: Used to determine the balance available in a pre-paid credit card.

P2: This transaction format is used to submit a POS generated "Reversal" message for a previously approved "Partial Authorization" Credit Card transaction. This transaction type is only supported for the Visa, MasterCard and Discover card types and will return "SERV NOT ALLOWED" for any other transaction.

This transaction type is not supported for any PIN Based Debit card transactions.

Note: Reversals are not available for American Express pre-paid credit cards.

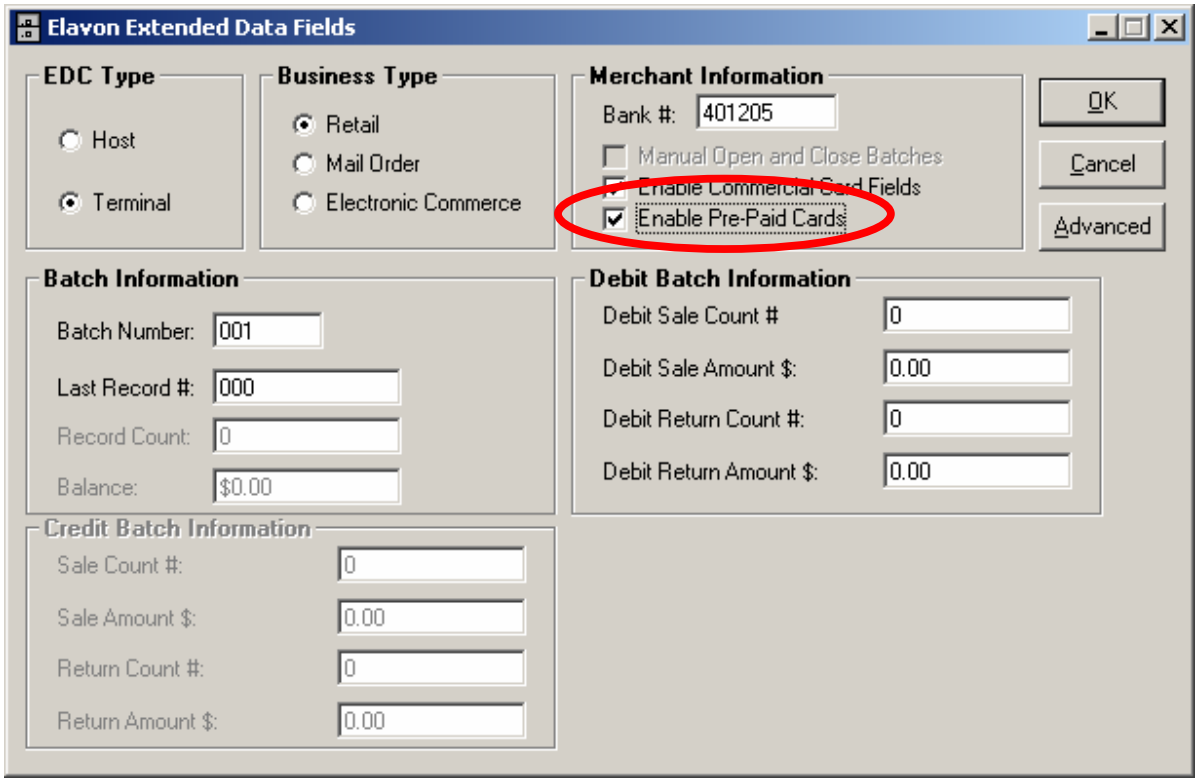
Pre-Paid functionality was originally added in *PCCharge version 5.7.1 release I sp6*. This feature was defaulted to "on" and the setting could not be changed. The merchant was required to contact their merchant account representative and opt-out of the program. Starting with *PCCharge version 5.7.1 release I sp9a* and now *version 5.8*, VeriFone has provided the merchant with a choice within the PCCharge user interface.

When enabled: A merchant runs a Pre-Paid card and the amount being processed is greater than the amount on the card. The processor will send an approval message with the approved amount. PCCharge will then calculate the AMOUNT DUE and passes it to the integrator.

When disabled: A merchant runs a Pre-Paid card and the amount being processed is greater than the amount on the card. The processor will send a declined message. If the card is Pre-Paid and the amount being processed is less than the card has open to buy, The processor will approve the transaction as if it is a normal credit card.

Note: If "Enable Pre-Paid Cards" is not checked, the merchant will not be able to run Credit or Debit Balance Inquiries as this functionality is tied to the Partial Authorization functionality.

Note: "Enable Pre-Paid Cards" option within PCCharge MUST be checked for this feature to work. The option is located under Setup>Credit Card Company>Extended. This option is only configurable in *PCCharge version 5.7.1 release I sp9a* and above.



Partial Authorization (Partial Auth)

In order for an integrator to process transactions where the initial form of payment does not cover the entire authorization amount, the POS must be ready to accept an AMOUNT_DUE that is returned from PCCharge. When this tag is present it indicates that an amount smaller than the original transaction amount was authorized. The next step is for the POS to send a new transaction with the remaining payment amount. As with Pre-Paid Transactions and FSA/HRA Partial Authorization may be an option that can be turned on or off within PCCharge and is usually found on the Extended Data screen for supporting processors.

Example of a Partial Auth Transaction:

INX:

```
<XML_REQUEST>
    <USER_ID>User1</USER_ID>
    <COMMAND>1</COMMAND>
    <PROCESSOR_ID>NPC</PROCESSOR_ID>
    <MERCH_NUM>000000000011</MERCH_NUM>
    <ACCT_NUM>4387753333333333</ACCT_NUM>
    <EXP_DATE>1210</EXP_DATE>
    <MANUAL_FLAG>0</MANUAL_FLAG>
    <TRANS_AMOUNT>5.00</TRANS_AMOUNT>
</XML_REQUEST>
```

OUX:

```
<XML_REQUEST>
    <USER_ID>User1</USER_ID>
    <TROUTD>10496</TROUTD>
    <RESULT>CAPTURED</RESULT>
    <AUTH_CODE>TAS185</AUTH_CODE>
    <REFERENCE>829814501105</REFERENCE>
    <AVS_CODE>0</AVS_CODE>
    <TRANS_DATE>102408</TRANS_DATE>
    <INTRN_SEQ_NUM>10496</INTRN_SEQ_NUM>
    <TRANS_ID>000000000894349</TRANS_ID>
    <TICODE>CK</TICODE>
    <MSI>N</MSI>
    <RET>M</RET>
    <PEM>5</PEM>
    <TIM>104858</TIM>
    <ACI>N</ACI>
    <PROC_RESP_CODE>10</PROC_RESP_CODE>
    <CMRCL_TYPE>0</CMRCL_TYPE>
    <PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
    <CARD_ID_CODE>M</CARD_ID_CODE>
    <ACCT_DATA_SRC>T</ACCT_DATA_SRC>
    <AMOUNT_DUE>1.00</AMOUNT_DUE>
    <AUTH_AMOUNT>4.00</AUTH_AMOUNT>
</XML_REQUEST>
```

☛	**Note :** Integrators will use either the AMOUNT_DUE or the AUTH_AMOUNT tags to have the POS indicate the remaining purchase amount to the end user.

☛	**Note :** When total cost of the transaction is covered by the initial purchase PCCharge will not return AMOUNT_DUE and AUTH_AMOUNT tags.

Citi© Transactions/ Credit Plan

PCCharge 5.8.0 will now have the ability to process Credit Applications as well as Account Lookups for Citi©.

Action Code	Description
P3	Credit Application
P4	Account Lookup
P5	Counter Offer **Note:** Follows a P3 when the result is OFFER.

Additional XML tags for the INX:
<MIDDLE_INITIAL>, <SUFFIX>, <APT_SUITE>, <EMAIL>, <EMP_NAME>, <HOME_OWNER>, <ANNUAL_INCOME>, <PHOTO_ID_STATE>, <WORK_PHONE_NUM>, <SOURCE_CODE>

Additional XML tags for the OUX:
<CORRELATION_UID>, <PENDING_NUMBER>, <REPLY_FLAG>, <OPEN_TO_BUY>

Credit Application (P3)

Approved
INX:

```
<XML_REQUEST>
  <USER_ID>User1</USER_ID>
  <COMMAND>P3</COMMAND>
  <PROCESSOR_ID>CITI</PROCESSOR_ID>
  <MERCH_NUM>512771092345</MERCH_NUM>
  <MANUAL_FLAG>0</MANUAL_FLAG>
  <TRANS_AMOUNT>1233.00</TRANS_AMOUNT>
  <ZIP_CODE>85284</ZIP_CODE>
  <STREET>8320</STREET>
  <STATE>GA</STATE>
  <PHONE_NUM>9123646469</PHONE_NUM>
  <DOB>02022002</DOB>
  <CUSTOMER_FIRSTNAME>Andy</CUSTOMER_FIRSTNAME>
  <CUSTOMER_LASTNAME>Apple</CUSTOMER_LASTNAME>
  <CUSTOMER_CITY>SAV</CUSTOMER_CITY>
  <ID_TYPE>D</ID_TYPE>
  <ID_NUMBER>121321313</ID_NUMBER>
  <SS_NUM>999999999</SS_NUM>
  <WORK_PHONE_NUM>9123665659</WORK_PHONE_NUM>
  <MIDDLE_INITIAL>Ad</MIDDLE_INITIAL>
  <APT_SUITE>8320</APT_SUITE>
  <EMAIL>Andy@verifone.com</EMAIL>
  <EMP_NAME>Verifone</EMP_NAME>
  <HOME_OWNER>O</HOME_OWNER>
  <ANNUAL_INCOME>12125</ANNUAL_INCOME>
  <PHOTO_ID_STATE>GA</PHOTO_ID_STATE>
  <SOURCE_CODE>AP</SOURCE_CODE>
</XML_REQUEST>
```

OUX:

```
<XML_REQUEST>
```

```
   <USER_ID>User1</USER_ID>
   <TROUTD>10587</TROUTD>
   <RESULT>APPROVAL</RESULT>
   <AUTH_CODE>Please write down the Acct# & credit limit on Customer
Application.</AUTH_CODE>
   <TRANS_DATE>072208</TRANS_DATE>
   <INTRN_SEQ_NUM>10587</INTRN_SEQ_NUM>
   <TIM>153055</TIM>
   <CMRCL_TYPE>0</CMRCL_TYPE>
   <PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
   <CARD_ID_CODE>@</CARD_ID_CODE>
   <ACCT_DATA_SRC>T</ACCT_DATA_SRC>
   <ACCT_NUM>1234123412341234</ACCT_NUM>
   <CREDIT_LIMIT>1.00</CREDIT_LIMIT>
</XML_REQUEST>
```

Rejected

INX: This transaction rejected as the <SOURCE_CODE> was sent as 'AO" and the Opt Out trigger was not turned ON in the PCCharge setup

```
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>P3</COMMAND>
   <PROCESSOR_ID>CITI</PROCESSOR_ID>
   <MERCH_NUM>512771092345</MERCH_NUM>
   <MANUAL_FLAG>0</MANUAL_FLAG>
   <TRANS_AMOUNT>1233.00</TRANS_AMOUNT>
   <ZIP_CODE>85284</ZIP_CODE>
   <STREET>8320</STREET>
   <STATE>GA</STATE>
   <PHONE_NUM>9123646469</PHONE_NUM>
   <DOB>02022002</DOB>
   <CUSTOMER_FIRSTNAME>Andy</CUSTOMER_FIRSTNAME>
   <CUSTOMER_LASTNAME>Apple</CUSTOMER_LASTNAME>
   <CUSTOMER_CITY>SAV</CUSTOMER_CITY>
   <ID_TYPE>D</ID_TYPE>
   <ID_NUMBER>121321313</ID_NUMBER>
   <SS_NUM>999999999</SS_NUM>
   <WORK_PHONE_NUM>9123665659</WORK_PHONE_NUM>
   <MIDDLE_INITIAL>Ad</MIDDLE_INITIAL>
   <APT_SUITE>8320</APT_SUITE>
   <EMAIL>Andy@verifone.com</EMAIL>
   <EMP_NAME>Verifone</EMP_NAME>
   <HOME_OWNER>O</HOME_OWNER>
   <ANNUAL_INCOME>12125</ANNUAL_INCOME>
   <PHOTO_ID_STATE>GA</PHOTO_ID_STATE>
   <SOURCE_CODE>AO</SOURCE_CODE>
</XML_REQUEST>
```

OUX:

```
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <TROUTD>10586</TROUTD>
   <RESULT>Error</RESULT>
   <AUTH_CODE>Invalid Alt Source Code</AUTH_CODE>
   <REFERENCE>Invalid Alt Source Code</REFERENCE>
   <INTRN_SEQ_NUM>10586</INTRN_SEQ_NUM>
</XML_REQUEST>
```

Counter Offer (P5)

Counter Offers are always generated as a result of an Offer response on a Credit Application (P3) transaction.

Approved
Credit Application that generated OFFER:
INX:

```
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>P3</COMMAND>
   <PROCESSOR_ID>CITI</PROCESSOR_ID>
   <MERCH_NUM>512771092345</MERCH_NUM>
   <MANUAL_FLAG>0</MANUAL_FLAG>
   <TRANS_AMOUNT>4000.00</TRANS_AMOUNT>
   <ZIP_CODE>75038</ZIP_CODE>
   <STREET>4210 MADERA</STREET>
   <STATE>TX</STATE>
   <PHONE_NUM>5043646469</PHONE_NUM>
   <DOB>10101951</DOB>
   <CUSTOMER_FIRSTNAME>CYNTHIA </CUSTOMER_FIRSTNAME>
   <CUSTOMER_LASTNAME>ROBINSON </CUSTOMER_LASTNAME>
   <CUSTOMER_CITY>IRVING</CUSTOMER_CITY>
   <ID_TYPE>D</ID_TYPE>
   <ID_NUMBER>121321313</ID_NUMBER>
   <SS_NUM>298580650</SS_NUM>
   <WORK_PHONE_NUM>5043665659</WORK_PHONE_NUM>
   <EMAIL>Oracle@verifone.com</EMAIL>
   <EMP_NAME>Verifone</EMP_NAME>
   <HOME_OWNER>O</HOME_OWNER>
   <ANNUAL_INCOME>12125</ANNUAL_INCOME>
   <PHOTO_ID_STATE>TX</PHOTO_ID_STATE>
   <SOURCE_CODE>AP</SOURCE_CODE>
</XML_REQUEST>
```

OUX:

```
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <TROUTD>10588</TROUTD>
   <RESULT>OFFER</RESULT>
   <AUTH_CODE>Offered Amount $1000.00. Does customer accept the
offer?</AUTH_CODE>
   <TRANS_DATE>072208</TRANS_DATE>
   <INTRN_SEQ_NUM>10588</INTRN_SEQ_NUM>
   <TIM>161918</TIM>
   <CMRCL_TYPE>0</CMRCL_TYPE>
   <PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
   <CARD_ID_CODE>@</CARD_ID_CODE>
   <ACCT_DATA_SRC>T</ACCT_DATA_SRC>
   <CREDIT_LIMIT>1000.00</CREDIT_LIMIT>
   <PENDING_NUMBER>072208ZG04552</PENDING_NUMBER>
   <CORRELATION_UID>0820400DOE</CORRELATION_UID>
</XML_REQUEST>
```

Counter Offer Submission (Acceptance of Counter):
INX:

```
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>P5</COMMAND>
   <TROUTD>10588</TROUTD>
   <PROCESSOR_ID>CITI</PROCESSOR_ID>
   <MERCH_NUM>512771092345</MERCH_NUM>
   <MANUAL_FLAG>0</MANUAL_FLAG>
   <PENDING_NUMBER>072208ZG04552</PENDING_NUMBER>
   <CORRELATION_UID>0820400DOE</CORRELATION_UID>
   <REPLY_FLAG>Y</REPLY_FLAG>
</XML_REQUEST>
```

OUX:

```
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <TROUTD>10588</TROUTD>
   <RESULT>APPROVAL</RESULT>
   <TRANS_DATE>072208</TRANS_DATE>
   <INTRN_SEQ_NUM>10589</INTRN_SEQ_NUM>
   <TIM>162225</TIM>
   <CMRCL_TYPE>0</CMRCL_TYPE>
   <PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
   <CARD_ID_CODE>@</CARD_ID_CODE>
   <ACCT_DATA_SRC>T</ACCT_DATA_SRC>
   <ACCT_NUM>6035511112200672</ACCT_NUM>
   <CREDIT_LIMIT>1000.00</CREDIT_LIMIT>
</XML_REQUEST>
```

Account Lookup (P4)

Account Lookup by Account Number

INX:

```
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>P4</COMMAND>
   <PROCESSOR_ID>CITI</PROCESSOR_ID>
   <MERCH_NUM>512771092345</MERCH_NUM>
   <MANUAL_FLAG>0</MANUAL_FLAG>
   <ACCT_NUM>6035511110201185</ACCT_NUM>
</XML_REQUEST>
```

OUX:

```
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <TROUTD>10602</TROUTD>
   <RESULT>CAPTURED</RESULT>
   <AUTH_CODE>The available credit displayed should be considered as a
   guideline only.</AUTH_CODE>
   <REFERENCE>Actual available credit could be different at the time of
   sale</REFERENCE>
   <TRANS_DATE>072408</TRANS_DATE>
   <INTRN_SEQ_NUM>10602</INTRN_SEQ_NUM>
   <TIM>113005</TIM>
```

```
   <CMRCL_TYPE>0</CMRCL_TYPE>
   <PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
   <CARD_ID_CODE>@</CARD_ID_CODE>
   <ACCT_DATA_SRC>T</ACCT_DATA_SRC>
   <ACCT_NUM>6035511110201185</ACCT_NUM>
   <CREDIT_LIMIT>500.00</CREDIT_LIMIT>
</XML_REQUEST>
```

Account Lookup by Social Security Number

INX:

```
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>P4</COMMAND>
   <PROCESSOR_ID>CITI</PROCESSOR_ID>
   <MERCH_NUM>512771092345</MERCH_NUM>
   <MANUAL_FLAG>0</MANUAL_FLAG>
   <SS_NUM>426124765</SS_NUM>
   <DOB>03271980</DOB>
</XML_REQUEST>
```

OUX:

```
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <TROUTD>10603</TROUTD>
   <RESULT>CAPTURED</RESULT>
   <AUTH_CODE>The available credit displayed should be considered as a
   guideline only.</AUTH_CODE>
   <REFERENCE>Actual available credit could be different at the time of
   sale</REFERENCE>
   <TRANS_DATE>072408</TRANS_DATE>
   <INTRN_SEQ_NUM>10603</INTRN_SEQ_NUM>
   <TIM>113106</TIM>
   <CMRCL_TYPE>0</CMRCL_TYPE>
   <PURCH_CARD_TYPE>0</PURCH_CARD_TYPE>
   <CARD_ID_CODE>@</CARD_ID_CODE>
   <ACCT_DATA_SRC>T</ACCT_DATA_SRC>
   <ACCT_NUM>6035511110201193</ACCT_NUM>
   <CREDIT_LIMIT>500.00</CREDIT_LIMIT>
</XML_REQUEST>
```

Canadian (Interac) Debit Transactions

Overview

Starting with **PC**Charge 5.7, the integration of Canadian (Interac) debit transactions is supported when using Global Payments East (NDC) as the processing company and the VeriFone SC5000 PINpad. Canadian debit transactions are also supported when using the processing company Chase Paymentech (GSAR) and the VeriFone SC5000 PINpad. Specifically, the use of these processors and PINpad allows merchants to process transactions for customers using Interac debit cards. Currently, **PC**Charge only supports a dial-up connection to Global Payments East (NDC). For the processor Chase Paymentech (GSAR), TCP/IP and dial-up connectivity are supported. Also, the programming language used by the integrator must support ActiveX OCX controls in order to support Canadian Debit transactions.

Note: The integration of Canadian debit transactions is more complicated than the integration of U.S. debit transactions. Prior to coding, it is highly recommended that integrators review this section, review the various API layouts that are referenced in this section, and then contact VeriFone, Inc.'s Development Support department to discuss the integration.

Note: If the processor Global Payments East (NDC) is used, once the integration is completed, the integration must be certified by Global Payments East (NDC). If the processor Chase Paymentech (GSAR) is used, no certification may be required. Contact VeriFone, Inc.'s Development Support department for more details.

Note: Canadian Debit is only supported when using the XML message format.

Definitions

MAC - Message Authentication Code – Canadian banks require "MACing" when performing debit transactions. MACing provides security when processing transactions because it ensures that the messages used to communicate to the PINpad and to the processor are authentic. (**Note:** This acronym is commonly pronounced as "mack".)

MAC Block – String data generated by the PINpad and the processor based on the transaction data submitted.

Interac Association – Interac is a Canadian organization responsible for the development of Canada's national debit service. This service is known as "Interac Direct Payment" (IDP). IDP was rolled out across Canada in 1994.

Interac request string - The Interac request string is a string of text that is sent from the integrated application to the PINpad—the transaction-specific data contained in this string is used by the PINpad to prompt the customer to verify the amount of the transaction, choose which back account will be used, enter their PIN, and optionally, specify a tip amount. Once this information has been entered by the customer, the encrypted PIN, MAC Block, and other data is returned to the integrated application in an Interac response string.

OMC (Out MaCing) **and IMC** (In MaCing) **files** – Files that **PC**Charge creates in the background to perform MACing and data validation.

ReMACing – ReMACing is the process of sending the MAC data portion of the Interac request string to the PINpad to retrieve a new MAC Block from the PINpad. During the ReMACing process, the customer is not prompted to enter any data on the PINpad. ReMACing will need to performed if the transaction

details change during the process of building the debit transaction. Specifically, ReMACing is necessary when adding a tip to a transaction or when the bank account type changes during the transaction.

Integration Notes

- Prior to sending the first Canadian debit transaction with the Verifone SC5000 PINpad, a key change must occur. A key change is performed from the PINpad setup menu in the **PC**Charge GUI. Choose **Verifone SC5000** from the list and the appropriate radio button, either Global Payments East (NDC) or Chase Paymentech (GSAR), modify the Com port settings if necessary, and then click the **Key Change** button. **PC**Charge will connect via modem to Global Payments East (NDC) or Chase Paymentech and perform the key change. A message will be returned indicating whether or not the key change operation was successful.
- An error code 63, returned from Global Payments East (NDC) when processing transactions, indicates that a key change request must be performed. This error can occur even after the initial key change request has been performed.
- Canadian debit transactions must be coded to process in a single threaded manner for each PINpad used. For example, for each PINpad, the first transaction must fully complete prior to sending the second transaction.
- Only two transaction types are supported when processing Canadian debit transactions with Global Payments East (NDC). These transaction types are Purchase (action code 41) and Refund (action code 42). With Chase Paymentech (GSAR), Sale (action code 41), Return (action code 42), Online Void Sale (action code D1), Online Void Return (action code D2), and Current Key Request/Key Change Request (action code M1) are supported.
- When testing (or running live) Canadian debit transactions, Interac debit cards must be used. Standard U.S. credit cards or debit cards will not work. The track II information is encoded differently on Interac debit cards than it is on U.S. credit or debit cards. Specifically, Interac debit cards have a character encoded on track II that indicates the language code of the card (English or French), U.S. cards do not.
- The option **Allow Duplicate Transactions** in the NDC Debit Setup extended screen must be checked if the same card and amount will be sent to Global Payments East (NDC) more than once. Duplicate transactions are common in testing. If this option is not set and a duplicate transaction is sent to the processor, the response "`AP DUP`" will be returned.
- After a transaction has been successfully processed by Global Payments East (NDC), the `PinSC550.Initialize` and `PinSC550.GetSerialBlock` methods must be called, in that sequence, to reset the PINpad prior to calling the `PinSC550.StartMSR` method. If these methods are not called in this sequence, the PINpad's chip serial number will not be passed properly to Global Payments East (NDC). This will cause transactions to decline.
- When using the processor Global Payments East (NDC), it is highly recommended that the integrator set the option to automatically process OMC files. To activate this option, first, set the `PinSC550.ServerPath` property to the **PC**Charge directory, and then set the `PinSC550.AutoProcess` property to `TRUE`. If the integrator chooses to manually process the data in OMC and IMC files on their own (not recommended), the `PinSC550` class provides three methods to facilitate manual processing. `Base64Decode` and `Base64Encode` allow integrators to encode and decode the Base 64 data in the OMC files. Encoding and decoding is required because the XML message format does not support the non-standard characters that appear in these files. `InteracAnalysis` allows integrators to send the string in the OMC file (after it is decoded) to the PINpad for verification. Once the string is verified by the PINpad, the merchant is assured that the transaction message returned by the processor is genuine. Again, these three methods will not be used if the OMC files will be processed automatically by the `PinSC550` class.
- Global Payments East (NDC) requires the use of a POS Sequence Number for each PINpad used to process transactions. The POS Sequence Number for each PINpad is stored by Global Payments East (NDC) and passed to **PC**Charge with each transaction response. Because of this,

when using a new PINpad, the first transaction should be processed with a blank POS Sequence number—this transaction will decline and return a result of "NOT CAPTURED" and reference of "Invalid Seq. Num". This error indicates that Global Payments East (NDC) has passed the current POS Sequence Number to **PC**Charge, and that **PC**Charge has updated a file located in the **PC**Charge directory named NDCDebitMsg.txt with the correct sequence number. At this point, future transactions with this PINpad will be able to retrieve the correct POS Sequence Number and process successfully.

- The GetPOSSequenceNumber method is provided in the OCX, DLL, and OLE/COM methods of integration to retrieve the POS Sequence Number from a text file. This text file is named NDCDebitMsg.txt and appears in the **PC**Charge directory. The POS Sequence Number is unique to each PINpad and is required when processing transactions. If the File Method or TCP Interface will be used to process transactions, the integrator must manually retrieve the POS Sequence Number from this text file. The POS Sequence Number is stored in the file in the format: <ChipSN><POSSeqNum>. For example, "658A3P777" would indicate the Chip Serial Number of "658A3P" and the POS Sequence Number of "777". **Note:** If multiple PINpads are used, this file will contain multiple Chip Serial Number / POS Sequence Number entries.
- When processing transactions that include tips, Global Payments East (NDC) only processes the total amount of the transaction (the total amount is defined as the amount of the sale plus the tip). However, **PC**Charge requires that two properties, Amount and Gratuity, be populated when submitting transactions that include tips. Once these two properties are set, their values will be added together automatically by **PC**Charge when the transaction is submitted to Global Payments East (NDC).
- The PinSC550 class has a communications monitor that can be used by the integrator to troubleshoot integration and processing issues. To activate the communication monitor, set the CommVisible property to TRUE.
- The OLE/COM method of integration may also be used to integrate Canadian Debit transactions. When using the OLE/COM method, the class must be declared asynchronously in order to activate events. For example, use "Dim WithEvents" in the declaration section. When calling the .Send method in the PCCDebit class, pass the "TRUE" parameter to execute the transaction asynchronously. This is only supported with Global Payments East (NDC).

Integration

Tools

The DevKit includes a control named SC550.OCX and the SC5X.OCX. This control contains two classes that are used to communicate to the Verifone SC5000 PINpad and to build and parse Interac request strings:

- **SC550.PinSC550** – This class provides properties and methods that allow client applications to communicate with the Verifone SC5000 PINpad. Tables with the descriptions of the properties and methods that are available in the SC550.PinSC550 class can be found starting on page 247. Only used for the processor Global Payments East (NDC).

- **SC550.clsInteracReq** – This class provides properties and methods to build and parse the Interac strings that are required to communicate to the PINpad. Tables with the descriptions of the properties and methods that are available in the SC550.clsInteracReq class can be found starting on page 253. Only used for the processor Global Payments East (NDC).

- **SC5X.OCX –** This OCX provides the properties and methods to communicate with the SC5000 pinpad. Only used for the processor Chase Paymentech (GSAR).

The DevKit also includes three tools that are also used to process Canadian debit card transactions with **PC**Charge:

- `Debit.OCX` (OCX Method)
- `Debit` Class (DLL Method)
- `PCCDebit` Class (OLE/COM Method)

Depending on the integration method, one of the above tools should be used to enable the client application to communicate to **PC**Charge. Refer to tables found in the applicable sections in **CHAPTER 6 -- PCCharge Integration Methods** (see page 157) for descriptions of the properties and methods used while integrating.

Process Flow when using the processor Global Payments East (NDC)

The following explains the steps that are required to integrate Canadian Debit transactions. There are three general categories:

1. **Start Up** – The steps outlined in **Start Up** should typically be performed each time the integrated application is started.

2. **Transaction Processing** – The steps outlined in **Transaction Processing** should typically be performed each time a debit card transaction is processed.

3. **Shut Down** – The steps outline in **Shut Down** should typically be performed when the integrated application shuts down.

Note: In the steps below, when referencing the various properties and methods in the `Debit.OCX`, the `Debit` class in `PSCharge.dll`, and the `PCCDebit` OLE class, all three of these tools will be referred to as "the debit control" for simplicity.

Start Up – When the integrated application starts up, the following steps should be performed:

1. **Set the initial properties.**
   In the `PinSC550` class, set the initial communication properties for the PINpad. These properties are marked with a ° in the **SC550.PinSC550 Class Properties** table.

2. **Open the port.**
   Use the `OpenPort` Method in the `PinSC550` class to open the Com port that the PINpad is connected to.

3. **Initialize the PINpad.**
   Use the `Initialize` Method in the `PinSC550` class to initialize the PINpad.

4. **Retrieve the PINpad's Chip Serial Number.**
   Use the `GetSerialBlock` Method in the `PinSC550` class to retrieve the PINpad's Chip Serial Number. Once the serial number has been retrieved from the PINpad, the `PinSC550` class will fire the `ActionUpdate` event and the `GPSPinPadAction` will be set to "`ENUM_ACTION_REQ_SERIAL`" (5). The `ChipSN` property in the `PinSC550` class will contain the PINpad's Chip Serial Number. ***Store the Chip Serial Number value as a variable***—it will need to be passed to the debit control.

5. **Activate the automatic processing of OMC files (recommended).**
   To active the automatic processing of OMC files, set the properties marked with a °° in the **SC550.PinSC550 Class Properties** table. The values that should be set are noted below:
   - Set the `ServerPath` property of the `PinSC550` class to the **PC**Charge directory
   - Set the `AutoProcess` property of the `PinSC550` class to `TRUE` (default is `FALSE`)
   - Set the `AutoInterval` property of the `PinSC550` class. This property determines often the class will poll for the OMC files. Default is "`1000`" (milliseconds).

Transaction Processing – Once all of the **Start Up** steps have been completed, perform the following steps to process transactions:

1. **Indicate to the PINpad to prompt the customer to swipe their debit card.**
   Use the `StartMSR` Method in the `PinSC550` class to prompt the customer to swipe their debit card. The message "`SWIPE CARD`" / "`GLISSER CARTE`" will appear on the PINpad's screen.

2. **Wait for the customer to swipe the card. After it is swiped, retrieve and parse the track II data and retrieve the language code.**
   Once the card has been swiped in the PINpad, the `PinSC550` class will fire the `ActionUpdate` event and the `GPSPinPadAction` will be set to "`ENUM_ACTION_MSR_RECEIVED_DATA`" (7). The `TrackII` property of the `PinSC550` class will now contain the track II string from the card. Parse this string to retrieve the card number and expiration date. ***Store the track II data, card number, and expiration date values as variables***—they will need to be passed to the debit control. Also, the `LanguageCode` property of the `PinSC550` class will contain the language code that was retrieved from the track II data. ***Store the language code value as a variable***—it will need to be passed back to the `PinSC550` class when calling the `RequestInterac` method.

3. **Retrieve the POS Sequence Number from PCCharge.**
   PCCharge stores the POS Sequence Number for each PINpad in a file in the **PC**Charge directory named `NDCDebitMsg.txt`. In order to retrieve this number, set the debit control's `Path` property to the **PC**Charge directory and then call the `GetPOSSequenceNumber` method in the debit control. **Note:** The PINpad's Chip Serial Number must be passed as a parameter when calling the `GetPOSSequenceNumber` method. ***Store the POS Sequence Number value as a variable***—it will need to be passed back to the `clsInteracReq` class when calling the `BuildInteracRequest` method.

4. **Build the Interac request string.**
   To build the Interac request string, instantiate the `clsInteracReq` class, populate the required properties, and then call the `BuildInteracRequest` method in the `clsInteracReq` class. The properties that are required when building the string are marked with a ◦ in the **SC550.clsInteracReq Class Properties** table. The method will return the Interac request string. ***Store the Interac request string as a variable***—it will need to be passed back to the `PinSC550` class when calling the `RequestInterac` method. **IMPORTANT:** Do not clear or change any of the properties in the `clsInteracReq` class or destroy the object at this point. If ReMACing is required when processing this transaction, many of the properties that have already been set will be re-used.

5. **Send the Interact request string (from step 4) and the card's language code (from step 2) to the PINpad. This will instruct the PINpad to prompt the customer to confirm the transaction amount, enter a tip amount (optional), choose the bank account type (Chequing or Savings), and enter their PIN.**

Use the RequestInterac method in the PinSC550 class to send the Interac request string to the PINpad. This will instruct the PINpad to prompt for the customer for various information. The RequestInterac method requires two parameters: the Interact request string from **step 4** and the language code from **step 2**. Once the RequestInterac method is executed, the PINpad will prompt the customer to 1) confirm the transaction amount; 2) specify the tip amount (optional); 3) choose their bank account type; and 4) enter their PIN.

6. **Wait for the transaction-specific data to be entered by the customer on the PINpad. When the PINpad indicates the data has been entered, parse the data.**
   Once data entry by customer has completed on the PINpad, the PinSC550 class will fire the ActionUpdate event and the GPSPinPadAction will be set to "ENUM_ACTION_INTERAC_RECEIVED_DATA" (16). The Interac response string returned by the PINpad will be placed in the DeviceData property of the PinSC550 class. To parse the string, pass it to the ParseResponseData method in the clsInteracReq class. When the ParseResponseData method returns TRUE, the properties marked with a ◦◦ in the **SC550.clsInteracReq Class Properties** table will be populated with the transaction-specific data. ***Store the value returned in TipAmount as a variable***—it may need to be passed to the debit control. ***Also, store the values returned in MacBlock and PinBlock as variables***—they will need to be passed to the debit control.

7. **Determine if ReMACing must occur. If ReMACing is not necessary, skip to step 9.**
   To determine if ReMACing must occur, check the value of the Boolean property RequireReMac in the PinSC550 class. If this property is set to TRUE, then ReMACing is required. Otherwise, ReMACing is not required, skip to step 9.

8. **Request a new MAC block from the PINpad.**
   If RequireReMac is set to TRUE, a new MAC block must be retrieved from the PINpad. The ReMacData property will contain a string of data that must be sent to the PINpad to retrieve a new MAC block. To perform the ReMAC, pass the string in the ReMacData property to the PINpad by using the RequestMAC method in the PinSC550 class. The RequestMAC method requires one parameter, the string of MAC data, to be passed to it. Once the PINpad creates and returns the new MAC block, the MACBlock property of the PinSC550 class will contain the new MAC block. ***Store the value returned in the MACBlock property as a variable***—it will need to be passed to the debit control.

9. **Process the transaction.**
   To process the transaction, set the path to the **PC**Charge directory in the debit control, check if SYS.PCC exists, and then populate the required properties in the debit control with the variables that have been stored by the application. The properties required to process a transaction are marked with a ◦◦ in the various debit control tables in **CHAPTER 6 -- PCCharge Integration Methods** (see page 157). Once the properties are populated, call the Send method. The Send method will instruct the debit control to send the transaction to **PC**Charge. **PC**Charge will then dial out to Global Payments East (NDC) and submit the transaction request. Once the transaction is processed by Global Payments East (NDC), a response is returned to **PC**Charge and the POS Sequence Number is updated. If the option to automatically process the OMC files is selected, **PC**Charge will validate the response message automatically, and then return the status of the transaction via the debit control to the integrated application. The most important information is returned in the GetResult, GetAuth, and GetRefNumber methods. Once all response data has been retrieved from the debit control, call the Clear or ClearVariables method to reset all of the properties. **Note:** If the OMC files will not be processed automatically, the integrator must poll for and then process the data returned in the OMC files to validate the transaction.

10. **Prepare the PINpad for the next transaction.**
    Once the transaction has completed, the PINpad's screen will display to the customer whether or not the transaction was approved. After a few moments, the message "`OBTAIN CARD`" will appear on the PINpad's screen. In order to prepare the PINpad for the next transaction, call the `Initialize` Method in the `PinSC550` class. The PINpad's screen will display the message "`WELCOME/BONJOUR`". To process the next transaction, return to step 1 of "Transaction Processing".

Shut Down – When the integrated application has finished processing transactions, the following step should be followed:

1. **Shut down the Port.**
   In the `PinSC550` class, shut down the port by calling the `ClosePort` method in the `PinSC550` class.

Note: All steps listed above are only applicable when using the processor Global Payments East (NDC).

Process Flow when using the processor Chase Paymentech (GSAR)

1. **Initialize the VeriFone SC5000 PINpad.**

   Using the SC5X.OCX, initialize the VeriFone SC5000 PINpad by setting the required properties (see p. 193 for SC5X.OCX properties). The RetrieveCreditSwipe method can be called to set the PINpad to a ready state and will prompt for the swipe. Once the swipe occurs, if successful, the .Card, .Member, .ExpDate, and .Track properties will be populated automatically, and the PINpad will prompt for the PIN number.

2. **Send the transaction to PCCharge.**

   Send the transaction to PCCharge by calling the `.Send 3` method of the SC5X.OCX. The transaction will be sent to the Path specified in the `.ServerPath` property of the SC5X.OCX.

3. **Retrieve the response.**

   Once **PC**Charge has processed the transaction request, call the various `.Get` methods to retrieve the response. Once the response has been retrieved, parse the results to determine the outcome of the transaction (see p. 194 for SC5X.OCX methods).

Transaction Inquiry

Overview

The **PC**Charge integration methods provide the ability to submit Transaction Inquiry requests. A Transaction Inquiry request (action code `ZI`) retrieves transaction data stored in the **PC**Charge database (`pccw.mdb`) based on the TroutD or account number that is submitted. The response provided by **PC**Charge will include all available transaction records returned in XML tags that correspond to the appropriate database fields. All five integration methods support the Transaction Inquiry request.

Note: Sensitive information such as credit card numbers are encrypted in the **PC**Charge database. The transaction inquiry command will not return credit card numbers in plain text. If credit card numbers from past transactions are needed, these numbers must be stored in the integrated application. See the sections **Important Security Notice** (see page 9) and **Warnings, Tips, and Guidelines** (see page 55) for more information on the storage of sensitive cardholder data and the regulations related to data storage.

Usage

OCX Method and DLL Method

To submit a Transaction Inquiry request via the OCX or DLL Methods, use the `Charge.OCX` control or the `Charge` class of `PSCharge.dll`. Populate the following properties:

- `User` – A valid user name in **PC**Charge
- `Path` – The path to the **PC**Charge directory
- `Command` – "`ZI`"
- `TroutD` or `Card` – Populate only one of these properties. The request will return the results based on the value passed in either one of these properties.

Once the properties are populated, call the `Send` method. Once the request is processed, the response can be retrieved by calling the `GetXMLResponse` method. This method returns the contents of the `.oux` file that is returned by **PC**Charge.

OLE/COM Method

To submit a Transaction Inquiry request via the OLE/COM Method, use the `debit` class (the `charge` class cannot be used because `charge`'s `action` property is defined as long and this property will not accept the "`ZI`" action code. Populate the following properties in the `debit` class:

- `User` – A valid user name in **PC**Charge
- `Path` – The path to the **PC**Charge directory
- `Action` – "`ZI`"
- `TroutD` or `Card` – Populate only one of these properties. The request will return the results based on the value passed in either one of these properties.

Once the properties are populated, call the Send method. Once the request is processed, the response can be retrieved by calling the GetXMLResponse method. This method returns the contents of the .oux file that is returned by **PC**Charge.

File Method / TCP Interface

To submit a Transaction Inquiry request via the File Method or TCP Interface, pass a request to **PC**Charge that contains the following tags:

- USER_ID - A valid user name in **PC**Charge
- COMMAND - ZI
- TROUTD or ACCT_NUM - Use only one of these tags. The request will return the results based on the value passed in either one of these tags.

Once the request is processed, **PC**Charge returns the response in an .oux file.

General Note: The RECORD_COUNT tag in the XML response indicates how many transaction records are returned for each account number or TroutD value. Multiple transaction records will be returned (as nested XML tags) if this is the case. For example, if a Pre-Authorization was completed with a Post-Authorization, and then Voided, RECORD_COUNT would indicate "3" and three transaction records would be returned.

Example

This example demonstrates performing a Transaction Inquiry request using the File Method or TCP Interface. Refer to the section **File Method** (see page 481) for more information on the File Method or TCP Interface API.

Request:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<COMMAND>ZI</COMMAND>
		<TROUTD>1003</TROUTD>
	</XML_REQUEST>
</XML_FILE>
```

Response:

```
<TRANS_INQUIRY>
	<RECORD_COUNT>1</RECORD_COUNT>
	<TRANS_RECORD>
		<TABLE>Trans</TABLE>
		<Number>1003</Number>
		<Ticket>123456789</Ticket>
		<Date>5/6/2004</Date>
		<Time>10:32:25</Time>
		<Station>User1</Station>
		<Processor>NOVA</Processor>
		<TID>99988836</TID>
		<Issuer>Visa</Issuer>
		<Member>VERIFONE TEST 3</Member>
		<ExpDate>1208</ExpDate>
		<Action>1</Action>
		<Manual>False</Manual>
		<Amount>$1.00</Amount>
		<Ref>00000000</Ref>
		<Result>CAPTURED</Result>
		<Auth>TESTVI</Auth>
		<Result_Ref>035</Result_Ref>
		<Tax_Amount>$0.00</Tax_Amount>
		<Total_Auth>$1.00</Total_Auth>
		<Trans_Indicator>1</Trans_Indicator>
		<ReqACI>Y</ReqACI>
		<RetACI>B</RetACI>
		<TransDate>0506</TransDate>
		<TransTime>103226</TransTime>
		<Card>4012........0026</Card>
		<PeriodicPayment>False</PeriodicPayment>
		<BatchNumber>0</BatchNumber>
		<ItemNumber>35</ItemNumber>
		<CVV2_Resp>X</CVV2_Resp>
		<Selected>False</Selected>
		<Commercial_Card>N</Commercial_Card>
		<Offline>N</Offline>
		<Status>A</Status>
		<TroutD>1003</TroutD>
		<CardPresent>0</CardPresent>
		<Business_Type>0</Business_Type>
	</TRANS_RECORD>
</TRANS_INQUIRY>
```

Batch Settlement

Overview

To submit a batch settlement request via the File Method or TCP Interface, pass a request to **PC**Charge that contains the following tags:

- USER_ID - A valid user name in **PC**Charge
- COMMAND – (see below)
- MERCH_NUM – The merchant number
- PROCESSOR_ID – The processor ID

Once the request is processed, **PC**Charge returns the response in an .oux file if using the File Method interface. If using TCP, the response is returned in the data stream.

Action Code	Description
30	Batch Inquiry
31	Batch Close/Settle
32	Private Label Batch Close
33	Amex Split Settle
39	Close/Settle Batches for all merchant numbers*

* When using action code 39, an appropriate timeout value must be set to allow for closing/settling of all merchant numbers set up in **PC**Charge. If action code 39 is chosen, a default value of 1200 seconds is selected. If a longer time is needed, the timeout property must be set to an adequate value.

Simple Batch Request

Example:

1) Example of batch settlement request:

```
<XML_FILE>
      <XML_REQUEST>
            <USER_ID>User1</USER_ID>
            <COMMAND>31</COMMAND>
            <PROCESSOR_ID>GSAR</PROCESSOR_ID>
            <MERCH_NUM>999999999999519</MERCH_NUM>
      </XML_REQUEST>
</XML_FILE>
```

Example of batch settlement response:

```
<XML_FILE>
      <XML_REQUEST>
            <USER_ID>User1</USER_ID>
            <RESULT>Accepted</RESULT>
            <AUTH_CODE>026 0207 9999</AUTH_CODE>
            <REFERENCE>999999999999519</REFERENCE>
            <INTRN_SEQ_NUM>1.00</INTRN_SEQ_NUM>
            <TRANS_ID>026 0207 9999</TRANS_ID>
            <TICODE>1</TICODE>
            <RET>1</RET>
            <TIM>1</TIM>
```

```
            <BATCH_NUMBER>26</BATCH_NUMBER>
      </XML_REQUEST>
</XML_FILE>
```

Example:

2) When the batch exceeds the maximum size allowed or that is configured in PCCharge, a response file should look similar to this:

```
<XML_FILE>
            <XML_REQUEST>
                        <USER_ID>User1</USER_ID>
                        <RESULT>Accepted</RESULT>
                        <AUTH_CODE>00001</AUTH_CODE>
                        <REFERENCE>000192000903</REFERENCE>
                        <INTRN_SEQ_NUM>8.00</INTRN_SEQ_NUM>
                        <TRANS_ID>00001</TRANS_ID>
                        <TICODE>5</TICODE>
                        <RET>2</RET>
                        <TIM>1</TIM>
                        <BATCH_NUMBER>001</BATCH_NUMBER>
            </XML_REQUEST>
            <XML_REQUEST>
                        <USER_ID>User1</USER_ID>
                        <RESULT>Accepted</RESULT>
                        <AUTH_CODE>00002</AUTH_CODE>
                        <REFERENCE>000192000903</REFERENCE>
                        <INTRN_SEQ_NUM>13.00</INTRN_SEQ_NUM>
                        <TRANS_ID>00002</TRANS_ID>
                        <TICODE>5</TICODE>
                        <RET>1</RET>
                        <TIM>2</TIM>
                        <BATCH_NUMBER>002</BATCH_NUMBER>
            </XML_REQUEST>
</XML_FILE>
```

Multiple Batches

Note: In the event there are multiple batches waiting to be settled in one settlement, the integrated application will need to be designed to loop through the settlement response to retrieve the response for each batch.

Batch Inquire

Before submitting transactions for settlement, some merchant will prefer to perform a Batch Inquire. Depending on the process this will return batch information to give the merchant more information regarding the batch and the totals.

Example Batch Inquire:

```
<XML_FILE>
      <XML_REQUEST>
            <USER_ID>User1</USER_ID>
            <COMMAND>30</COMMAND>
            <PROCESSOR_ID>GSAR</PROCESSOR_ID>
            <MERCH_NUM>999999999999519</MERCH_NUM>
      </XML_REQUEST>
</XML_FILE>
```

Batch Inquire with Card Type

Beginning with PCCharge version 5.9.2 there is now the ability to perform a Batch Inquiry by Card Type. This involves sending the same type of transaction for Batch Inquire and adding one new request tag.

New Request Tag:	<CARD_TYPE>
Valid Values:	MC, VISA, AMEX, DISC, DEBIT, EBT, PL, JCB
New Response Tag:	<VISA_COUNT>, <VISA_AMOUNT>, <MC_COUNT>, <MC_AMOUNT>, <DISC_COUNT>, <DISC_AMOUNT>, <PL_COUNT>, <PL_AMOUNT>, <JCB_COUNT>, <JCB_AMOUNT>
Pre-existing XML Response Tags:	<DEBIT_COUNT>, <DEBIT_TOTAL_AMOUNT>, <EBT_COUNT>, <EBT_TOTAL_AMOUNT>, <AMEX_COUNT>, <AMEX_AMOUNT>, <TOTAL_COUNT>, <TOTAL_BALANCE>

Example Batch Inquire with Card Type

```
<XML_FILE>
      <XML_REQUEST>
            <USER_ID>User1</USER_ID>
            <COMMAND>30</COMMAND>
            <PROCESSOR_ID>GSAR</PROCESSOR_ID>
            <MERCH_NUM>999999999999519</MERCH_NUM>
            <CARD_TYPE>MC</CARD_TYPE>
      </XML_REQUEST>
</XML_FILE>
```

Amex Settlement

Action Code	Description
33	Settlement Request
35	Reverse Batch
36	Resubmit Batch
37	Get Results
38	Finalize Batch

Additional XML tag: <BATCH_NUMBER>

AMEX Split Settlement
Split settle is the ability to connect directly to American Express to settle AMEX card transactions. Both split dial and split settle will typically get lower per-transaction rates on American Express transactions. Thru integration sending a command of 33 will allow developer to send an AMEX split settlement request to PCCharge.

At the completion of settlement file transfer, a message of completion or error status will be returned to the integrator. Once the transactions have successfully been transferred to AMEX, you may request the AMEX settlement results as described in the **Getting Amex settlement section below**. Alternatively, merchants may visit AMEX's website to view their settlement results. The following information (supplied to VeriFone by AMEX) illustrates this concept:

"For questions regarding your American Express merchant account, American Express offers a free Internet service, Online Merchant Services, which supports online account management specific to payments, customer disputes and more. For information on how to enroll in Online Merchant Services, please contact your account representative or visit http://www.americanexpress.com/merchant. For merchants that are not enrolled in the online account management tool, paper statements are issued for general account maintenance. For general customer service, please call 800-528-5200."

A Credit card detail report request also is a good way to obtain those transactions that were sent to American Express, the request should specify Card Type:, Start:, and End: filters to specify AMEX and the date of the transactions to be viewed).

Getting Amex Settlement Result – This feature will allow you to view the confirmation and acknowledgement files that have been obtained from AMEX regarding the settlement. The request should include the action code 37 as well as the merchant account whose confirmation files developer wishes to retrieve. Upon Request, PCCharge contact AMEX and download any confirmation/acknowledge files for the selected merchant. Note that after settling, the acknowledgement file could take up to 30 minutes and the confirmation file could take up to 4 hours. The files will be located in the AMEXARCH folder in PCCharge Install directory.

Finalize Transactions – The request to mark AMEX transactions as settled.

Reverse – submit a reversal of an archived file.

Resubmit - To re-transmit a settlement file to American Express, simply send a request with action code 36, a batch should not be re-transmitted to American Express unless directed by a Technical Support Representative.

Example of an AMEX Split batch settlement request:

```
<XML_FILE>
      <XML_REQUEST>
            <USER_ID>User1</USER_ID>
            <COMMAND>33</COMMAND>
            <PROCESSOR_ID>VISA</PROCESSOR_ID>
            <MERCH_NUM>999999999911</MERCH_NUM>
      </XML_REQUEST>
</XML_FILE>
```

Example of an AMEX Split batch settlement response:

```
<XML_FILE>
      <XML_REQUEST>
            <USER_ID>User1</USER_ID>
            <RESULT>Accepted</RESULT>
            <AUTH_CODE>FTP transmission successful.</AUTH_CODE>
            <REFERENCE>888000000359</REFERENCE>
            <TICODE>0</TICODE>
            <RESULT_CODE>2</RESULT_CODE>
            <BATCH_NUMBER>5</BATCH_NUMBER>
            <MERCH_NUM>888000000359</MERCH_NUM>
            <PROCESSOR_ID>AMEX</PROCESSOR_ID>
      </XML_REQUEST>
</XML_FILE>
```

Note: I have included cashier tags in the examples below. For backwards compatibility, these tags are not mandatory. However, if the integrator does pass in these tags, validation will take place for both cashier name/password accuracy and permissions.

Reverse Batch (35)

Use this action code to reverse an entire batch so that it can be edited and resubmitted.

```
INX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>35</COMMAND>
   <CASHIER_NAME>Manager</CASHIER_NAME>
   <PASSWORD>Test123$</PASSWORD>
   <PROCESSOR_ID>VISA</PROCESSOR_ID>
   <MERCH_NUM>888000000359</MERCH_NUM>
   <BATCH_NUMBER>004</BATCH_NUMBER>
   <TXN_TIMEOUT>5000</TXN_TIMEOUT>
</XML_REQUEST>

OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <RESULT>CAPTURED</RESULT>
   <AUTH_CODE>Reversal File Sent Successfully</AUTH_CODE>
   <INTRN_SEQ_NUM>10596</INTRN_SEQ_NUM>
   <MERCH_NUM>888000000359</MERCH_NUM>
</XML_REQUEST>
```

Resubmit Batch (36)

Use this action code to resubmit a previously settled batch. Would use this action if Amex did not receive the previous settlement or if you have reversed and edited a batch and need to re-settle.

```
INX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>36</COMMAND>
   <CASHIER_NAME>Manager</CASHIER_NAME>
   <PASSWORD>Test123$</PASSWORD>
   <PROCESSOR_ID>VISA</PROCESSOR_ID>
   <MERCH_NUM>888000000359</MERCH_NUM>
   <BATCH_NUMBER>006</BATCH_NUMBER>
   <TXN_TIMEOUT>5000</TXN_TIMEOUT>
</XML_REQUEST>

OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <RESULT>CAPTURED</RESULT>
   <AUTH_CODE>Resubmittal File Sent Successfully</AUTH_CODE>
   <INTRN_SEQ_NUM>10597</INTRN_SEQ_NUM>
   <MERCH_NUM>888000000359</MERCH_NUM>
</XML_REQUEST>
```

Get Results (37)

Use this action code in order to retrieve any results files available on the Amex ftp site. Please note that even though the integrator submits a batch number, this transaction will pull any files available at Amex for this particular merchant setup. Therefore, this action may or may not return files for the batch number that the integrator supplied in the request. The user would still have to go to the UI to actually view the results.

```
INX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>37</COMMAND>
   <CASHIER_NAME>Manager</CASHIER_NAME>
   <PASSWORD>Test123$</PASSWORD>
   <PROCESSOR_ID>VISA</PROCESSOR_ID>
   <MERCH_NUM>888000000359</MERCH_NUM>
   <BATCH_NUMBER>004</BATCH_NUMBER>
   <TXN_TIMEOUT>5000</TXN_TIMEOUT>
</XML_REQUEST>

OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <RESULT>CAPTURED</RESULT>
   <AUTH_CODE>Files Retrieved</AUTH_CODE>
   <INTRN_SEQ_NUM>13167</INTRN_SEQ_NUM>
   <MERCH_NUM>888000000359</MERCH_NUM>
</XML_REQUEST>
```

If no files are present on the Amex site, the integrator will receive the following OUX.

```
OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <RESULT>NOT CAPTURED</RESULT>
   <AUTH_CODE>No File Present</AUTH_CODE>
   <INTRN_SEQ_NUM>13168</INTRN_SEQ_NUM>
   <MERCH_NUM>888000000359</MERCH_NUM>
</XML_REQUEST>
```

If the account is not configured for Amex direct settlement, the integrator will receive the following OUX.

```
OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <RESULT>NOT CAPTURED</RESULT>
   <AUTH_CODE>AMEX Settlement Not Set Up</AUTH_CODE>
   <INTRN_SEQ_NUM>13166</INTRN_SEQ_NUM>
   <MERCH_NUM>888000001005</MERCH_NUM>
</XML_REQUEST>
```

Finalize Batch (38)

Use this action code to update the database to mark all the transactions in the batch as “Settled”. Should only be done after the ACK and CON files have been retrieved and the merchant has verified that the batch did indeed settle without errors.

```
INX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <COMMAND>38</COMMAND>
   <CASHIER_NAME>Manager</CASHIER_NAME>
   <PASSWORD>Test123$</PASSWORD>
   <PROCESSOR_ID>VISA</PROCESSOR_ID>
   <MERCH_NUM>888000000359</MERCH_NUM>
   <BATCH_NUMBER>006</BATCH_NUMBER>
   <TXN_TIMEOUT>5000</TXN_TIMEOUT>
</XML_REQUEST>

OUX:
<XML_REQUEST>
   <USER_ID>User1</USER_ID>
   <RESULT>CAPTURED</RESULT>
   <AUTH_CODE>Transactions Finalized</AUTH_CODE>
   <INTRN_SEQ_NUM>13171</INTRN_SEQ_NUM>
   <MERCH_NUM>888000000359</MERCH_NUM>
</XML_REQUEST>
```

Gift Settlement

RBS WorldPay (LYNK)

To submit gift settlement with RBS WorldPay we send an Action “E1”, this action invokes a gift settlement routine. The settlement is processed then returns a response. The response string contains the settlement response. The response will state whether the batch is Accepted, a Batch Record Count, and the Batch Net Total.

Example:

```
INX:
<XML_REQUEST>
      <USER_ID>User1</USER_ID>
      <COMMAND>E1</COMMAND>
      <PROCESSOR_ID>LYNK</PROCESSOR_ID>
      <MERCH_NUM>000001000041</MERCH_NUM>
      <TXN_TIMEOUT>5000</TXN_TIMEOUT>
</XML_REQUEST>

OUX:
<XML_REQUEST>
      <USER_ID>User1</USER_ID>
      <TROUTD>1038</TROUTD>
      <RESULT>Accepted</RESULT>
      <INTRN_SEQ_NUM>1038</INTRN_SEQ_NUM>
      <TRANS_ID>000000003</TRANS_ID>
      <TICODE>1</TICODE>
      <RESULT_CODE>2</RESULT_CODE>
      <AUTH_AMOUNT>1.00</AUTH_AMOUNT>
      <RECORD_COUNT>3</RECORD_COUNT>
      <MERCH_NUM>000001000041</MERCH_NUM>
</XML_REQUEST>
```

Batch Totals Storage

PCCharge stores batch totals in the `BatchTotals` table of the **PC**Charge database. Developers can access this table by building database queries. The data in this table can be used to write custom reports that can assist merchants with reconciliation.

Note: This functionality is currently available only for FDMS Atlanta dial, lease-line, and TCP/IP Pass through connections. Other processors and connection types will be added in future releases.

BatchTotals Table

PCCharge stores batch totals received from the processor in the `PCCW.mdb` database. These totals are stored in the `BatchTotals` table after the completion of an inquiry or settlement. Information stored includes:

Column Name	Database Data Type	Description
`TransNum`	Text 10 characters	Transaction number from **PC**Charge
`Date`	Date/time	Date of transaction
`Time`	Text 8 characters	Time of transaction
`Action`	Text 2 characters	Action code of transaction: 30 = Inquiry 31 = Settlement
`TID`	Text 32 characters	Terminal ID number for merchant account
`GrandTotal`	Currency	Gross amount of settlement/inquiry (includes totals plus returns plus voids)
`Net`	Currency	Net amount of settlement/inquiry (includes totals minus returns minus voids)

Additional columns include the following processor dependent columns: fee amount, returns, voids, debit, EBT, check, stored value totals, and totals for each card issuer along with transaction counts. More columns will be added in the future.

The totals information can be tracked using transaction number, date, time, action code, merchant number, or any combination thereof. Queries can be linked with the settlement table.

Health Message Transaction

This is a transaction that will tell the state of PCCharge to determine if it is available accept transactions. The Action Code for this transaction is 'ZH'. No parameters other than this action and the user are needed. This function will return a string with a code that reflects PCCharge's state.

Note: PC Charge must be running for the ZH command to function.

The codes are as follow:

0 = OK response, transactions should process fine (No sys.pcc exists)
2 = Sys.pcc exists with no predefined code in it
3 = A Batch is in progress (Sys.pcc exists with a code 1 in the sys.pcc file)
4 = Database is being repaired (Sys.pcc exists with a code 2 in the sys.pcc file)
5 = Backup Zip is in progress (Sys.pcc exists with a code 3 in the sys.pcc file)
6 = Modem is being initialized (Sys.pcc exists with a code 4 in the sys.pcc file)
7 = Database is being archived (Sys.pcc exists with a code 5 in the sys.pcc file)
8 = The settlement file is locked (Sys.pcc exists with a code 6 in the sys.pcc file)
9 = A Reversal is in process
10 = A reversal is waiting (Only used for FDMS Atlanta)
11 = Reserved
12 = Reserved
13 = Reserved

For testing purposes, create a sys.pcc with the codes outlined above; verify that the response contains the correct associated result code.

Request Example:

```
<XML_REQUEST>
      <USER_ID>User1</USER_ID>
      <COMMAND>ZH</COMMAND>
</XML_REQUEST>
```

Response Example:

```
<XML_REQUEST>
      <USER_ID>User1</USER_ID>
      <RESULT>0</RESULT>
</XML_REQUEST>
```

Command Line Switches

The following table lists various command line switches that can be used to customize how **PC**Charge Pro or Payment Server runs. Simply add these switches to the end of target line of the **PC**Charge Pro or Payment Server shortcut.

Command Line Switch	Description
`/UI`	Loads the **PC**Charge Payment Server graphical user interface (GUI) on initialization. (**PC**Charge Payment Server only)
`/IOL`	Turns on the `IODebug.log` file. This file is used for troubleshooting purposes.
`/D`	Activates Demo mode. In Demo Mode, **PC**Charge will operate with all five integration methods if the XML message format is used. **Important Note**: Prior to activation, the Demo mode command line switch must be deleted to enable LIVE mode.

Command Line Switch Format

```
"C:\Program Files\PCCW\Pccw.exe" [/IOL] [/D]
"C:\Program Files\Active-Charge\Active-Charge.exe" [/UI] [/IOL] [/D]
```

Text inside [ ] is optional.

The command line switches must appear in the order specified above.

CHAPTER 5 -- DevKit Constants

DevKit Constants

Action Codes

Refer to **CHAPTER 3 -- Payment Processing Basics** (see page 41) for further descriptions of the various transactions.

Credit Card

Action Code	Description
1	Sale
2	Credit
3	Void Sale **Note:** Reversals should be processed when supported instead of Void. See page 93
4	Pre-Authorization
5	Post-Authorization
6	Void Credit
7	Void Post-Authorization
8	Commercial Card Sale
9	Commercial Card Credit
10	Commercial Card Post-Authorization
11	Pre-Authorization Return **Note:** Only available for processor **NBS**
13	Gratuity (finalizes or modifies a Restaurant Sale)
14	Sale with Gratuity
15	Book
17	Void Ship
P1	Pre-Paid Credit Card Balance Inquiry
P2	Credit Card Authorization Reversal
P3*	Credit Application – **Citi**© Private Label Only
P4*	Account Lookup – **Citi**© Private Label Only
P5*	Counter Offer – **Citi**© Private Label Only **Note:** Follows a P3 when the result is OFFER.

* These Action Codes are available as of PCCharge version 5.8. They are the result of the new Citi© Credit Plan functionality. **Note:** See **Citi Credit Plan** on page 116 for examples.

Debit

Action Code	Description
40	Pre-Authorization **Note:** Only available for processor **NBS**
41	Sale
42	Return
43	Void **Note:** Not all debit card processors in PCCharge support debit void functionality
44	Debit Card Balance Inquiry **Note:** Only available for processor **Elavon (NOVA)**
46	Void Return
47	Post-Authorization **Note:** Only available for processor **NBS**
48	Sale Recovery
49	Return Recovery
M1	Key Change Request (Canadian Debit only)
D1	Online Void Sale (Canadian Debit via Chase Paymentech GSAR only)
D2	Online Void Return (Canadian Debit via Chase Paymentech GSAR only)
D3	Debit Reversal
S21°°	Gratuity (Canadian Debit via Chase Paymentech GSAR only)

°°The SC5X.OCX supports gratuity with the S21 action code. Gratuity is input by the user into the VeriFone SC5000 exclusively before the transaction is sent to **PC**Charge. Gratuity adjustments after the transaction are not available.

Check Verification

Action Code	Description
20	MICR (verify)
21	COD / Phone (verify)
22	Driver's License (verify)
23	Double ID (verify)
50	MICR (verify in truncation mode)
54	Manager Override

Check Conversion

Action Code	Description
24	Void Check
51	Sale (MICR)
52	Void (MICR)
53	Force (MICR)
59	Settle (MICR)

Check Conversion (Telecheck Only)

Action Code	Transaction	Description	Properties
51	ECA Sale	**PC**Charge sends the ECA request and automatically handles the status request	**Required:** .User .Path .Processor .MerchantNumber .Amount .MICRStatus .CHECK_READER_CODE .CheckType .MICR_DATA or .Check_Number .Account_Number .Transit_Number **Conditional** .DL_TRACK_II or .Drivers_License .State .Phone_Number **Optional:** .CustomerCity .CustomerStreet .Zip_Code .CustomerName .Birth_Date .Ticket
52	ECA Void	**PC**Charge sends the void request to TeleCheck	**Required:** .User .Path .TroutD **Optional:** .Ticket
56	ECA Adjustment	**PC**Charge sends the Adjustment request to TeleCheck	**Required:** .User .Path .TroutD .Amount **Optional:** .Ticket
C4 C4 (continued)	ECA Authorization	**PC**Charge sends only the ECA Auth request and waits for the POS to send either a C5 or C6 prior to sending the status message	**Required:** .User .Path .Processor .MerchantNumber .Amount .MICRStatus .CHECK_READER_CODE .CheckType .MICR_DATA or .Check_Number .Account_Number .Transit_Number **Conditional** .DL_TRACK_II or .Drivers_License .State .Phone_Number **Optional:** .CustomerCity .CustomerStreet .Zip_Code

			.CustomerName .Birth_Date .Ticket
C5	ECA Completion Accepted	**PC**Charge sends the status message indicating the ECA has been Accepted	**Required:** .User .Path .TroutD **Optional:** .Ticket
C6	ECA Completion Refused	**PC**Charge sends the status message indicating the ECA as been Refused. The merchant may still hold the check for paper deposit.	**Required:** .User .Path .TroutD **Optional:** .Ticket

EBT

Action Code	Description
60	Account Inquiry (GSAR and TSYS: Cash Benefits / Food Stamp Balance Inquiry)
61	Cash Withdrawal
62	Food Stamp Purchase (GSAR: Food Stamp Sale)
63	Food Stamp Credit (Return) (GSAR: Food Stamp Return)
64	Cash Post-Authorization (GSAR: Cash Benefits Prior Auth Sale)
65	Food Stamp Post-Authorization (Electronic Voucher) (GSAR: Food Stamp Prior Auth Sale)
66	Food Stamp Credit Post-Authorization
67	Cash Void
68	Food Stamp Void
69	Food Stamp Credit Void
70	Purchase or Purchase w/Cashback (GSAR: Cash Benefits Sale with or w/o Cash Back)
71	Foodstamp Purchase Recovery
72	Foodstamp Credit Recovery
73	Foodstamp Post-Auth Recovery
74	Cash Withdrawal Recovery
75	Cash Purchase Recovery

Gift

Note: Refer to the section **Gift Card Transactions** (see page 110) for a list of Gift Card processors and the action codes they support.

Action Code	Description
18	Balance Inquiry
25	Redemption / Sale
26	Register / Replace
27	Add Value / Add Points / Prior Issuance / Increment / Reload
28	Activation / Activate
29	Void / Cancel
0A	Deactivate / Close
0B	Refund / Credit
0C	Totals Inquiry / Current Day Totals
0D	Previous Day Totals
0H	Balance with Lock
0I	Post-Authorization / Redemption Unlock
0N	Redeem Points
0P	Balance Merge
0Q	Balance Adjustment / Unload
0R	Balance Transfer
0S	Report Lost / Stolen
0T	Cash Out / Cash Back
0U	Add Tip
0V	Pre-Auth

Batch

Action Code	Description
30	Batch Inquiry
31	Batch Close/Settle
32	Private Label Batch Close
33	Amex Split Settle
35	Reversing Batches for Amex Split Settle
36	Resubmitting Batches for Amex Split Settle
37	Getting Result Files for Amex Split Settle
39	Close/Settle Batches for all merchant numbers*
E1	LYNK Gift Settlement

* When using action code 39, an appropriate timeout value must be set to allow for closing/settling of all merchant numbers set up in **PC**Charge. If action code 39 is chosen, a default value of 1200 seconds is selected. If a longer time is needed, the timeout property must be set to an adequate value.

Report

Action Code	Description
81	Credit Card Detail report
82	Batch Pre-Settle report
83	Batch Post-Settle report
84	Check Summary report

*Action code 80 is no longer a supported action code.

Miscellaneous

Action Code	Description
90	Reinitialize Modem
99	Pre-Dial
ZA	Transaction Archive
ZC	Database Archive Configuration
ZI	Transaction Inquiry
ZH	Health Message
Zm	Minimize **PC**Charge window
ZM	Maximize **PC**Charge window
ZS	Shut down **PC**Charge
ZD	Delete Cashier
ZL	Logout
ZP	Change Password
ZR	Reenable Cashier
ZU	Add Cashier

Address Verification Response Codes

Response Code	Address Match
A	Address matches, ZIP code does not
B	Address matches, postal code does not
C	No match on address or postal code
D	Street address and postal code matches
E	AVS error
G	Service not supported by non-US issuer
I	Address not verified for international transaction
M	Street address and postal code matches
N	No match on address or ZIP code
P	Postal code matches, address does not
R	Retry, system is unavailable or timed out
S	Service not supported by issuer (card type does not support AVS)
U	Address information is unavailable
W	9-digit ZIP code matches, address does not
X	Exact match
Y	Address and 5-digit ZIP code match
Z	5-digit ZIP code matches, address does not
0	No response sent

CVV2/CVC2/CID Response Codes

Response Code	Address Match
M	CVV2/CVC2/CID match
N	CVV2/CVC2/CID mismatch
P	Not processed -- Either the CVV2/CVC2/CID was not provided, or the card does not have a CVV2/CVC2/CID value. If the CVV2/CVC2/CID was left blank, resubmit as a zero dollar amount for the transaction so the customer's credit line won't be affected by the second CVV2/CVC2/CID request.
S	Issuer indicates that the CVV2/CVC2/CID data should be present on the card, but the merchant has indicated that the CVV2/CVC2/CID data is not present on the card.
U	Issuer has not certified for CVV2/CVC2/CID or issuer has not provided Visa/MasterCard with the CVV2/CVC2/CID encryption keys.

Credit Card Types

Credit Card Issuer	Credit Card Type
American Express	"AMEX"
Carte Blanche	"CBLN"
Diner's Club	"DCCB"
Discover	"DISC"
Enroute	"ENRT"
Fleet One	"FLT1"
Fuelman	"FUEL"
JAL	"JAL "
JCB	"JCB "
MasterCard	"MC "
Wright Express	"WEX "
Visa	"VISA"
Voyager	"VGER"

Note: All card types are padded with spaces to four characters.

System Error Codes and Descriptions

If an error occurs while using the OCX, DLL, or OLE/COM methods of integration, the `GetErrorCode` and `GetErrorDesc` methods available in each class should be used to determine to cause of the error. The table below shows the errors and descriptions that are returned by these methods.

Error Code	Error Description	Description
-14		A valid tid.pcc file does not exist in the directory provided in the Path property.
-1	PC-Charge not running	If an error occurred while calling `PccSysExists` (and `PccSysExists` did not set the error code), this error code will be set.
-1	Invalid Card Number	When processing a credit card transaction, if the `Send` is called and the `CheckCard` property is set to `TRUE`, `Send` will attempt to verify the credit card. If the card fails that test, this error code will be set.
	File Error	If there was an error while using the `DeleteUserFiles` method, "`File Error`" will be placed in the `Error Description` field (`GetErrorDesc`).
0	No Error	Indicates that there were no errors while the function was being performed.
1	PC-Charge not running	This error will occur if **PC**Charge is not running.
2	Batch function in progress	This error will occur If **PC**Charge is running a batch function (close/inquire/settle).
3	Repair/Compact in progress	This error will occur if **PC**Charge is running a repair or compact
4	Backup or restore in progress	This error will occur If **PC**Charge is in the process of backing up or restoring its system files.
5	Unable to initialize Modem	If **PC**Charge was unable to initialize the modem, this error will occur.
6	Timeout	If the transaction times out waiting for a reply from **PC**Charge, a timeout error will occur.
7	Database backup in progress	This error will occur if a database backup is in progress in **PC**Charge.
8	Invalid Credit Card Number	If the credit card is not a valid credit card and `VerifyCreditCard` is called, `VerifyCreditCard` will set the error code and description to Invalid Credit Card Number (for the OCX Method, it will not fire the error event.)
9	Check Service not supported	If the `Service` property is set to a service that the processor specified does not support, this error will occur.
10	Invalid Expiration Date	If the expiration date is not a valid date and `VerifyExpDate` is called, `VerifyExpDate` will set the error code and description to invalid expiration date (for the OCX Method, it will not fire the error event.)
11	Invalid Amount	If the amount is set to a negative amount or no decimal is provided and `VerifyAmount` is called, `VerifyAmount` will set the error code and description to invalid amount (for the OCX Method, it will not fire the error event.)
12	Invalid Last Valid Date	If the `LastValidDate` property is set to an invalid format the `Charge.OCX` will set the error code and description (for the OCX Method, it will not fire the error event).
13	Settlement File Locked	This error will occur if a settlement file in **PC**Charge is locked.
14	Configuration Change	This error will occur if there is a configuration change in progress.
15	Unable to erase system files	If the files cannot be erased before processing the function, this error will occur.
16	Sys.pcc unknown state	This error will occur if the sys.pcc file is an unknown state.
17	Transaction Canceled	If the .pro file is deleted while the function is being processed and the class never receives an `.oux` file, this error will occur.
18	Invalid Birth Date	If the `Birth_Date` property is set to an invalid format (**Example:** `11/02`), this error will occur.
19	Invalid Format	If the Amount property is set to an invalid format, ".2", (for the OCX Method, the ERROR event will fire).
30	No Key Serial Number Retrieved	Debit.OCX – Did not retrieve a Key Serial Number to process pin-based debit.
31	No Pin Block Retrieved	Debit.OCX – Did not retrieve a Pin Block to process pin-based debit.
50	Need to enter Driver's License when transaction amount is greater than $XX.XX	For SPS check, the Driver's License number must be supplied when the amount of the sale transaction is greater than the DL Limit amount specified in the PCCharge SPS settings.

51	Need to enter State Code when transaction amount is greater than $XX.XX	For SPS check, the State Code (GA, CA, NY, FL, etc) must be supplied when the amount of the sale transaction is greater than the DL Limit amount specified in the PCCharge SPS settings.
52	Need to enter Date of Birth when transaction amount is greater than $XX.XX	For SPS check, the Date of Birth must be supplied when the amount of the sale transaction is greater than the DL Limit amount specified in the PCCharge SPS settings.
53	Need to enter Phone Number	For SPS check, the Phone number must be supplied on all transactions.
100	Invalid File Name	If no file name or merchant number are provided before calling a method that accesses a file in the PSCharge.Offline class, this error will occur.
110	File Not Found	If the file name that was provided is not a valid file name, and a method in the PSCharge.Offline class tries to access the file, this error will occur.
120	Invalid Record Number	If attempting to void a record in a .bch file and that record does not exist, this error will occur.
150	Invalid Pccw Path	If PccwPath was not provided while performing the ProcessFile method, this error will occur.
200	Error Erasing TMP File	If there is a problem sending the .tmp file to the Recycle Bin while performing the Compact method, this error will occur.

SYS.PCC Codes and Descriptions

The presence of a file named SYS.PCC in the **PC**Charge directory indicates a busy or an error state. The following are **PC**Charge system (application) codes that will appear the SYS.PCC file if it is present. The SYS.PCC file will be written to the **PC**Charge directory and contain one of the following code only if **PC**Charge shuts down or if it is in the middle of a function that will not allow transactions to be processed. Once **PC**Charge is ready to process transactions again, the file will be deleted automatically.

Code	Description
0	PCCharge is not running
1	Batch Function in progress
2	Repair / Compact in progress
3	Backup or restore in progress
4	Unable to initialize modem
5	Database backup in progress
6	Settlement file locked
7	Configuration change in progress

Note: If using the OCX, DLL, or OLE/COM methods of integration, the following methods may be used:

- PccSysExists – use to check for the existence of the SYS.PCC
- GetErrorCode – if SYS.PCC exists, use to retrieve the code from SYS.PCC
- GetErrorDesc – if SYS.PCC exists, use to retrieve the error description associated with the code

Cashier Permissions Constants

An integrator that desires to grant Cashier Permissions through integration will be required to create and send a bitmap. This will be sent as part of the <ADD_CASHIER></ADD_CASHIER> string submitted to PCCharge. Each bit (0 = Not Allowed, 1 = Allowed) will create a section separated by pipes "|". An integrator wanting to pass a string giving a cashier only base permissions the bitmap would appear as as 6 pipes "||||||". This will default all permissions to false.

NOTE: See **Cashier Permissions** on page 75 for an example.

Sections	Permissions	Value
1: Credit	Refund Function (Credit Refunds) Void Functions (Credit Voids) Batch Functions (Batch and Settlement) AMEX_Settlement (AMEX Settlement Management) Import Functions (Importing)	Each Permission must be represented with either a **0 (Not Allowed)** or a **1(Allowed)**
2: Debit	Debit Authorizations (Debit Transactions) Debit Refund Debit Void	Each Permission must be represented with either a **0 (Not Allowed)** or a **1(Allowed)**
3: Check	Check Authorizations Check Refunds Check Voids	Each Permission must be represented with either a **0 (Not Allowed)** or a **1(Allowed)**
4: EBT	EBT Auth (EBT Transaction) EBT Refund EBT Void	Each Permission must be represented with either a **0 (Not Allowed)** or a **1(Allowed)**
5: Gift	Gift Authorization Gift Increment Gift Cancel/Void Gift Activate Gift Register Gift Points Gift Balance Merge Gift Balance Adjustment Gift Balance Transfer Gift Report Lost Stolen Gift Cash Out Gift Deactivate	Each Permission must be represented with either a **0 (Not Allowed)** or a **1(Allowed)**

6: Account Information	Account Info Reports (Reporting) Customer Database Account Access Customer Database Entry	Each Permission must be represented with either a **0 (Not Allowed)** or a **1(Allowed)**
7: System Information	Merchant Setup System Configuration Hardware Configuration Purge Trans Access User Information System Shutdown	Each Permission must be represented with either a **0 (Not Allowed)** or a **1(Allowed)**
8: CITI Private Label	Credit Application Account Lookup	Each Permission must be represented with either a **0 (Not Allowed)** or a **1(Allowed)**

Processing Company Codes

Note: The processor drop-down lists found in the various setup screens in **PC**Charge also serve as accurate lists of the available processor codes.

Credit Card

Processing Company	Processor Code
Alliance Data Systems, Inc.	ADSI
American Express	AMEX
FDMS Atlanta	BPAS
Citibank Private Label	CITI
ECHO	ECHO
FDMS Nashville / Envoy	FDCN
FDMS New Nashville	FDNN
FDMS North / Cardnet	CES
FDMS Omaha / FDR	FDC
FDMS South / NaBanco	NB
Fifth-Third Bank – St. Pete	BPS
Global Payment-East	NDC
Heartland Payment Systems	HPTS
RBS WorldPay	LYNK
National Bankcard Services	NBS
National Processing Company	NPC
Elavon (NOVA)	NOVA
Chase Paymentech	GSAR
TSYS (Formerly Vital)	VISA

Debit

Processing Company	Processor Code
Alliance Data Systems, Inc.	ADSI
FDMS Atlanta	BPAS
FDMS New Nashville	FDNN
FDMS North / CardNet	CES
FDMS Omaha / FDR	FDC
FDMS South / NaBanco	NB
Fifth-Third Bank – St. Pete	BPS
Global Payments / East	NDC
Heartland Payment Systems	HPTS
RBS WorldPay	LYNK
National Bankcard Services	NBS
National Processing Company	NPC
Elavon (NOVA)	NOVA
Chase Paymentech	GSAR
TSYS (Formerly Vital)	VISA

Check Verification / Conversion

Processing Company	Processor Code
Alliance Data Systems, Inc.	ADSI
ArJay/SCAN Data Corporation	ARJ
Certegy	EFAX
Check Services powered by RMRS	EZCK
CrossCheck	CRCK
FDMS North / CardNet	CES
Fifth-Third Bank – St. Pete	BPS
National Check Network	RMRS
Elavon (NOVA) Check Services	NOVA
Chase Paymentech Check Services	GSAR
Secure Payment Systems	SPS
TeleCheck International, Inc.	TECK

EBT

Processing Company	Processor Code
Alliance Data Systems, Inc.	ADSI
FDMS Atlanta	BPAS
Fifth-Third Bank – St. Pete	BPS
FDMS New Nashville	FDNN
National Processing Company	NPC
Chase Paymentech	GSAR
RBS WorldPay	LYNK
TSYS (Formerly Vital)	VISA

Gift

Processing Company	Processor Code
FDMS Atlanta	BPAS
Datamark Gift Card	DMRK
Fifth-Third Bank – St. Pete	BPS
Givex	GVEX
RBS WorldPay	LYNK
Mellennia	MELL
Chase Paymentech	GSAR
Secure Payment Systems	SPS
Smart Transaction Systems	SMTS
Stored Value Systems	SVSI
ValueLink	VLNK
Valutec	VTEC
TSYS (Formerly Vital)	VISA
World	WRLD

Transaction Result Constants

Result	Transaction Type	Description
CAPTURED	Monetary	Successful online transaction now ready for settlement
NOT CAPTURED	Varies	Unsuccessful online transaction
APPROVED	Non-Monetary	Successful offline transaction for Terminal based processors, or successful Pre-Authorization for Host based processors)
NOT APPROVED	Varies	Unsuccessful offline transaction or unsuccessful Pre-Authorization for Host based processors
PROCESSED	Off-line Transaction, Report	Transaction was processed (Terminal based processors only); report was generated
CANCELLED	Any	Transaction canceled by operator or modem never connected
VOIDED	Void	Successful (with most Terminal based processors)
SALE NOT FOUND	Follow On (Void, Gratuity, etc.)	Unsuccessful (with most Terminal based processors)
GRATUITY ADDED	Gratuity	Successful (Offline Transaction for Terminal based processors. Depending on the processor and amount, some Gratuity transactions may be authorized online for Terminal based processors)
Error	Varies	Unsuccessful transaction
Problem	Report	Unsuccessful Report Request
SALE RECOVERED	Debit Sale Recovery	Successful Debit Sale Recovery
RETURN RECOVERED	Debit Return Recovery	Successful Return Recovery
Settle Error	Settlement	Unsuccessful Settlement
Closed	Batch Close	Successful Batch Close
not closed	Batch Close	Unsuccessful Batch Close
OPEN TO BUY	Private Label	Successful Open to Buy Inquiry on an ADSI Private Label card
INVALID PARAM	Transaction Inquiry	Account number or TroutD not passed to Transaction Inquiry command
Accepted	Settlement	Successful Settlement
Result Code = 2	Settlement	Batch Closed/Settled
Result Code = 6	Settlement	Batch Declined
Result Code = 8	Settlement	Batch Deferred

CHAPTER 6 -- PCCharge Integration Methods

Pseudo-code

This section includes several programming algorithms that may be followed when using the OCX, DLL, or OLE/COM methods of integration to perform payment processing. These examples are intended to be general pseudo-code type of examples and are not intended to reflect any one particular programming environment.

The examples in this section represent the most common transactions that integrators may want to support when enabling payment processing in their application. Integrators can refer to the API, code samples, or contact Development support for questions about any of the transactions that are not covered in this section.

Credit Card Sale/Pre-Authorization – Retail / Card Present

This algorithm demonstrates the coding required to perform a swiped credit card transaction in a retail environment. **Note:** The processor must be configured for "Retail" or "Restaurant" in the Credit Card Company Setup in **PC**Charge to support swiped transactions.

```
'Create A Charge Object
Set Charge = new ChargeComponentOrReference

With Charge

'Set Required Initial Properties

  .Path = "C:\Program Files\active-charge\"  'Set path to the PCCharge
directory
  .User = "User1"                    'The username set up in PCCharge
  .Processor = "PROC"                'Processor Code set up in PCCharge
  .MerchantNumber = "12345"          'Merchant Number set up in PCCharge
  .Action = "1"                      '1 = sale, 4 = pre-Authorization

'Set Optional Initial Properties

  .TimeOut = 45                      'TimeOut value in seconds
  .Multi = "1"                       'Multi-trans Wait flag
  .LastValidDate = "12"              'Last Valid Date that can be accepted
  .PrintReceipts = "1"               'Number of receipts printed by PCCharge

'Check to see if PCCharge is running and available to process transactions

  If .PccSysExists then   'Notify user of error and exit procedure

    Print "Error: " & .GetErrorCode
    Print "Description: " & .GetErrorDesc

  Else  'PCCharge is running and ready to process transactions

'Collect the transaction data from card reader and user input and set the
'credit card transaction properties:

    .Card = "Account Number"     'Credit Card Account Number. It is required
                                 that this value is trimmed to remove spaces.
                                 - REQUIRED

    .ExpDate = "MMYY"           'Expiration Date on the Credit Card –
                                'REQUIRED

    .Amount = "1.00"            'The transaction amount in two decimal places
                                'format without commas - REQUIRED

    .Member = "Cardholder Name" 'The Cardholder Name - OPTIONAL

    .Manual = 1                 'Manual flag.  1 = card is swiped - REQUIRED

    .Track = "Track II data"    'Track II data from the credit card –
                                'REQUIRED Track II data qualifies the
```

```
                                ‘merchant for the best transaction rate

    .Ticket = “123456789”       ‘Invoice or ticket number assigned by the
                                ‘integrator or merchant – CONDITIONAL
                                ‘(required by some processors)

‘Validate the input.  The various .Verify methods can be used to validate the
‘credit card number, expiration date, and amount.  Validation should also
‘occur at input time.

    If .VerifyCreditCard (“Account Number”) = False Then
      ‘Exit and return to credit card input

    If .VerifyExpDate = False Then
      ‘Exit and return to credit card input

    If .VerifyAmount = False Then
      ‘Exit and return to credit card input


‘Process the transaction by calling the .Send method

    .Send 3 ‘OCX / DLL - the “3” parameter activates the XML message format
    .Send , 3 ‘OLE/COM - the “3” parameter activates the XML message format
```

NOTE: Some languages such as Delphi and C++ require that a variable be set and that all optional arguments must be passed. Check with language documentation for variable settings and syntax.

```
     Var TTYPE_XML: LongWord;   ‘C++ equivalent would be a long

     TTYPE_XML := 3;
    .Send(FileType(TTYPE_XML));



    ‘The .Send method with the “3” parameter creates a file called
    ‘<username>.inx that contains the transaction request and drops it in the
    ‘PCCharge directory.  Depending on how integration occurs, the Send
    ‘method is either a blocking (synchronous) or a non-blocking
    ‘(asynchronous) call.  The OCX supports asynchronous (event-driven)
    ‘programming.  The DLL Method supports synchronous programming.  The
    ‘OLE/COM method supports both synchronous and asynchronous programming.

    ‘If the integration is event-driven (asynchronous), the FINISH event will
    ‘fire within a few moments indicating that the transaction has completed
    ‘processing.  If the transaction cannot be submitted or a TimeOut occurs,
    ‘an ERROR event will fire instead.  These events should kick off result
    ‘checking sub-routines in the application.

    ‘If the integration is not event-driven, place the result checking sub-
    ‘routines immediately after the .Send method.

‘Check the results of the transaction using the GetResult method

    Select Case Trim(.GetResult)
```

```
'If the transaction is approved, display the results of the transaction on
'the user's screen, print a receipt, etc.

      Case "CAPTURED" , "APPROVED"  'Successful results

        Print "TRANSACTION SUCCESSFUL"
        Print .GetResult      'Transaction Result Constant
        Print .GetAuth        'Authorization code from Issuing bank
        Print .GetRefnumber   'Reference number
        Print .GetTroutD      'PCCharge Transaction Routing ID

        'Use any of the other .Get methods in the class to display
        'information about the transaction.

'If the transaction is approved, store pertinent information such as the
'TroutD, Authorization code, Result, Amount, Cardholder name, etc. in a
'database table or other storage medium.  This information can used for
'reporting purposes and to enable "follow-on" transactions such as Reversals,
'Voids and Post-Authorizations.

        INSERT INTO DatabaseTable VALUES
                    (.GetTroutD, 'The PCCharge Transaction Routing ID
                     .GetAuth,   'Authorization code from Issuing bank
                     .GetResult, 'Transaction result constant
                     .Amount,    'Transaction amount
                     .Member)    'Cardholder name

        'Store any other data desired.  Please note that the credit card
        'associations prohibit the storing any type of Track I or Track II
        'data.  If the credit card number and expiration date are to be
        'stored, they must stored in an encrypted state.

'If the transaction is not approved, display the results to the user.

      Case "NOT CAPTURED", "NOT APPROVED", "CANCELED", "Error"

        Print "TRANSACTION NOT SUCCESSFUL"
        Print .GetResult   'Transaction Result Constant
        Print .GetAuth     'The reason why transaction was not approved.
                           'This value is provided by the processor.

'If an error occurs (such as a TimeOut error), display the error code and
'description to the user.  If the integration is event-driven, this code
'would be placed in the ERROR event.

      Case Else

        Print "Error: " & .GetErrorCode
        Print "Description: " & .GetErrorDesc

    End Select

'Once the transaction has completed and all data has been extracted from the
'result file, delete the file from the PCCharge directory.

    .DeleteUserFiles 'Deletes all files associated with the transaction.  If
                     'the integration is event-driven, DeleteUserFiles should
```

```
                    'be called in both the FINISH and ERROR events to avoid
                    'double-charging or reconciliation issues.

        .Clear      'Resets all properties and provides a clean environment to
                    'process the next transaction.

        End if '.PccSysExists check

End With

'IMPORTANT - Destroy the Charge object to reset all properties and methods.
'Also, the Clear method or ClearVariables method can be used to reset all
'properties and methods in the object.

Set Charge = Nothing
```

Credit Card Sale/Pre-Authorization – Card Not Present

This algorithm demonstrates the coding required to perform a manually keyed credit card transaction in the following industries: Retail, eCommerce, and Mail Order/Telephone Order

```
'Create A Charge Object
Set Charge = new (ChargeComponentOrReference)

With Charge

'Set Required Initial Properties

  .Path = "C:\Program Files\active-charge\"  'Set path to the PCCharge
directory
  .User = "User1"                   'The username set up in PCCharge
  .Processor = "PROC"               'Processor Code set up in PCCharge
  .MerchantNumber = "12345"         'Merchant Number set up in PCCharge
  .Action = "1"                     '1 = sale, 4 = pre-Authorization

'Set Optional Initial Properties

  .TimeOut = 45                     'TimeOut value in seconds
  .Multi = "1"                      'Multi-trans Wait flag
  .LastValidDate = "12"             'Last Valid Date that can be accepted
  .PrintReceipts = "1"              'Number of receipts printed by PCCharge

'Check to see if PCCharge is running and available to process transactions

  If .PccSysExists then   'Notify user of error and exit procedure

    Print "Error: " & .GetErrorCode
    Print "Description: " & .GetErrorDesc

  Else  'PCCharge is running and ready to process transactions

'Collect the transaction data from user input and set the credit card
'transaction properties:

    .Card = "Account Number"     'Credit Card Account Number. It is required
                                  that this value is trimmed to remove spaces.
                                  - REQUIRED

    .ExpDate = "MMYY"           'Expiration Date on the Credit Card –
                                'REQUIRED

    .Amount = "1.00"            'The transaction amount in two decimal places
                                'format without commas - REQUIRED

    .Member = "Cardholder Name" 'The Cardholder Name - OPTIONAL

    .Manual = 0                 'Manual flag.  0 = card is manually keyed –
                                'REQUIRED

    .Ticket = "123456789"       'Invoice or ticket number assigned by the
                                'integrator or merchant – REQUIRED for lowest
                                'rate (also required by some processors)
```

```
    .Street = "Billing Street"  'Cardholder's street address – REQUIRED for
                                'lowest rate

    .Zip = "Billing Zip"        'Cardholder's zip code – REQUIRED for lowest
                                'rate

    .CVV2 = "123"               'Card Verification Value - OPTIONAL

'Validate the input.  The various .Verify methods can be used to validate the
'credit card number, expiration date, and amount.  Validation should also
'occur at input time.

    If .VerifyCreditCard ("Account Number") = False Then
      'Exit and return to credit card input

    If .VerifyExpDate = False Then
      'Exit and return to credit card input

    If .VerifyAmount = False Then
      'Exit and return to credit card input

'Process the transaction by calling the .Send method

    .Send 3 'OCX / DLL - the "3" parameter activates the XML message format
    .Send , 3 'OLE/COM - the "3" parameter activates the XML message format

    'The .Send method with the "3" parameter creates a file called
    '<username>.inx that contains the transaction request and drops it in the
    'PCCharge directory.  Depending on how integration occurs, the Send
    'method is either a blocking (synchronous) or a non-blocking
    '(asynchronous) call.  The OCX supports asynchronous (event-driven)
    'programming.  The DLL Method supports synchronous programming.  The
    'OLE/COM method supports both synchronous and asynchronous programming.

    'If the integration is event-driven (asynchronous), the FINISH event will
    'fire within a few moments indicating that the transaction has completed
    'processing.  If the transaction cannot be submitted or a TimeOut occurs,
    'an ERROR event will fire instead.  These events should kick off result
    'checking sub-routines in the application.

    'If the integration is not event-driven, place the result checking sub-
    'routines immediately after the .Send method.

'Check the results of the transaction using the GetResult method

    Select Case Trim(.GetResult)

      Case "CAPTURED" , "APPROVED" 'Successful results

'If the transaction is approved, display the results of the transaction on
'the user's screen, print a receipt, etc.

        Print "TRANSACTION SUCCESSFUL"
        Print .GetResult      'Transaction Result Constant
        Print .GetAuth        'Authorization code from Issuing bank
        Print .GetRefnumber   'Reference number
```

```
        Print .GetAVS          'Address Verification Response from card issuer
        Print .GetCVV2         'Card Verification Response from card issuer
        Print .GetTroutD       'PCCharge Transaction Routing ID

        'Use any of the other .Get methods in the class to display
        'information about the transaction.

'If the transaction is approved, store pertinent information such as the
'TroutD, Authorization code, Result, Amount, Cardholder name, etc. in a
'database table or other storage medium.  This information should used for
'reporting purposes and to enable "follow-on" transactions such as Voids and
'Post-Authorizations.

        INSERT INTO DatabaseTable VALUES
                    (.GetTroutD, 'The PCCharge Transaction Routing ID
                     .GetAuth,   'Authorization code from Issuing bank
                     .GetResult, 'Transaction result constant
                     .Amount,    'Transaction amount
                     .Member)    'Cardholder name

        'Store any other data desired.  Please note that the credit card
        'associations prohibit the storing of the CVV2, CVC2, or CID values.
        'If the credit card number and expiration date are to be stored, they
        'must stored in an encrypted state.

'If the transaction is not approved, display the results to the user.

      Case "NOT CAPTURED", "NOT APPROVED", "CANCELED", "ERROR"

        Print "TRANSACTION NOT SUCCESSFUL"
        Print .GetResult   'Transaction Result Constant
        Print .GetAuth     'The reason why transaction was not approved.
                           'This value is provided by the processor.

'If an error occurs (such as a TimeOut error), display the error code and
'description to the user.  If the integration is event-driven, this code
'would be placed in the ERROR event.

      Case Else

        Print "Error: " & .GetErrorCode
        Print "Description: " & .GetErrorDesc

    End Select

'Once the transaction has completed and all data has been extracted from the
'result file, delete the file from the PCCharge directory.

    .DeleteUserFiles 'Deletes all files associated with the transaction.  If
                     'the integration is event-driven, DeleteUserFiles should
                     'be called in both the FINISH and ERROR events to avoid
                     'double-charging or reconciliation issues.

    .Clear         'Resets all properties and provides a clean environment to
                   'process the next transaction.

  End if '.PccSysExists check
```

```
End With

'IMPORTANT - Destroy the Charge object to reset all properties and methods.
'Also, the Clear method or ClearVariables method can be used to reset all
'properties and methods in the object.

Set Charge = Nothing
```

Level II (Commercial, Purchasing, etc.) Card Sale

This algorithm demonstrates the coding required to perform a swiped Level II Sale transaction. Typically, commercial, purchasing, procurement, business, and government card transactions require additional information in order for them to qualify for the lowest rates.

```
'Create A Charge Object
Set Charge = new ChargeComponentOrReference

With Charge

'Set Required Initial Properties

  .Path = "C:\Program Files\active-charge\"  'Set path to the PCCharge
directory
  .User = "User1"                    'The username set up in PCCharge
  .Processor = "PROC"                'Processor Code set up in PCCharge
  .MerchantNumber = "12345"          'Merchant Number set up in PCCharge

  'If Level II card processing will supported in the application, use the
  '.CommercialCardType method (in Charge.OCX or in the DLL's Charge class)
  'or the .CommercialCard function (in the PccBin OLE Class) to determine
  'whether the credit card is a commercial card prior to assigning the
  '.Action.  For example, if the transaction is a sale, and the card is a
  'commercial card, the action code should be set to "8" and customer should
  'be prompted for the level II data (tax and customer code).  Otherwise, the
  'action code should be set to "1" for a standard credit card sale.

  .Action = "8"                      '8 = Level II card sale

'Set Optional Initial Properties

  .TimeOut = 45                      'TimeOut value in seconds
  .Multi = "1"                       'Multi-trans Wait flag
  .LastValidDate = "12"              'Last Valid Date that can be accepted
  .PrintReceipts = "1"               'Number of receipts printed by PCCharge

'Check to see if PCCharge is running and available to process transactions

  If .PccSysExists then   'Notify user of error and exit procedure

    Print "Error: " & .GetErrorCode
    Print "Description: " & .GetErrorDesc

  Else  'PCCharge is running and ready to process transactions

'Collect the transaction data from card reader and user input and set the
'credit card transaction properties:

     .Card = "Account Number"     'Credit Card Account Number. It is required
                                  that this value is trimmed to remove spaces.
                                  - REQUIRED
```

```
    .ExpDate = "MMYY"           'Expiration Date on the Credit Card –
                                'REQUIRED

    .Amount = "1.00"            'The transaction amount in two decimal places
                                'format without commas - REQUIRED

    .Member = "Cardholder Name" 'The Cardholder Name - OPTIONAL

    .Manual = 1                 'Manual flag.  1 = card is swiped - REQUIRED

    .Track = "Track II data"    'Track II data from the credit card –
                                'REQUIRED Track II data qualifies the
                                'merchant for the best transaction rate

    .Ticket = "123456789"       'Invoice or ticket number assigned by the
                                'integrator or merchant – CONDITIONAL
                                '(required by some processors)

                                'rates

    .CustCode = "Customer Code" 'Cardholder's Customer code – REQUIRED for
                                'lowest rates

    .TaxAmt = "0.05"            'The amount of tax included in the total
                                'amount (inclusive).  Set to 0.00 if customer
                                'is tax exempt - REQUIRED

    .TaxExempt = False          'Tax Exempt Flag – REQUIRED

'If using the OCX or DLL Method, use the following code to set the
'.CommercialCardFlag:

    .CommercialCardFlag = .getCommercialCardType("Account Number")

'If using the OLE/COM Method, the PccBin class must also be used.  Use the
'following code to set the .CmrclCardFlag:

    If PccBin1.CommercialCard("Account Number") Then
      .CmrclCardFlag = PccBin1.CommercialCardType
    End If

'Validate the input.  The various .Verify methods can be used to validate the
'credit card number, expiration date, and amount.  Validation should also
'occur at input time.

    If .VerifyCreditCard ("Account Number") = False Then
      'Exit and return to credit card input

    If .VerifyExpDate = False Then
      'Exit and return to credit card input

    If .VerifyAmount = False Then
      'Exit and return to credit card input


'Process the transaction by calling the .Send method
```

```
    .Send 3 'OCX / DLL - the "3" parameter activates the XML message format
    .Send , 3 'OLE/COM - the "3" parameter activates the XML message format

    'The .Send method with the "3" parameter creates a file called
    '<username>.inx that contains the transaction request and drops it in the
    'PCCharge directory.  Depending on how integration occurs, the Send
    'method is either a blocking (synchronous) or a non-blocking
    '(asynchronous) call.  The OCX supports asynchronous (event-driven)
    'programming.  The DLL Method supports synchronous programming.  The
    'OLE/COM method supports both synchronous and asynchronous programming.

    'If the integration is event-driven (asynchronous), the FINISH event will
    'fire within a few moments indicating that the transaction has completed
    'processing.  If the transaction cannot be submitted or a TimeOut occurs,
    'an ERROR event will fire instead.  These events should kick off result
    'checking sub-routines in the application.

    'If the integration is not event-driven, place the result checking sub-
    'routines immediately after the .Send method.

'Check the results of the transaction using the GetResult method

   Select Case Trim(.GetResult)

'If the transaction is approved, display the results of the transaction on
'the user's screen, print a receipt, etc.

     Case "CAPTURED" , "APPROVED"  'Successful results

       Print "TRANSACTION SUCCESSFUL"
       Print .GetResult      'Transaction Result Constant
       Print .GetAuth        'Authorization code from Issuing bank
       Print .GetRefnumber   'Reference number
       Print .GetTroutD      'PCCharge Transaction Routing ID

       'Use any of the other .Get methods in the class to display
       'information about the transaction.

'If the transaction is approved, store pertinent information such as the
'TroutD, Authorization code, Result, Amount, Cardholder name, etc. in a
'database table or other storage medium.  This information can used for
'reporting purposes and to enable "follow-on" transactions such as Voids and
'Post-Authorizations.

       INSERT INTO DatabaseTable VALUES
                   (.GetTroutD, 'The PCCharge Transaction Routing ID
                    .GetAuth,   'Authorization code from Issuing bank
                    .GetResult, 'Transaction result constant
                    .Amount,    'Transaction amount
                    .Member)    'Cardholder name

       'Store any other data desired.  Please note that the credit card
       'associations prohibit the storing any type of Track I or Track II
       'data.  If the credit card number and expiration date are to be
       'stored, they must stored in an encrypted state.

'If the transaction is not approved, display the results to the user.
```

```
      Case "NOT CAPTURED", "NOT APPROVED", "CANCELED", “Error”

        Print “TRANSACTION NOT SUCCESSFUL”
        Print .GetResult   ‘Transaction Result Constant
        Print .GetAuth     ‘The reason why transaction was not approved.
                           ‘This value is provided by the processor.

‘If an error occurs (such as a TimeOut error), display the error code and
‘description to the user.  If the integration is event-driven, this code
‘would be placed in the ERROR event.

      Case Else

        Print “Error: “ & .GetErrorCode
        Print “Description: “ & .GetErrorDesc

    End Select

‘Once the transaction has completed and all data has been extracted from the
‘result file, delete the file from the PCCharge directory.

    .DeleteUserFiles ‘Deletes all files associated with the transaction.  If
                     ‘the integration is event-driven, DeleteUserFiles should
                     ‘be called in both the FINISH and ERROR events to avoid
                     ‘double-charging or reconciliation issues.

    .Clear         ‘Resets all properties and provides a clean environment to
                   ‘process the next transaction.

  End if ‘.PccSysExists check

End With

‘IMPORTANT - Destroy the Charge object to reset all properties and methods.
‘Also, the Clear method or ClearVariables method can be used to reset all
‘properties and methods in the object.

Set Charge = Nothing
```

Credit Card Void

This algorithm demonstrates the coding required to perform a credit card void transaction. A void is considered a **PC**Charge "follow-on" transaction because of its use of the TroutD functionality.

```
'Create A Charge Object
Set Charge = new (ChargeComponentOrReference)

With Charge

'Set Required Initial Properties

  .Path = "C:\Program Files\active-charge\"  'Set path to the PCCharge
directory
  .User = "User1"                    'The username set up in PCCharge
  .Action = "3"                      '3 = void; 6 = void credit; 7 = void
                                     'post-authorization; 17 = void ship

'Set Optional Initial Properties

  .TimeOut = 45                      'TimeOut value in seconds
  .Multi = "1"                       'Multi-trans Wait flag
  .PrintReceipts = "1"               'Number of receipts printed by PCCharge

'Check to see if PCCharge is running and available to process transactions

  If .PccSysExists then   'Notify user of error and exit procedure

    Print "Error: " & .GetErrorCode
    Print "Description: " & .GetErrorDesc

  Else  'PCCharge is running and ready to process transactions

'The third-party application should provide a list of transactions that can
'be voided (or other similar option).  The application should allow the user
'to pick the transaction to be voided from the list.  The third-party
'application should now pass the transaction's stored TroutD value to
'PCCharge:

    .TroutD = "1234"          'Transaction Routing ID from the original Sale
                              'or Post-Authorization that will be voided -
                              'REQUIRED

'Process the transaction by calling the .Send method

    .Send 3 'OCX / DLL - the "3" parameter activates the XML message format
    .Send , 3 'OLE/COM - the "3" parameter activates the XML message format

    'The .Send method with the "3" parameter creates a file called
    '<username>.inx that contains the transaction request and drops it in the
    'PCCharge directory.  Depending on how integration occurs, the Send
    'method is either a blocking (synchronous) or a non-blocking
    '(asynchronous) call.  The OCX supports asynchronous (event-driven)
    'programming.  The DLL Method supports synchronous programming.  The
    'OLE/COM method supports both synchronous and asynchronous programming.
```

```
    'If the integration is event-driven (asynchronous), the FINISH event will
    'fire within a few moments indicating that the transaction has completed
    'processing.  If the transaction cannot be submitted or a TimeOut occurs,
    'an ERROR event will fire instead.  These events should kick off result
    'checking sub-routines in the application.

    'If the integration is not event-driven, place the result checking sub-
    'routines immediately after the .Send method.

'Check the results of the transaction using the GetResult method

   Select Case Trim(.GetResult)

     Case "VOIDED", "CAPTURED"  'Successful results

'If the transaction is voided, display the results of the transaction on
'the user's screen, print a receipt, etc.

       Print "TRANSACTION SUCCESSFUL"
       Print .GetResult      'Transaction Result Constant
       Print .GetAuth        'Void Response (may be returned)
       Print .GetRefnumber   'Reference number (may be returned)
       Print .GetTroutD      'PCCharge Transaction Routing ID

       'Use any of the other .Get methods in the class to display
       'information about the transaction.

'If the transaction is voided successfully, update the transaction's status
'in the third-party application's database table.

       SELECT * FROM DatabaseTable WHERE TroutD="1234"
       "Status".Value = "VOIDED"

'If the void fails for some reason, display the results to the user.

     Case "Sale Not Found", "Error"

       Print "TRANSACTION NOT SUCCESSFUL"
       Print .GetResult   'Transaction Result Constant
       Print .GetAuth     'The reason why transaction was not approved.
                          'This value is provided by the processor.

'If an error occurs (such as a TimeOut error), display the error code and
'description to the user.  If the integration is event-driven, this code
'would be placed in the ERROR event.

     Case Else

       Print "Error: " & .GetErrorCode
       Print "Description: " & .GetErrorDesc

   End Select


'Once the transaction has completed and all data has been extracted from the
'result file, delete the file from the PCCharge directory.
```

```
    .DeleteUserFiles 'Deletes all files associated with the transaction.  If
                     'the integration is event-driven, DeleteUserFiles should
                     'be called in both the FINISH and ERROR events to avoid
                     'reconciliation issues.

     .Clear          'Resets all properties and provides a clean environment to
                     'process the next transaction.

  End if '.PccSysExists check

End With

'IMPORTANT - Destroy the Charge object to reset all properties and methods.
'Also, the Clear method or ClearVariables method can be used to reset all
'properties and methods in the object.

Set Charge = Nothing
```

Credit Card Sale/Pre-Authorization – Restaurant

This algorithm demonstrates the coding required to perform a swiped credit card transaction in a restaurant environment.

Note: The processor must support and be configured for "Restaurant" in the Credit Card Company Setup in **PC**Charge to support restaurant transactions.

Note: Manually keyed transactions are also supported in a restaurant environment. If transactions will be manually keyed, make sure the appropriate rate qualifying information is sent with each transaction. See the **Credit Card Sale/Pre-Authorization – Card Not Present** algorithm (see page 163) for more information.

```
'Create A Charge Object
Set Charge = new (ChargeComponentOrReference)

With Charge

'Set Required Initial Properties

  .Path = "C:\Program Files\active-charge\"  'Set path to the PCCharge
                                              directory
  .User = "User1"                          'The username set up in PCCharge
  .Processor = "PROC"                      'Processor Code set up in PCCharge
  .MerchantNumber = "12345"                'Merchant Number set up in PCCharge
  .Action = "1"                            '1 = sale, 4 = pre-Authorization

'Set Optional Initial Properties

  .TimeOut = 45                            'TimeOut value in seconds
  .Multi = "1"                             'Multi-trans Wait flag
  .LastValidDate = "12"                    'Last Valid Date that can be accepted
  .PrintReceipts = "1"                     'Number of receipts printed by PCCharge

'Check to see if PCCharge is running and available to process transactions

  If .PccSysExists then   'Notify user of error and exit procedure

    Print "Error: " & .GetErrorCode
    Print "Description: " & .GetErrorDesc

  Else  'PCCharge is running and ready to process transactions

'Collect the transaction data from card reader and user input and set the
'credit card transaction properties:

    .Card = "Account Number"     'Credit Card Account Number. It is required
                                 that this value is trimmed to remove spaces.
                                 - REQUIRED


    .ExpDate = "MMYY"           'Expiration Date on the Credit Card –
                                'REQUIRED

    .Amount = "10.00"           'The transaction amount in two decimal places
                                'format without commas - REQUIRED
```

```
    .Member = "Cardholder Name"  'The Cardholder Name - OPTIONAL

    .Manual = 1                  'Manual flag.  1 = card is swiped - REQUIRED

    .Track = "Track II data"     'Track II data from the credit card –
                                 'REQUIRED to qualify the merchant for the
                                 'best transaction rate

    .Ticket = "123456789"        'Invoice or ticket number assigned by the
                                 'integrator or merchant – CONDITIONAL
                                 '(required by some processors)

    .EstGratuityAmount = "1.50"  'The estimated gratuity amount (not added to
                                 'settlement amount) in two decimal places
                                 'format without commas- OPTIONAL

    .MCSN = "02"                 'The Server ID – OPTIONAL Processor specific note: The
                                  Server ID is required for AMEX card transactions in restaurant setting.

'Validate the input.  The various .Verify methods can be used to validate the
'credit card number, expiration date, and amount.  Validation should also
'occur at input time.

    If .VerifyCreditCard ("Account Number") = False Then
      'Exit and return to credit card input

    If .VerifyExpDate = False Then
      'Exit and return to credit card input

    If .VerifyAmount = False Then
      'Exit and return to credit card input

'Process the transaction by calling the .Send method

    .Send 3 'OCX / DLL - the "3" parameter activates the XML message format
    .Send , 3 'OLE/COM - the "3" parameter activates the XML message format

    'The .Send method with the "3" parameter creates a file called
    '<username>.inx that contains the transaction request and drops it in the
    'PCCharge directory.  Depending on how integration occurs, the Send
    'method is either a blocking (synchronous) or a non-blocking
    '(asynchronous) call.  The OCX supports asynchronous (event-driven)
    'programming.  The DLL Method supports synchronous programming.  The
    'OLE/COM method supports both synchronous and asynchronous programming.

    'If the integration is event-driven (asynchronous), the FINISH event will
    'fire within a few moments indicating that the transaction has completed
    'processing.  If the transaction cannot be submitted or a TimeOut occurs,
    'an ERROR event will fire instead.  These events should kick off result
    'checking sub-routines in the application.

    'If the integration is not event-driven, place the result checking sub-
    'routines immediately after the .Send method.

'Check the results of the transaction using the GetResult method
```

```
   Select Case Trim(.GetResult)

     Case "CAPTURED" , "APPROVED" 'successful results

'If the transaction is approved, display the results of the transaction on
'the user's screen, print a receipt, etc.

       Print "TRANSACTION SUCCESSFUL"
       Print .GetResult      'Transaction Result Constant
       Print .GetAuth        'Authorization code from Issuing bank
       Print .GetRefnumber   'Reference number
       Print .GetTroutD      'PCCharge Transaction Routing ID

       'Use any of the other .Get methods in the class to display \
       'information about the transaction.

'If the transaction is approved, store pertinent information such as the
'TroutD, Authorization code, amount, cardholder name, etc. in a database
'table or other storage medium.  This information should used for reporting
'purposes and to enable "follow-on" transactions such as Gratuity, Voids and
'Post-Authorizations.

       INSERT INTO DatabaseTable VALUES
                   (.GetTroutD, 'The PCCharge Transaction Routing ID
                    .GetAuth,   'Authorization code from Issuing bank
                    .GetResult, 'Transaction result constant
                    .Amount,    'Transaction amount
                    .Member)    'Cardholder name

       'Store any other data desired.  Please note that the credit card
       'associations prohibit the storing any type of Track I or Track II
       'data.  If the credit card number and expiration date are to be
       'stored, they must stored in an encrypted state.

'If the transaction is not approved, display the results to the user.

     Case "NOT CAPTURED", "NOT APPROVED", "CANCELED", "Error"

       Print "TRANSACTION NOT SUCCESSFUL"
       Print .GetResult   'Transaction Result Constant
       Print .GetAuth     'The reason why transaction was not approved.
                          'This value is provided by the processor.

'If an error occurs (such as a TimeOut error), display the error code and
'description to the user.  If the integration is event-driven, this code
'would be placed in the ERROR event.

     Case Else

       Print "Error: " & .GetErrorCode
       Print "Description: " & .GetErrorDesc

   End Select

'Once the transaction has completed and all data has been extracted from the
'result file, delete the file from the PCCharge directory.
```

```
    .DeleteUserFiles 'Deletes all files associated with the transaction.  If
                     'the integration is event-driven, DeleteUserFiles should
                     'be called in both the FINISH and ERROR events to avoid
                     'double-charging or reconciliation issues.

     .Clear          'Resets all properties and provides a clean environment to
                     'process the next transaction.

  End if '.PccSysExists check

End With

'IMPORTANT - Destroy the Charge object to reset all properties and methods.
'Also, the Clear method or ClearVariables method can be used to reset all
'properties and methods in the object.

Set Charge = Nothing
```

Credit Card Gratuity – Restaurant

This algorithm demonstrates the coding required to perform a gratuity in a restaurant environment. A Gratuity is considered a **PC**Charge "follow-on" transaction because of its use of the TroutD functionality.

Note: The processor must support and be configured for "Restaurant" in the Credit Card Company Setup in **PC**Charge to support restaurant transactions.

Note: PC Charge supports gratuity adjustments. If, for any reason, an existing gratuity is incorrect or the gratuity must be changed after it has been added to the transaction, the gratuity action (action code 13) can be used to correct or change the amount as many times as needed prior to settlement.

```
'Create A Charge Object
Set Charge = new (ChargeComponentOrReference)

With Charge

'Set Required Initial Properties

  .Path = "C:\Program Files\active-charge\"  'Set path to the PCCharge
directory
  .User = "User1"                    'The username set up in PCCharge
  .Action = "13"                     '13 = Gratuity add / adjustment

'Set Optional Initial Properties

  .TimeOut = 45                      'TimeOut value in seconds
  .Multi = "1"                       'Multi-trans Wait flag
  .PrintReceipts = "1"               'Number of receipts printed by PCCharge

'Check to see if PCCharge is running and available to process transactions

  If .PccSysExists then   'Notify user of error and exit procedure

    Print "Error: " & .GetErrorCode
    Print "Description: " & .GetErrorDesc

  Else  'PCCharge is running and ready to process transactions

'The third-party application should provide a list of transactions that can
'be voided (or other similar option).  The application should allow the user
'to pick the transaction to be voided from the list.  The third-party
'application should now pass the transaction's stored TroutD value to
'PCCharge:


    .TroutD = "1234"           'Transaction Routing ID from the original Sale
                               'or Post-Authorization that will be voided –
                               'REQUIRED

    .GratuityAmount = "1.50"  'The estimated gratuity amount - REQUIRED

'Process the transaction by calling the .Send method
```

```
    .Send 3 'OCX / DLL - the "3" parameter activates the XML message format
    .Send , 3 'OLE/COM - the "3" parameter activates the XML message format

    'The .Send method with the "3" parameter creates a file called
    '<username>.inx that contains the transaction request and drops it in the
    'PCCharge directory.  Depending on how integration occurs, the Send
    'method is either a blocking (synchronous) or a non-blocking
    '(asynchronous) call.  The OCX supports asynchronous (event-driven)
    'programming.  The DLL Method supports synchronous programming.  The
    'OLE/COM method supports both synchronous and asynchronous programming.

    'If the integration is event-driven (asynchronous), the FINISH event will
    'fire within a few moments indicating that the transaction has completed
    'processing.  If the transaction cannot be submitted or a TimeOut occurs,
    'an ERROR event will fire instead.  These events should kick off result
    'checking sub-routines in the application.

    'If the integration is not event-driven, place the result checking sub-
    'routines immediately after the .Send method.

'Check the results of the transaction using the GetResult method

   Select Case Trim(.GetResult)

     Case "Gratuity Added"  'Successful result

'If the gratuity is added or adjusted, display the results of the transaction
'on the user's screen, print a receipt, etc.

       Print "TRANSACTION SUCCESSFUL"
       Print .GetResult      'Transaction Result Constant
       Print .GetAuth        'Void Response (may be returned)
       Print .GetRefnumber   'Reference number (may be returned)
       Print .GetTroutD      'PCCharge Transaction Routing ID

       'Use any of the other .Get methods in the class to display
       'information about the transaction.

'If the gratuity is added or adjusted successfully, in the third-party
'application's database table: add the gratuity to the transaction and update
'the transaction's status.

       SELECT * FROM DatabaseTable WHERE TroutD="1234"
       "Status".Value = "Gratuity Added"
       "Gratuity Amount".Value = .GratuityAmount


'If the gratuity transaction fails for some reason, display the results to
'the user.

     Case "Sale Not Found", "Error"

       Print "TRANSACTION NOT SUCCESSFUL"
       Print .GetResult   'Transaction Result Constant
       Print .GetAuth     'The reason why transaction was not approved.
                          'This value is provided by the processor.
```

```
'If an error occurs (such as a TimeOut error), display the error code and
'description to the user.  If the integration is event-driven, this code
'would be placed in the ERROR event.

      Case Else

        Print "Error: " & .GetErrorCode
        Print "Description: " & .GetErrorDesc

    End Select

'Once the transaction has completed and all data has been extracted from the
'result file, delete the file from the PCCharge directory.

    .DeleteUserFiles 'Deletes all files associated with the transaction.  If
                     'the integration is event-driven, DeleteUserFiles should
                     'be called in both the FINISH and ERROR events to avoid
                     'reconciliation issues.

     .Clear         'Resets all properties and provides a clean environment to
                    'process the next transaction.

  End if '.PccSysExists check

End With

'IMPORTANT - Destroy the Charge object to reset all properties and methods.
'Also, the Clear method or ClearVariables method can be used to reset all
'properties and methods in the object.

Set Charge = Nothing
```

Debit Sale

This algorithm demonstrates the coding required to perform an online debit transaction in a retail environment. Specifically, this algorithm demonstrates processing a debit transaction using DUKPT encryption using a Verifone 1000/101 PINpad. Also, this algorithm only applies to U.S.-based debit processing companies.

To integrate a PINpad, use `Device.OCX` or the `PccPinPad` OLE class provided with the DevKit. Alternatively, a direct integration to the PINpad may also be used. Contact the PINpad manufacturer(s) for details on integrating directly to PINpads.

```
'Declare local variables to store the Key Serial Number and the PIN that are
'retrieved from the PINpad.

Dim KSN, EncryptedPIN as String

'Initialize the PINpad.  To initialize using Device.OCX or the PccPinPad OLE
'class, set required properties such as the baud rate, parity, etc. and then
'call the .Initialize method.

If PinPad.Initialize Then MsgBox "PINpad Initialized"

'Create A Debit Object

Set Debit = new DebitComponentOrReference

With Debit

'Set Required Initial Properties

  .Path = "C:\Program Files\active-charge\"  'Set path to the PCCharge
directory
  .User = "User1"                    'The username set up in PCCharge
  .Processor = "PROC"                'Processor Code set up in PCCharge
  .MerchantNumber = "12345"          'Merchant Number set up in PCCharge
  .Action = "41"                     '41 = Debit sale

'Set Optional Initial Properties

  .TimeOut = 45                      'TimeOut value in seconds

'Check to see if PCCharge is running and available to process transactions

  If .PccSysExists then   'Notify user of error and exit procedure

    Print "Error: " & .GetErrorCode
    Print "Description: " & .GetErrorDesc

  Else  'PCCharge is running and ready to process transactions

'Collect the debit transaction information from the card reader and user
'input.  Once the debit transaction information has been collected, prompt
'the cardholder for their PIN.
```

```
'If using the Device.OCX:

  'Set the .Card and .Amount properties and then call the .GetPin method:

  PinPad.Card = "Account Number"  'Must send Debit Card Number to PINpad
  PinPad.Amount = "25.00"         'Must send transaction Amount + CashBack
                                  'amount to PINpad

  'Collect the data by calling the .GetPin and .GetKeySerialNumber methods:

  EncryptedPIN = .GetPin          'Prompts the customer for the PIN and
                                  'returns the Encrypted PIN value
  KSN = .GetKeySerialNumber       'Returns the Key Serial Number – not
                                  'returned by all PINpads


'If using the PccPinPad OLE class:

  'Set the .Card and .Amount properties and then call the .GetPin method:

  PinPad.Card = "Account Number"  'Must send Debit Card Number to PINpad
  PinPad.Amount = "25.00"         'Must send transaction Amount + CashBack
                                  'amount to PINpad

  If PinPad.GetPin Then           'Prompts the customer for the PIN. After
                                  'the customer has successfully entered
                                  'the PIN into the PINpad, the .GetPin
                                  'method will return TRUE.  The encrypted
                                  'PIN and Key Serial Number are available
                                  'in the .PIN and .KeySerialNumber
                                  'properties

    EncryptedPIN = .PIN           'The Encrypted PIN value
    KSN = .KeySerialNumber        'The Key Serial Number – not returned by
                                  'all PINpads

  End If

'If using a direct integration to a PINpad, collect the Encrypted PIN and Key
'Serial Number from the PINpad and assign these values to the proper
'variables.


'Set the debit card transaction properties:

  .Card = "Account Number"    'Debit Card Account Number - REQUIRED

  .ExpDate = "MMYY"           'Expiration Date on the Debit Card - REQUIRED

  .Amount = "20.00"           'The transaction amount in two decimal places
                              'format without commas – REQUIRED

  .CashBack = "5.00"          'The CashBack amount in two decimal places
                              'format without commas - OPTIONAL

  .Member = "Cardholder Name" 'The Cardholder Name - OPTIONAL
```

```
    .Manual = 1                 'Manual flag.  1 = card is swiped - REQUIRED

    .Track = "Track II data"    'Track II data from the debit card – REQUIRED

    .Ticket = "123456789"       'Invoice or ticket number assigned by the
                                'integrator or merchant – CONDITIONAL
                                '(required by some processors)

    .Pin = EncryptedPIN         'PIN from PINpad - REQUIRED

    .KeySerialNumber = KSN      'Key Serial number from PINpad – CONDITIONAL
                                '(populate if returned by the PINpad)

'Process the transaction by calling the .Send method

    .Send 3 'OCX / DLL - the "3" parameter activates the XML message format
    .Send , 3 'OLE/COM - the "3" parameter activates the XML message format

    'The .Send method with the "3" parameter creates a file called
    '<username>.inx that contains the transaction request and drops it in the
    'PCCharge directory.  Depending on how integration occurs, the Send
    'method is either a blocking (synchronous) or a non-blocking
    '(asynchronous) call.  The OCX supports asynchronous (event-driven)
    'programming.  The DLL Method supports synchronous programming.  The
    'OLE/COM method supports both synchronous and asynchronous programming.

    'If the integration is event-driven (asynchronous), the FINISH event will
    'fire within a few moments indicating that the transaction has completed
    'processing.  If the transaction cannot be submitted or a TimeOut occurs,
    'an ERROR event will fire instead.  These events should kick off result
    'checking sub-routines in the application.

    'If the integration is not event-driven, place the result checking sub-
    'routines immediately after the .Send method.

'Check the results of the transaction using the GetResult method

   Select Case Trim(.GetResult)

'If the transaction is approved, display the results of the transaction on
'the user's screen, print a receipt, etc.

     Case "CAPTURED" , "APPROVED"  'Successful results

       Print "TRANSACTION SUCCESSFUL"
       Print .GetResult      'Transaction Result Constant
       Print .GetAuth        'Authorization code from Issuing bank
       Print .GetTroutD      'PCCharge Transaction Routing ID

       'Use any of the other .Get methods in the class to display
       'information about the transaction.

'If the transaction is approved, store pertinent information such as the
'TroutD, Authorization code, Result, Amount, Cardholder name, etc. in a
'database table or other storage medium.  This information can used for
'reporting purposes.
```

```
        INSERT INTO DatabaseTable VALUES
                    (.GetTroutD, 'The PCCharge Transaction Routing ID
                     .GetAuth,   'Authorization code from Issuing bank
                     .GetResult, 'Transaction result constant
                     .Amount,    'Transaction amount
                     .Member)    'Cardholder name

        'Store any other data desired.  Please note that the card
        'associations prohibit the storing any type of Track I or Track II
        'data or PIN information.  If the debit card number and expiration
        'date are to be stored, they must stored in an encrypted state.

'If the transaction is not approved, display the results to the user.

      Case "NOT CAPTURED", "NOT APPROVED", "CANCELED", "Error"

        Print "TRANSACTION NOT SUCCESSFUL"
        Print .GetResult   'Transaction Result Constant
        Print .GetAuth     'The reason why transaction was not approved.
                           'This value is provided by the processor.

'If an error occurs (such as a TimeOut error), display the error code and
'description to the user.  If the integration is event-driven, this code
'would be placed in the ERROR event.

      Case Else

        Print "Error: " & .GetErrorCode
        Print "Description: " & .GetErrorDesc

    End Select

'Once the transaction has completed and all data has been extracted from the
'result file, delete the file from the PCCharge directory.

    .DeleteUserFiles 'Deletes all files associated with the transaction.  If
                     'the integration is event-driven, DeleteUserFiles should
                     'be called in both the FINISH and ERROR events to avoid
                     'double-charging or reconciliation issues.

     .Clear        'Resets all properties and provides a clean environment to
                   'process the next transaction.

  End if '.PccSysExists check

End With

'IMPORTANT - Destroy the Debit object to reset all properties and methods.
'Also, the Clear method or ClearVariables method can be used to reset all
'properties and methods in the object.

Set Debit = Nothing
```

Reports

This algorithm demonstrates the coding required to submit a report request to **PC**Charge. To submit a report request using the OCX Method, use `Charge.OCX`, for the DLL Method, use `PSCharge.charge`, and for OLE/COM, use `ActiveCharge.PccCharge`.

```
'Create A Charge Object
Set Charge = new (ChargeComponentOrReference)

With Charge

'Set Required Initial Properties

  .Path = "C:\Program Files\active-charge\"  'Set path to the PCCharge
                                              directory
  .User = "User1"                  'The username set up in PCCharge
  .CheckCard = False               'Turns of credit card validity testing.
                                   'Only necessary for the OCX and DLL
                                   'methods
  .Action = "81"                   '81 = Credit Card Detail; 82 = Batch Pre-
                                   'Settle; 83 = Batch Post-Settle; 84 =
                                   'Check Summary


'Set Optional Initial Properties

  .TimeOut = 45                    'TimeOut value in seconds

'If no other parameters are passed, PCCharge will print a report for the
'current 24 hour period to the default report printer that is set up in
'PCCharge once the Send method is called.  Integrators may pass in optional
'parameters for output and filtering to customize the reports and how they
'are delivered.

'Set Optional Output parameters

  .PeriodicPayment = "1"           'Output type:  0 – Print to PCCharge's
                                   'default report printer; 1 – Print to
                                   'file using filename specified in TRANSID

  .Track = "C:\Documents and Settings\JohnD\Desktop\"  'Destination directory
                                                       'for output file (40
                                                       'characters max)

  .CustCode = "ReportDirectory\"   'Continuation of destination directory if
                                   'directory name is greater than 40
                                   'characters.  The values in Track and
                                   'CustCode are concatenated together once
                                   'submitted.  The last character of the
                                   'directory name must be "\". (25
                                   'characters max)

  .TRANSID = Report.pdf            'Filename and extension of the report
                                   'file.  The extension determines the file
                                   'type returned.  Valid extensions: .pdf,
                                   '.rtf, or .txt
```

```
'Set Optional Filter parameters

  .Card = "User1"                    'User name Filter
  .MerchantNumber = "999999999911"   'Merchant Number Filter
  .Street = 04/07/08 12:00:00 PM     'Starting Date / Time Filter
  .Member = 04/08/08 11:00:00 AM     'Ending Date / Time Filter
  .Manual = 1                        'Transaction Result Filter: 0 = all
                                     '(default); 1 = approved; 2 = declined

'Check to see if PCCharge is running and available to accept requests

  If .PccSysExists then   'Notify user of error and exit procedure

    Print "Error: " & .GetErrorCode
    Print "Description: " & .GetErrorDesc

  Else  'PCCharge is running and ready to process transactions

'Submit the report request by calling the .Send method

    .Send 3 'OCX / DLL - the "3" parameter activates the XML message format
    .Send , 3 'OLE/COM - the "3" parameter activates the XML message format

    'The .Send method with the "3" parameter creates a file called
    '<username>.inx that contains the transaction request and drops it in the
    'PCCharge directory.  Depending on how integration occurs, the Send
    'method is either a blocking (synchronous) or a non-blocking
    '(asynchronous) call.  The OCX supports asynchronous (event-driven)
    'programming.  The DLL Method supports synchronous programming.  The
    'OLE/COM method supports both synchronous and asynchronous programming.

    'If the integration is event-driven (asynchronous), the FINISH event will
    'fire within a few moments indicating that the report request has
    'completed.  If the report request cannot be submitted or a TimeOut
    'occurs, an ERROR event will fire instead.  These events should kick off
    'result checking sub-routines in the application.

    'If the integration is not event-driven, place the result checking sub-
    'routines immediately after the .Send method.

'Check the results of the report request using the GetResult method

    Select Case Trim(.GetResult)

      Case "Processed"

'If the report request is successful, notify the user that the report has
'been printed or the report file is available.  If accessing the report
'programmatically, begin parsing the report.

        Print "REPORT NOW AVAILABLE"

      Case "Problem"

'If the report request fails for some reason, display a message to the user.
```

```
        Print "REPORT REQUEST UNSUCCESSFUL"

'If an error occurs (such as a TimeOut error), display the error code and
'description to the user.  If the integration is event-driven, this code
'would be placed in the ERROR event.

      Case Else

        Print "Error: " & .GetErrorCode
        Print "Description: " & .GetErrorDesc

    End Select


'Once the report request has completed and all data has been extracted from
'the result file, delete the file from the PCCharge directory.

    .DeleteUserFiles 'Deletes all files associated with the transaction.  If
                     'the integration is event-driven, DeleteUserFiles should
                     'be called in both the FINISH and ERROR events.

     .Clear         'Resets all properties and provides a clean environment to
                    'process the next transaction.

  End if '.PccSysExists check

End With

'IMPORTANT - Destroy the Charge object to reset all properties and methods.
'Also, the Clear method or ClearVariables method can be used to reset all
'properties and methods in the object.

Set Charge = Nothing
```

OCX (ActiveX) Method

Several OCX controls are included with the DevKit. These controls allow developers to access various processing functions using any Windows-based visual programming environment that supports OCX controls.

Before an OCX control can be used, the control must be added to the visual programming language toolbox or equivalent. For example, in Microsoft Visual Basic 6, follow the procedure below:

1. Choose **Project** | **Components** from the Visual Basic menu bar.

2. After the Components window opens, from the list, scroll to and check the box next to the control(s) that will be used:

   - **VeriFone's Charge ActiveX Control** (for Credit card integration)
   - **VeriFone's Debit ActiveX Control** (for Debit card or EBT integration)
   - **VeriFone's Check ActiveX Control** (for Check integration)
   - **VeriFone's GiftCard ActiveX Control** (for Gift/Loyalty integration)
   - **VeriFone's Batch ActiveX Control** (for Batch/Settlement integration)
   - **VeriFone's Device ActiveX Control** (for PINpad integration)
   - **VeriFone's SC550 ActiveX Control** (for VeriFone SC5000 PINpad integration-Canadian debit for Global Payments East NDC)
   - **VeriFone's SC5X ActiveX Control** (for VeriFone SC5000 PINpad integration-Canadian debit for Chase Paymentech GSAR)

   and click **OK**.

   Note: The DevKit installation registers the controls by default. If the controls are not yet registered on the system, use `regsvr32.exe` to register them, or use the Browse button on the Components window to register the component and add it to the toolbar.

3. The control(s) will now be added to the toolbar. Each control can be added to the project by placing it in the desired area on the form.

Each control's properties, methods, and events can be viewed through the object browser. If MS VB6 is not being used, refer to the language documentation for instructions on using ActiveX OCX Controls.

Note to .Net Framework Users

The OCX method is available in .Net through the use of the provided Charge-net.OCX, Debit-net.OCX and Batch-net.OCX. The regular Check.OCX, GiftCard.OCX, Device.OCX can be used with .Net.

Note to Delphi Users

The OCX method is available in Delphi through use of the provided Type Library Files (.tlb) for the corresponding Control. Rather than importing the OCX components to the tool palette, import the Type Libraries and create a wrapper. This can be done by following the procedure below:

1. Select **Project | Import Type Library** from the menu bar. (In Delphi 2005 / 2006, select **Component | Import Component** from the menu bar. Make sure that the radio button for **Import a Type Library** is selected, and then click the **Next** button)

2 A list of Libraries (.dll, .ocx, .olb, .tlb) will be available from which to choose. Click on the **Add** button, and browse to the location of the Type Library Files you wish to import. Once the new Library is highlighted (click the **Next** button in Delphi 2005 / 2006), there will be property fields indicating the classes of the Library, the palette page, the Unit Dir Name, and the Search Path. These fields can be left as default. Make sure that **Create Wrapper** is checked (Delphi 7 and below), and select **Create Unit** (click the **Next** button in Delphi 2005 / 2006 and then **Create Unit**).

3. This will add a text wrapper the Delphi Project. The name of this wrapper will be the name of the Library plus _TLB.pas. (For example: if importing the Charge.tlb, the wrapper name will be Charge_TLB.pas). Save this file to your project's working directory. It then must be added to the Uses clause of the .pas file in which it is intended to be used.

```
unit Unit1;

Interface

Uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, StrUtils, Charge_TLB;
```

4. The next step would be to instantiate the main class of the Library, and load the OCX into memory so that the properties are accessible. One way to dynamically load the OCX into memory is to declare a THandle variable and a variable to hold the class, use LoadLibrary.The following example shows how to load the Charge OCX using this method:

```
Type                              //
  TCharge1 = class(OCXCharge)     // Instantiate the main class of the Library
End;                              //

procedure TForm1.ProcessButtonClick(Sender: TObject);
var
  hCharge: THandle;               // Create a handle for the Library
  Charge: TCharge1;               // Variable to hold the Charge Class
Begin
  hCharge := LoadLibrary('Charge.ocx');
  if (hCharge = 0) then
    Begin
      ShowMessage('Error Loading Library');
      Exit;
    End
  Else
  If (hCharge <> 0) then
    Begin
      Charge := TCharge1.Create(Self);
  End;
```

5. Now all of the Properties for the main class of the Charge OCX will be available.

Note to Visual FoxPro 6 Users

The instruction "`application.autoyield = .F.`" must be executed before the `Send` method can be called for any of the ActiveX controls. A sample Visual FoxPro 6 application is included. It is located in the directory `C:\Program Files\active-charge SDK\Ocx\FoxPro\`.

Note to Visual C++ Users

The standard versions of the OCX controls can be used in a Visual C++ project. The control can be added by going to **Project** | **Components** | **Add Registered Controls**. Visual C++ will provide a wrapper that will give the project access to the various properties and methods in the control. A sample Visual C++ application is included. It is located in the directory `C:\Program Files\active-charge SDK\Ocx\Cpps\`.

Charge.OCX

Charge.OCX provides integrators with properties and methods used to submit credit card transactions to **PC**Charge. To use Charge.OCX to integrate transaction processing, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to process transactions by using the PccSysExists method.

2. Assign the appropriate values to the properties required for the transaction to be performed and validate the values using the various .Verify methods. (The properties marked with a ° in the **Charge.OCX properties** table are the minimum required to process a Sale or Pre-Authorization transaction.) **NOTE:** Beginning with **PC**Charge **version 5.8** the transaction messages may include the CashierName and CashierPassword properties to take advantage of PCI Cashier Changes.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

3. Call the Send method. (**Note:** When calling the Send method, it is recommended that "3" is passed as a parameter to activate the XML message format)

4. Wait for the Error or Finish event to occur.

5. Call the various .Get methods to determine the outcome of the transaction. The most important information can be acquired by calling the GetResult and GetAuth methods. If an error occurs, call the GetErrorCode and GetErrorDesc methods to determine the nature of the error.

6. Call the DeleteUserFiles method to delete all files related to the transaction.

7. Call the Clear method to reset all the properties and methods related to the transaction or destroy the object.

Consult the **Pseudo-code** section (see page 158) for various examples that may be followed when using the Charge.OCX to perform transaction processing.

Note: **PC**Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

Charge.OCX Dependencies

The following dependencies are required for the Charge.ocx to operate properly upon deployment. Most of these files are already installed with the Windows© operating system.

File name	Version	File name	Version
USER32.DLL	5.1.2600.2622	DARTCERTFICATE.DLL	N/A
RPCRT4.DLL	5.1.2600.2180	DARTSECURE2.DLL	N/A
OLEAUT32.DLL	5.1.2600.2180	DARTSOCK.DLL	N/A
OLE32.DLL	5.1.2600.2726	MSXML3.DLL	N/A
NTDLL.DLL	5.1.2600.2180		
MSVCRT.DLL	7.0.2600.2180		
GDI32.DLL	5.1.2600.2818		
MSVBVM60.DLL	6.0.97.82		
KERNEL32.DLL	5.1.2600.2945		
ADVAPI32.DLL	5.1.2600.2180		

Charge.OCX Properties

Property	Data Type	Description - Charge.OCX Properties
Action°	Long	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
Amount°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
AmxChargeDescription	String	The American Express Charge Description. This is a general description describing merchandise: the AMEX representative and the merchant will decide on an appropriate description. **Note:** Only Required for Retail, MOTO and Restaurant transactions when using AMEX direct settlement or TSYS. **Max Length:** 23 bytes
AmxDescription_1	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AmxDescription_2	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AmxDescription_3	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AmxDescription_4	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AuthCode	String	The Authorization code. This value is returned by the issuing bank and should only be set in a transaction request if processing a Post-Authorization and the Post-Authorization is being used to add a Voice-Authorization to the batch or to "store" a Voice-Authorization. (For information on stored Voice-Authorizations, see page 91). The AuthCode property does not need to be set if the Post-Authorization completes a standard Pre-Authorization using the TroutD value of the Pre-Authorization. See the section **Follow On Transactions** for more information (see page 70).
Billpay	String	Only valid for Visa debit and credit transactions. It is used to indicate the transaction is being processedfor payment of a bill (ultilty, monthly gym dues, etc.) **Valid values:** 0 – Non-Bill payment transaction 1 – Bill payment transaction

Property	Data Type	Description - Charge.OCX Properties
Card°	String	The credit card number that will be used when processing the transaction. **Max Length:** 20 characters. **Example:** 5424180279791765 **NOTE:** Is it the integrator's responsibility to remove spaces in the card number if there are spaces in the Track II card number data.
CheckCard	Boolean	Flag that indicates whether to activate credit card validity testing. **Valid Values:** TRUE; FALSE. **Default value:** TRUE. This value must be set to FALSE when performing Follow on transactions such as Voids or Gratuities because the card number is omitted from these transaction requests.
Command	String	The action code that identifies what type of transaction will be performed. **Valid Values:** 1-10, 13-15, ZI, ZH. **Example:** If running a credit card sale, set the action code to "1". Consult the section **DevKit Constants** for a list of valid values (see page 143). **Note:** Because the Action property is defined as "long", this property was added to allow action codes that contain strings (such as Transaction Inquiry - ZI). If the Command property is set, it's value will override the value set in Action.
CommercialCardFlag	String	The type of commercial card being submitted. The getCommercialCardType method should be used to retrieve the 1 character value from **PC**Charge that indicates what type of commercial card will be submitted. See the section **Commercial Card Transactions** (see page 93) for more information. **Max Length**: 1 character **Valid values:** B – Business P,L,G -- Purchase C – Corporate F – Fleet
CommMethod	Enum	Specifies which communication method will be used. 0 – File_Transfer 1 – TCP/IP Please refer to page 20 for a description of these methods. If TCP/IP is selected, the IPAddress, Port and EnableSSL properties must also be set. If File_Transfer is set then the Path property must be set.
CustCode	String	Customer code for purchasing/commercial cards. This property must be set for commercial card transactions in order to get the best discount rate. Additionally, the transaction's action code must indicate that the transaction is a commercial card transaction. **Note:** Global East (NDC), terminal based, requires the customer code be all upper case. **Max Length:** 25 characters, alphanumeric only.
CreditPlanNumber	String	The credit plan number, only applicable when using Citi as the processor for private label cards.
CVV2	String	The CVV2 value for the transaction. The card verification value (CVV2 for Visa, CVC2 for MasterCard, and CID for AMEX and Discover) is a 3 or 4 digit number that is embossed in the signature panel for Visa, MasterCard, and Discover and on the front of the card for AMEX. All AMEX cards utilize a 4 digit CID. **Max Length:** 4 characters. CVV2 should only be passed on non-swiped transactions.
Demo	Boolean	The demo mode flag. In demo mode, a simulated response is returned in which even amounts return approved, and odd amounts return declined. **Valid Values:** TRUE – Activates demo mode FALSE – Deactivates demo mode (default)
DEST_ZIP_CODE	String	Destination Zip Code for American Express purchasing/commercial cards. This property must be set for American Express commercial card transactions when using American Express as the processor (or via split dial) in order to get the best discount rate. Additionally, the transaction's action code must indicate that the transaction is a commercial card transaction.
DriverID	String	Driver identification field. Only required for Wright Express, Voyager and Fleet One cards.
DriverPIN	String	Driver personal identification number. Only required for Fuelman cards.
EnableSSL	Boolean	**For use with TCP/IP CommMethod only**. SSL is available starting with PCCharge version 5.8. Set to True to enable SSL Socket integration. **Note**: See SSL Information on page 70 for more information.

Property	Data Type	Description - Charge.OCX Properties
EstGratuityAmount	String	**For use with Restaurant transactions only.** The estimated gratuity amount for a Sale (action code 1) or Pre-Authorization (action code 4) transaction. If the EstGratuityAmount is populated, **PC**Charge will submit the sum of the values in the Amount and EstGratuityAmount fields for authorization. If the transaction is authorized, only the value in the Amount field will be placed in the **PC**Charge settlement file (if running a Sale). By using the EstGratuityAmount, the merchant can help ensure that the customer has enough available credit on their card to leave a tip. Once the customer indicates the amount of the tip that will be left, a gratuity transaction (action code 13) must be performed on the sale prior to settlement in order to add the actual gratuity to the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. See the section **Restaurant Transactions** (see page 105) for more information. **Note:** It is recommended to check with the processor or merchant service provider for guidance on what amount to set this value to. Incorrectly setting this value can result in downgrades**.**
ExpDate°	String	The expiration date associated with the credit card number that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208
GratuityAmount	String	**For use with Restaurant transactions only.** The actual gratuity amount for a Sale with Gratuity (action code 14) , Gratuity (action code 13) , or Post-Authorization (action code 5) transaction. See the section **Restaurant Transactions** (see page 105) for more information.
IDNumber	String	Only required for Voyager cards, dependant on Restriction Code. Four to six digits. **Note:** Only used for Pre-Authorization transactions **Update:** Beginning with PCCharge version 5.8, this is also used for Citi© Private Label.
IPAddress	String	**For use with TCP/IP CommMethod only.** IPAddress of machine where PCCharge is running. Defaults to 127.0.0.1
ItemID	String	The Item ID for the transaction. This field is only used for Chase Paymentech (GSAR) and can store five (5) four-digit codes that are defined by Chase Paymentech. **Example:** If the ItemID is set to 0001000200030004 0005, it stores 5 item IDs (0001, 0002, 0003, 0004, and 0005). These numbers must be obtained from Chase Paymentech.
LastValidDate	String	The last year that will be considered a valid expiration date. Currently, the default value in the control is "09". It is recommended that a setting is provided by which the end-user can change this property; otherwise, in the future, end users will require a new control to be distributed to resolve expiration date issues. **Length:** 2 digits. **Format:** YY **Example:** If LastValidDate is set to 05, then cards between 06 and 99 are considered to be 1906 to 1999, and cards between 00 and 05 are 2000 to 2005.
Manual°	Long	Flag that indicates whether the transaction was manually entered or swiped. If the transaction was swiped, the Track property must also be set. **Valid values:** 0 = manual transaction, 1 = swiped transaction
MCSC	String	The Multiple Count Sequence Count. This is the total number of installments that will be charged in a non-restaurant recurring billing scenario. **Max Length**: 2 characters. **Example:** If there are 5 payments to be made, set this property to "5".

Property	Data Type	Description - Charge.OCX Properties
MCSN	String	**In a restaurant environment:** The server or cashier id. **Max Length:** 2. This field should be passed for reporting and reconciliation purposes. See the section **Restaurant Transactions** (see page 105) for more information. **Processor specific note:** The Server ID is required for AMEX card transactions. Also required when using the processor NB and GSAR in restaurant business type. **In a non-restaurant environment**, this field is the Multiple Count Sequence Number. This is the transaction number within the total number of payment installments in a recurring billing scenario. **Max Length**: 2 characters. **Example:** If there are 5 payments to be made and this transaction is the first transaction, set this property to "1". The first transaction should also include the CVV property, but this value should **not** be stored or sent for subsequent transactions.
Member	String	The cardholder's name. **Max Length:** 20 characters.
MerchantNumber° ***	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Credit Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
MSMQEncrypted	*Boolean*	*No longer supported*
MSMQOffSite	*Boolean*	*No longer supported*
MSMQServerName	*String*	*No longer supported*
MSMQTransaction	*Boolean*	*No longer supported*
Multi	String	Flag that indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PC**Charge GUI. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid values:** 1 = TRUE, 0 = FALSE **Default value:** 0. See the section **Multi-trans Wait** for more information (see page 67). This Flag has no effect if processing will occur over IP or leased line.
Odometer	String	The odometer reading. Only required for Fleet One (7 digits), Voyager (7 digits), and Fuelman (6 digits) cards.
Offline	String	Flag that indicates whether **PC**Charge should process the transaction offline. If the offline flag is set, **PC**Charge will put the transaction into a .BCH file that resides in the **PC**Charge directory for importing at a later time. The file can only be imported from the **PC**Charge GUI. **Valid values:** 1 = TRUE, 0 = FALSE
OutDelay	Single	The delay time before the **PC**Charge directory is polled for a transaction response file (.oux file). The default is 0.25 seconds. This value should only be modified if the integration is not performing properly. This could be caused if the client machine is slow or there is network lag that causes the server to spend more time checking for .oux files than processing transactions.
Path°	String	**For use with File_Tranfer CommMethod only.** The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ (**default**) **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
PeriodicPayment	String	Flag that indicates whether the transaction is a recurring transaction. **Valid values:** 1 = TRUE, 0 = FALSE **Note:** If periodic payment is set to true, the recurring billing properties must also be set to achieve the best processing rates.
Port	String	**For use with TCP/IP CommMethod only.** Open port of PCCharge.
PrintReceipts	String	The number of receipts that **PC**Charge should print for the transaction. This value will override the corresponding value in the **PC**Charge GUI. **PC**Charge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.

Property	Data Type	Description - Charge.OCX Properties
Processor° ***	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
ProductDetailAmount_XX	String	**Note:** Only required for the processor NBS. This is the total dollar amount for PRODUCT_DETAIL_PRODUCT_CODE_XX being authorized. For example, PRODUCT_DETAIL_PRODUCT_CODE_1 has a PRODUCT_DETAIL_QUANTITY_1 = 2 and a PRODUCT_DETAIL_UNIT_PRICE_1 = $2.00, therefore the PRODUCT_DETAIL_AMOUNT_1 = $4.00
ProductDetailCount	String	**Note:** Only required for the processor NBS. All card types are configurable except for Fleet One which is limited to 7 records. Only 1 – 10 records are currently supported through PCCharge for all card types.
ProductDetailCode_XX	String	**Note:** This is the number of items for RODUCT_DETAIL_PRODUCT_CODE_XX. PCCharge will support 1 – 10. • NBS supports using 1 – 10. • BAMS Tandem (NPC) supports using 1 – 3.
ProductDetailQuantity_XX	String	**Note:** Only used for the processor NBS. This is the unit price for PRODUCT_DETAIL_PRODUCT_CODE_XX. This is only used for Fleet One and Fuelman. Currently, PCCharge will support 1 – 10.
Reference	String	The reference number from the original transaction (returned by the processor). Set this property only if processing a Post-Authorization and the Post-Authorization is being used to add a Voice-Authorization to the batch or to "store" a Voice-Authorization. (For information on stored Voice-Authorizations, see page 91). The Reference property does not need to be set if the Post-Authorization completes a standard Pre-Authorization using the TroutD value of the Pre-Authorization. See the section **Follow On Transactions** for more information (see page 70). **Max Length:** 8 characters. **Note:** NBS/ Fleet One cards require a Reference Number to be sent with each transaction. This is a minimum of 2 digits and a max of 15. This must be all numeric.
RestrictionCode	String	Only required for Voyager cards. This is used to determine the level of identification and which fields are required. Two digits. **Valid Values:** 00 - No ID Number or Odometer required. Fuel and Other allowed. 01 - No ID Number or Odometer required. Fuel only allowed. 10 - ID Number only required. Fuel and Other allowed. 11 - ID Number only required. Fuel only allowed. 20 - Odometer only required. Fuel and Other allowed. 21 - Odometer only required. Fuel only allowed. 30 - ID Number and Odometer required. Fuel and Other allowed. 31 - ID Number and Odometer required. Fuel only allowed. **Note:** Required for both manual and swiped transactions.
RFID	String	Set to 1 if card information was read from RFID (Radio Frequency Identification) device. If card was read from from RFID, track data must be populated and manual flag must be set to 1. Set to 0 otherwise.
Store	String	Flag indicating whether a Voice Authorization transaction should be stored. This flag should only be submitted when performing a Post-Authorization transaction (action code 5) that includes an authorization code from the voice operator. For more information on stored Voice-Authorizations, see page 91. **Valid Value:** 1 - Store the Voice Authorization transaction.
Street	String	The cardholder's billing street address. The Street property is used for address verification. Address verification can only be performed on non-swiped transactions. **For FDC:** Use first 5 digits only. **Note:** For manually keyed transactions, Street is required to qualify for the lowest transaction rates. **Max Length:** 20 characters **Citi**©- When used with Citi Plan PO Box's are not allowed.
TaxAmt	String	The tax amount. This is the portion of the amount that is tax. Providing the tax amount is required to obtain the best rate on commercial card transactions. **Max Length**: 9 characters (including the decimal). **Format:** DDDDDD.CC. The transaction's action code must indicate that it is a commercial transaction. Tax amount should be included in the amount field.

Property	Data Type	Description - Charge.OCX Properties
TaxExempt	String	Tax Exempt Flag. This flag is used to indicate if the purchase is tax exempt. Used only for Commercial Card Transactions. **Valid Values:** 1 – Purchase is tax exempt; 0 – Purchase is not tax exempt.
Ticket	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** For manually keyed transactions, Ticket is required to qualify for the lowest transaction rates. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.
TimeOut	Long	The number of seconds after which a timeout error will be returned from the control. The count will start when the Send method is called. The default timeout value is 90 seconds. It is highly recommended that integrators review the section **Timeouts** (see page 59). Setting the TimeOut value improperly could cause reconciliation issues and problems such as double-charging a customer's account.
TotalAmount	*String*	*No longer needed*
Track	String	The track II data captured from the magnetic strip of the credit card. The track II data is required to ensure the lowest per-transaction rate from the processing company when performing swiped transactions (Retail and Restaurant). Sending the track II data is not allowed if the merchant's industry type is MOTO or eCommerce. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in. Alternatively, the GetParseData method can be used to parse the track data and set the Card, ExpDate, and Track properties automatically.
TroutD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
User° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).
VehicleID	String	Only required for Wright Express cards (5 digits) and Voyager cards (8 digits). **Note:** Required for both manual and swiped transactions.
XMLtran	*Boolean*	*Set to True to activate the XML message format. It is recommended that the "3" parameter be passed to the Send method to activate the XML message format instead of using the XMLtran property. See the description for the Send method for more information.*
Zip	String	The cardholder's zip code. The Zip property is used for address verification. **Max Length:** 9 digits. Address verification can only be performed on non-swiped transactions. **Note:** For manually keyed transactions, the Zip is required to qualify for the lowest transaction rates. **Note:** For manually keyed transactions, the Zip is required to qualify for the lowest transaction rates. **Note:** If submitting the 9-digit zip, do not include the dash. **Citi©** - When used with Citi Plan the zip is limited to 5-digits.
CfgType	Long	The database archive configuration type setup in PCCharge. Valid value: Currently, only 0 is supported. 0 = CFG_TXN_ARCHIVE = Configure Transaction Archive. Use action code ZC.
CfgEnabled	Boolean	Enable or disable current database archive configuration (1 = Enable, 0 = Disable).
CfgPath	String	Specify path for saved output files (Example: backed up transaction database). Must end with a backslash "\".
CfgSizeLimit	String	Transaction archive size limit for GUI archive prompting and validation. Specified in megabytes.

Property	Data Type	Description - Charge.OCX Properties
CfgKeepDays	String	Transaction archive preservation range. All transactions within the past number of "keep days" will remain in the pccw.mdb database following a transaction archive command.
CashierName°°	String	Submit Cashier's login name.
CashierPassword°°	String	Submit Cashier's password.
NewCashierPassword	String	Submit a new password for the Target Cashier. **Note:** case-sensitive Must be complex: • minimum 7 characters • must have at least one upper case character • one numeric character • one special character (e.g., @, $, %, etc.) **NOTE:** See **Cashier Permissions** on page 75 for an example.
NewCashierConfirmPassword	String	Resubmit the password for verification. **Note**: Must match the original NewCashierPassword **NOTE:** See **Cashier Permissions** on page 75 for an example.
AddCashier1	String	Allows the integrator to submit new cashiers up to 5 at a time. Requires 3 Name Value pairs to be entered for each AddCashier tag submitted. **CASHIER_NAME <Name>** **PASSWORD <Password>** **PERMISSIONS <Permissions>** Example: <ADD_CASHIER_1>CASHIER_NAME CreditR PASSWORD CreditR123% PERMISSIONS 10000|000|000|000|000000000000|0000|000000 </ADD_CASHIER_1> **NOTE:** See **Cashier Permissions** on page 75 for an example.
AddCashier2	String	
AddCashier3	String	
AddCashier4	String	
AddCashier5	String	
TargetCashierName	String	Submitted when performing a function to edit a cashier's settings. This specifies which cashier's information to alter. **NOTE:** See **Cashier Permissions** on page 75 for an example.
CustomerFirstName [1]	String	Applicant's first name.
MiddleInitial [1]	String	Applicant's middle initial.
CustomerLastName [1]	String	Applicant's last name.
Suffix [1]	String	Applicant's suffix. (Ex. M.D., Ph.D, Jr.)
AptSuite [1]	String	Applicant's apartment or suite number.
CustomerCity [1]	String	Applicant's city.
State [1]	String	Applicant data. (Ex: GA, FL, MA…)
Email [1]	String	Applicant data. (Ex: XXX@XX.XXX)
Phone_Number [1]	String	Applicant data. Format xxxxxxxxxx 10-digits No Dashes
SSNum [1]	String	Applicant data. Format xxxxxxxxx 9-digits No Dashes
Birth_Date [1]	String	Applicant data. Format is MMDDYYYY.
EmpName [1]	String	Applicant data. Employer name.
WorkPhoneNumber [1]	String	Applicant data. Employer phone number.
HomeOwner [1]	String	Applicant data. Format: '**O**' = Own '**R**' = Rent '**B**' = Board '**P**' = Live w/ parents '**M**' = Military
AnnualIncome [1]	String	Applicant data. Format: Up to 6 digits. Whole dollar amount with no decimals.
PhotoIDState [1]	String	Applicant data. (Ex: GA, FL, MA…)
CorrelationUID [1]	String	Received as a response to a Credit App (P3)
PendingNumber [1]	String	Received as a response to a Credit App (P3)

Property	Data Type	Description - Charge.OCX Properties
ReplyFlag [1]	Boolean	Reply notification. Format: **Y** = Accept **N** = Decline
SourceCode [1]	String	**Citi Plan** Format: Opt In – **"AP"** Opt Out – **"AO"**
IDType [1]	String	Applicant data. Type of identification being submitted. Format: **'D'** – Driver's License **'O'** – Other
FraudFlag [1]	Boolean	Code 10. Values are **1** or **0: 1** = True**, 0** = False
HealthCareAmount [#]	String	Total Healthcare Amount. This amount must be greater than or equal to the sum of the other amount categories. Max: **12 - digits** Format: **DDD.CC**
PrescriptionAmount [#]	String	(Optional) Total amount of the prescription-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
VisionAmount [#]	String	(Optional) Total amount of the vision-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
ClinicAmount [#]	String	(Optional) Total amount of the clinic-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
DentalAmount [#]	String	(Optional) Total amount of the dental-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
TransitAmount [#]	String	(Optional) Reserved for future use. Max: **12 - digits** Format: **DDD.CC**
CopayAmount [#]	String	(Optional) Reserved for future use. Max: **12 - digits** Format: **DDD.CC**
FSA [#]	Boolean	Indicates the transaction is FSA. This will bypass PCCharge BIN checking as PCCharge will assume that the POS has verified that this card has met the BIN requirements for an FSA transaction. **1** = True, **0** = False **Note:** If this is not passed it defaults to False.

° These properties are the minimum required to process a Sale or Pre-Authorization transaction.
°° These properties are additional requirements beginning in **PC**Charge version 5.8 and are in addition to the preceeding requirements for a Sale or Pre-Authorization transaction. These are only necessary if taking advantage of the new PCI Cashier Changes.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

*** If the "Use Default Processor" option is enabled in the **PC**Charge preferences, and the `Processor` and `MerchantNumber` properties are omitted from the transaction request, **PC**Charge will process all transactions using the "Default Processor". The "Default Processor" is defined as the first merchant number that is set up **PC**Charge. Consult the **Multi-Merchant Support** section (see page 68) for more information on the "Use Default Processor" option. In addition, `Processor` and `MerchantNumber` should not be set when doing follow-on transactions. Refer to the section **Follow On Transactions** (see page 70) for more information.

[1] These properties are required to process **Citi applications**. For more information and examples see Citi Credit Plan on page 116.

[#] Additional tags used to process FSA/HRA transactions. For more information please see the section about **FSA on page 99**.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

Charge.OCX Methods

Method	Returned Value	Description - Charge.OCX Methods
Abort	Boolean	The Abort method attempts to cancel the transaction in progress and will return a Boolean value that indicates whether or not the transaction was canceled. **Note:** This method is not available when integrating using FoxPro. Use the Cancel method instead.
About	MsgBox	The About method will display the About box associated with the control.
Cancel	None	The Cancel method attempts to cancel the transaction in progress. Calling the Cancel method does not guarantee that the transaction will be canceled; it simply attempts to cancel the transaction.
Clear	None	The Clear method will clear the values in all properties and methods. It is recommended that this method be called: a. after the transaction results have been retrieved by using the various .get methods b. after the DeleteUserFiles method has been called c. prior to running the next transaction
CommercialCardType	String	The CommercialCardType method is used to determine whether or not a credit card is a commercial card. CommercialCardType requires that a string parameter, the credit card number, is passed when calling the method, that the Path property is set to a valid **PC**Charge directory, and that a valid Bin.mdb database resides in that directory. CommercialCardType returns TRUE if the BIN range of the card appears in the Bin.mdb database, FALSE if it does not.
DeleteUserFiles	None	The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files, the Error event will be triggered and the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150). *Not for use with TCP/IP CommMethod.
GetAcctDataSrc	String	Returns the entry method of the transaction.
GetACI	String	Returns the VPS indicator to indicate wherever the card is a VISA, MC or AMEX card PS2000 data. This value is not returned by all processing companies.
GetAddText1	String	Only supported on Fleet One, this field contains miscellaneous additional text returned from host. Currently PCCharge will support GetAddText1-GetAddText4.
GetApproved	Boolean	The GetApproved method returns TRUE if **PC**Charge returns "APPROVED" as the result of the transaction. Otherwise, FALSE will be returned. An "APPROVED" response indicates that a Pre-Authorization has been approved, but not placed in the open batch.
GetAmountDue	String	Returns the amount due, only for Elavon (NOVA) pre-paid and FSA functionality. **Format is DDD.CC** **Note:** "Enable Pre-Paid Cards" option within PCCharge MUST be checked for this feature to work. The option is located under Setup>Credit Card Company>Extended Data and only with the Credit Card Company set to Elavon (NOVA). This option is only configurable in *PCCharge version 5.7.1 release I sp9a* and above. **Note:** For this to work with FSA transactions, one must enable FSA and Partial Auth within the Extended Data Screen for supported processors.
GetAuth	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
GetAuthAmount	String	Used in partial redemption transactions where only part of the amount was authorized. Returns the actual authorized amount.

Method	Returned Value	Description - Charge.OCX Methods
GetAVS	String	Returns the AVS response code from the issuing bank. If performing Address Verification on card-not-present transactions, this code indicates how well the AVS information passed in matches what the issuing bank has on file for the cardholder. Consult the section **DevKit Constants** for a description of values that may be returned (see page 143).
GetCCAvailBalance	String	**Pre-paid credit cards with Elavon (NOVA) -** Returns the PrePaid card balance. **FSA -** Returns the available balance on the FSA/HRA card. **Format: DDD.CC**
GetCaptured	Boolean	The GetCaptured method returns TRUE if **PC**Charge returns "CAPTURED" as the result of the transaction. Otherwise, FALSE will be returned. The GetCaptured method is used to determine if a transaction that will result in a monetary transfer (Sale, Credit, Post-Authorization, etc.) is approved or declined. A "CAPTURED" response indicates that the transaction has been approved, and that the transaction has been placed in the open batch.
GetCardIDCode	Boolean	Returns a code that is used to verify the identity of the cardholder.
getCommercialCardType	String	The getCommercialCardType method requires a string parameter, the credit card number, and will determine the credit card number's commercial card type. This method requires that the Path variable be set to a valid **PC**Charge directory and it uses the Bin.mdb database in the **PC**Charge directory to determine the commercial card type. **Valid return values:** B – Business P,L,G -- Purchase C – Corporate F – Fleet **Example:** getCommercialCardType("4055011111111111") will return "P".
GetCreditCardIssuer	String	The GetCreditCardIssuer method returns the abbreviation of the credit card issuer's name for the card number that is present in the Card property. Consult the section **DevKit Constants** for a description of values (see page 143).
GetCreditCardType	String	The GetCreditCardType method returns either the abbreviation of the credit card issuer of the card that is present in the Card property, or the optional card parameter that is passed to the GetCreditCardType method. Consult the section **DevKit Constants** for descriptions of values (see page 143). (GetCreditCardType is the same as GetCardIssuer).
GetCurrentDBSize	String	Current transaction database size in bytes.
GetConfigDBSize	String	Current configured size limit for transaction archive in bytes.
GetCVV2	String	Returns the CVV2/CVC2/CID response code from the issuing bank. If performing CVV2/CVC2/CID validation on card-not-present transactions, this code indicates if the CVV2/CVC2/CID code passed in matches what the issuing bank has on file for the cardholder. Consult the section **DevKit Constants** for a description of values that may be returned (see page 143).
GetDCAvailBalance	String	Returns the available balance on pre-paid debit cards. Only for pre-paid debit cards with Elavon (NOVA).
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetEstGratuityAmount	String	The GetEstGratuityAmount echoes the estimated gratuity from the original transaction.
GetGratuityAmount	String	The GetGratuityAmount echoes the gratuity amount from the original transaction.

Method	Returned Value	Description - Charge.OCX Methods
GetHostType	Integer	The GetHostType method returns an integer that indicates if a processor / merchant number is Host based or Terminal based. GetHostType requires three parameters: 1) **Processor code** - Consult the section **DevKit Constants** (see page 143) for a list of valid processor codes 2) **Merchant account** - Must be a valid merchant account set up in **PC**Charge 3) **TID type** - Valid Values for TID type: 0 – Credit; 1 – Check; 2 – Debit; 3 – EBT; 4 – GiftCard GetHostType will return one the following values based on the parameters passed in: 0 – The processor is Host based 1 – The processor is Terminal based -1 – The processor is a Hybrid (supports both Host and Terminal processing) or invalid processor / merchant number. **Example:** .GetHostType("VISA", "999999999911", 0) will return 0 **Note:** Chase Paymentech (GSAR), Elavon (NOVA), and FDMS South / NaBanco (NB) are considered hybrid processors. GetHostType will return a -1 for these processors.
GetIND	String	Returns the IND code. The IND code is a transaction description code and an Interchange compliance field. This value is not returned by all processing companies.
GetItemID	String	The GetItemID echoes the item ID from the original transaction.
GetMerchantInfo	String	The GetMerchantInfo method returns a string containing all of the merchant numbers and processors set up in **PC**Charge. The string will also indicate whether the processor is Host based (H), Terminal based (T), or a hybrid (Y). The string will begin with STX and will end with ETX. GS will separate each record, and FS will separate fields within a record. **Example:** <STX>CES <FS>0000009279962967<FS>T<GS>GSAR<FS> 999999999999519<FS>T<GS>VISA<FS>999999999911<FS>T<ETX> Refer to the section **Multi-Merchant Support** (see page 68) for more information on the GetMerchantInfo method.
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetMSI	String	Returns the Market Specific Indicator. This value indicates the transaction's market segment. This value is assigned by the card associations and is not returned with all transactions.
GetParseData	String	The GetParseData method will parse a string (containing credit card track data) passed to it and populate the Card, ExpDate, Member, and Track properties with the appropriate data. GetParseData will return an integer indicating its success. **Valid return values:** 0 (error parsing data), 1 (track I successful), or 2 (track I & II successful).
GetPLProcessor	String	Retrieves the private label Processor ID currently setup in PCCharge. The .path property must be specified.
GetPLMerchantNumber	String	Retrieves the private label Merchant Number currently setup in PCCharge. The .path property must be specified.
GetPCard	String	The GetPCard method returns the PurchaseCard field from the .oux file. Not all processing companies support this field. **Valid values:** 1 = Purchasing card, 0 = Otherwise
GetPEM	String	Returns the Point of Entry Mode that is associated with the transaction. This value is not returned by all processing companies.
GetPS2000	String	PS2000 Data. This data will be as received during the original authorization processing. It will not be present for offline transactions. PS2000 Data is a variable; it will either be one character or up to 20. It is data concerning the card type and transaction that the processor will send back during the authorization process. This value is not returned by all processing companies.
GetRecordCount	String	The number of records matching the inquiry

Method	Returned Value	Description - Charge.OCX Methods
GetRefNumber	String	Returns the reference number associated with the transaction. The reference number is assigned by the card associations. The reference number is used to help identify the transaction and is useful for the cardholder and merchant when doing research. This value is not returned with all transactions.
GetRespCode	String	Returns the response code that is provided by the processor. This value is not returned by all processing companies.
GetResponseCommercialType	String	Returns the type of commercial card that was used for the transaction. This value is not returned by all processing companies.
GetResponsePurchaseCardType	String	Returns a flag indicating whether the processor indicated whether the card was a Purchasing Card or not. This value is not returned by all processing companies. **Valid values:** 1 = Purchasing Card, 0 = Otherwise
GetRestrictCode	String	**Note:** Only supported on Fleet One. The product restriction code.
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
GetRET	String	Returns the Retrieval reference number. This value is not returned by all processing companies
GetTraceNumber	String	Returns the trace number from the processor. Only for pre-paid credit cards with Elavon (NOVA).
GetTBatch	String	Returns the active batch number for the transaction. This value is not returned by all processing companies.
GetTerminationStatus	String	Retrieves Termination Status which returns a "6" upon successful transaction. If anything other than a "6" is returned there may be an issue with the settlement file and should be investigated. This applies to : **GSAR** and **NBS** **Response:** **6 = Successful Transaction**
GetTDate	String	Returns the date that the transaction was processed. This value is not returned by all processing companies.
GetTI	String	This will indicate the transaction Identifier for VISA or AMEX, it will also return the MC Bank Net reference if the card is a MasterCard. This value is not returned by all processing companies.
GetTicket	String	Returns the ticket number or invoice of the transaction. This value is echoed back from the original transaction or is generated by **PC**Charge if one is required to complete the transaction.
GetTICode	String	Returns the validation code for VISA or the Bank Net Date if the card is a MasterCard**.**This value is not returned by all processing companies.
GetTIM	String	Returns the Time of the transaction. This value is not returned by all processing companies.
GetTitem	String	Returns the Transaction Item number or the number that is associated with the transaction in the settlement file. This value is not returned by all processing companies.
GetTransNum	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
GetTransRecord	String	Contains nested XML tags providing information on transaction(s) pulled from Trans table in the PCCharge database (pccw.mdb)
GetTransactionReferenceNumber	String	Returns the transaction reference number from the processor. Only for pre-paid credit cards with Elavon (NOVA).
GetTroutD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.

Method	Returned Value	Description - Charge.OCX Methods
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved from the .oux file can be used to view the results of a Transaction Inquiry (ZI) transaction. Refer to the section **Transaction Inquiry** (see page 128) for more information. The text can also be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.
GetXMLRequest	String	This method is used to echo the text that is sent in the request file associated with the transaction. The request (.inx) file contains XML string data. The text that is sent in the .inx file can be used to view the message of any transaction sent to the server. Note: This method must be called after the calling send and before DeleteUserFiles method.
PccSysExists	Boolean	The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions. *Not for use with TCP/IP CommMethod.
Send	Integer	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the control will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has an optional parameter that indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send 3 **Note:** The other values that appear in the enumerated list are for internal use only—do not attempt to use any values other than the ones listed above.
ValidCardLength	Boolean	The ValidCardLength method returns TRUE if the card is of the correct length, or FALSE if it is not. ValidCardLength has an optional string parameter in which a card can be passed. If that parameter is blank, ValidCardLength will analyze the value set in the Card property.
ValidDate	Boolean	The ValidDate method returns TRUE if the expiration date provided in the ExpDate property is valid, or FALSE if it is not.
VerifyAmount	Boolean	The VerifyAmount method returns TRUE if the amount provided in the Amount property is in a valid format (DDDDDD.CC), or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).

Method	Returned Value	Description - Charge.OCX Methods
VerifyCreditCard	Boolean	The VerifyCreditCard method returns TRUE if the credit card number's format is valid and meets the requirements set forth by the credit card companies, FALSE if it does not. If FALSE is returned, use the GetErrorCode and GetErrorDesc methods to determine the reason for failure. VerifyCreditCard has an optional string parameter in which a credit card number can be passed. If the parameter is left blank, VerifyCreditCard will analyze the value set in the Card property.
VerifyExpDate	Boolean	The VerifyExpDate method returns TRUE if the expiration date provided in the ExpDate property is correct and in the right format, or FALSE if it is not. VerifyExpDate calls the ValidDate function to validate the expiration date. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
VerifyMerchantNumber	Boolean	The VerifyMerchantNumber method returns TRUE if the merchant number that is passed to it is set up in **PC**Charge, otherwise, FALSE is returned. Specifically, this method checks for the merchant number in the file TID.PCC, which is located in the **PC**Charge directory. The Path property must be set before calling this Method.
VerifyProcessor	Boolean	*No longer supported*
SetSSLCertificate	None	User can set the local certificate to authenticate the SSL TCP/IP request. **Note:** This routine has four parameters. CertificateName : this will store the Store name of the store CertificateLocation : this will assign the store location CertificateIssuesby : This will assign Certificate issued by CertificateSerialNumber : This will assign certificate serial number **Note**: See SSL Information on page 70 for more information.
GetPasswordExpiry	String	Returns MMDDYY when the cashier password will expire.
GetCorrelationUID [1]	String	Returned from Citi, needed for Counter Offer Submission.
GetPendingNumber [1]	String	Returned from Citi, needed for Counter Offer Submission.
GetReplyFlag [1]	String	**'Y'** or **'N'** sent to Citi to notify message is a reply to an offer.
GetOpenToBuy [1]	String	Only present in the response if the "Display OTB" is turned ON. Decimal is included.
GetCreditLimit [1]	String	Only present the in the response. Retrieves credit limit.

[1] These properties are required to process Citi applications. For more information and examples see Citi Credit Plan on page 116.

Charge.OCX Events

Event	Description - Charge.OCX Events
Error	The Error event is fired any time an error occurs in the control. Once an Error event has fired, call GetErrorCode and GetErrorDesc to determine what kind of error has occurred. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
Finish	The Finish event will fire when the transaction has been completed. This means that **PC**Charge has processed the transaction successfully and has placed a file with the extension of .oux in the **PC**Charge directory. The name of the .oux file will be what was set in the User property of the transaction request. Call the GetResult method to determine whether or not the transaction was approved. A list of valid results can be found in the **Transaction Result Constants** section (see page 156).

Note: When doing asynchronous transactions in an event-driven programming model, it is important to place all result or error routines in either the Finish or Error events. Do not place any code that uses the .get methods after invoking the Send method.

Debit.OCX

Debit.OCX provides integrators with properties and methods used to submit debit card and EBT transactions to **PC**Charge. To use Debit.OCX to integrate transaction processing, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to process transactions by using the PccSysExists method.

2. Retrieve pertinent data, such as the PIN, the Key Serial Number (if DUKPT), etc., from the PINpad. Device.OCX may be used to collect data from the PINpad. See page 242 for more information on Device.OCX.

3. Assign the appropriate values to the properties required for the transaction to be performed and validate the values using the various .Verify methods. (The properties marked with a ° in the **Debit.OCX properties** table are the minimum required to process a Debit Sale transaction.)

4. Call the Send method. (**Note:** When calling the Send method, it is recommended that "3" is passed as a parameter to activate the XML message format)

5. Wait for the Error or Finish event to occur.

6. Call the various Get methods to determine the outcome of the transaction. The most important information can be acquired by calling the GetResult and GetAuth methods. If an error occurs, call the GetErrorCode and GetErrorDesc methods to determine the nature of the error.

7. Call the DeleteUserFiles method to delete all files related to the transaction.

8. Call the Clear method to reset all the properties and methods related to the transaction or destroy the object.

When processing debit cards, a PINpad is required to allow the customer to enter their PIN. In addition, debit card information is always collected via a card swipe device, never via keyboard entry. Because of this, a card reader is also required. (Some EBT transactions can be manually entered).

Refer to the **Device.OCX** section (see page 242) for information on integrating to a PINpad device. Device.OCX gives the application the ability to retrieve the PIN from the PINpad and use it to process debit card transactions.

When processing U.S. debit card transactions, merchants have the option of allowing the customer to receive cash back on a transaction. For instance, the customer purchases $50 of products and wants $25 cash back, set the Amount to 50.00 and CashBack to 25.00. This will withdraw a total of $75 from the debit card account, $50 for the products and $25 for cash to give to the customer.

Consult the **Pseudo-code** section (see page 158) for an example that may be followed when using the Debit.OCX to perform transaction processing.

For information on integrating Canadian Debit, see the section **Canadian (Interac) Debit Transactions** (see page 113).

Note: PC Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

Debit.OCX Properties

Property Name	Data Type	Description – Debit.OCX Properties
Action° °°	Long	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
Amount° °°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
AuthCode	String	**EBT Only**: For an EBT Post (Prior Auth Sale) or Force transaction: The Authorization code from the original voice authorization.
Billpay	String	Only valid for Visa debit and credit transactions. It is used to indicate the transaction is being processed for payment of a bill (ultilty, monthly gym dues, etc.) **Valid values:** 0 – Non-Bill payment transaction 1 – Bill payment transaction
Card° °°	String	The Debit/EBT card number that will be used when processing the transaction. **Max Length:** 20 characters. **Example:** 5424180279791765 **NOTE:** Is it the integrator's responsibility to remove spaces in the card number if there are spaces in the Track II card number data.
CashBack	String	The amount of cash back that the customer will receive. This amount is in addition to value entered in Amount property. For example, if the total amount of the sale is $10 and the customer has requested $5 cash back, Amount should be set to $10 and CashBack should be set to $5. The CashBack property should be formatted the same the Amount property. **Max Length:** 9 characters. **Note:** Some debit processors do not support the cash back feature.
Command	String	Because the Action property is defined as "long", this property was added to allow action codes that contain strings. This property is only used for action code M1 (Key Change Request).
CommMethod	Enum	Specifies which communication method will be used. 0 – File_Transfer 1 – TCP/IP Please refer to page 20 for a description of these methods. If TCP/IP is selected, the IPAddress, Port and EnableSSL properties must also be set. If File_Transfer is set then the Path property must be set.
DebitType°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Set this to the bank account type that the customer specified when entering transaction data into the PINpad. **Valid Values:** "Chequing" or "Savings"
EnableSSL	Boolean	**For use with TCP/IP CommMethod only**. SSL is available starting with PCCharge version 5.8. Set to True to enable SSL Socket integration. **Note**: See SSL Information on page 70 for more information.
ExpDate° °°	String	The expiration date associated with the Debit/EBT card number that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208 Set this property if there is an expiration date associated with the Debit/EBT card.
FoodStamp	Boolean	**EBT Only:** Indicates what type of EBT transaction will be performed. **Valid Values:** 1 – Food stamp transaction; 0 – Cash benefits transaction
Gratuity°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. This is the Gratuity Amount of the transaction.

Property Name	Data Type	Description - Debit.OCX Properties
IPAddress	String	**For use with TCP/IP CommMethod only.** IPAddress of machine where PCCharge is running. Defaults to 127.0.0.1
KeySerialNumber°°	String	If a Key Serial Number is returned from the PINpad, this property should be populated with that number. **If processing transactions with a PINpad using DUKPT encryption**, this value is sixteen or twenty characters long (depending on the processor's encryption). The **PC**Charge DevKit provides several tools for retrieving data from PINpads. If the **PC**Charge integration method chosen doesn't support these tools or the tools do not support the PINpad being used, a direct interface to the PINpad must be written by the integrator. **If processing transactions with a Verifone SC5000 PINpad**, set this property to the Chip Serial Number of the PINpad.
LanguageCode°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Set this to the language that is indicated by the Language Code that is encoded in the track II data on the customer's card. **Valid Values:** "English" or "French" (pass in the literal string)
MACData°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Set this to the MAC Block value returned by the PINpad.
Manual° °°	Boolean	Flag that indicates whether the transaction was swiped or manually entered. This property must be set to 1 (swiped) for Debit transactions or swiped EBT transactions. If the transaction was swiped, the Track property must also be set. If performing a manually keyed EBT transaction, such as a Force or Voucher, set this property to 0 (manually entered).
Member	String	The cardholder's name. **Max Length:** 20 characters.
MerchantNumber° °° ***	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Debit Card Setup** window or **EBT Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
OrigPurchData	String	The Original Purchase Data. Used when performing a Debit Return with the processors TSYS, Heartland, RBS WorldPay, and NPC. This is the original transaction date. **Format:** DDMMhhmm
OutDelay	Single	The delay time before the **PC**Charge directory is polled for a transaction response file (.oux file). The default is 0.25 seconds. This value should only be modified if the integration is not performing properly. This could be caused if the client machine is slow or there is network lag that causes the server to spend more time checking for .oux files than processing transactions.
Path° °°	String	**For use with File_Tranfer CommMethod only.** The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ (**default**) **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Pin° °°	String	The encrypted PIN block that is retrieved from the PINpad. The PIN is provided to the processor for verification. **Length:** 16 characters. The **PC**Charge DevKit provides several tools for retrieving data from PINpads. If the **PC**Charge integration method chosen doesn't support these tools or the tools do not support the PINpad being used, a direct interface to the PINpad must be written by the integrator.
Port	String	**For use with TCP/IP CommMethod only.** Open port of PCCharge.
PrintReceipts	String	The number of receipts that **PC**Charge should print for the transaction. This value will override the corresponding value in the **PC**Charge GUI. **PC**Charge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.

Property Name	Data Type	Description - Debit.OCX Properties
Processor° °° ***	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
ProductDetailAmount_XX	String	**Note:** Only required for the processor NBS. This is the total dollar amount for PRODUCT_DETAIL_PRODUCT_CODE_XX being authorized. For example, PRODUCT_DETAIL_PRODUCT_CODE_1 has a PRODUCT_DETAIL_QUANTITY_1 = 2 and a PRODUCT_DETAIL_UNIT_PRICE_1 = $2.00, therefore the PRODUCT_DETAIL_AMOUNT_1 = $4.00
ProductDetailCount	String	**Note:** Only required for the processor NBS. All card types are configurable except for Fleet One which is limited to 7 records. Only 1 – 10 records are currently supported through PCCharge for all card types.
ProductDetailCode_XX	String	**Note:** This is the number of items for RODUCT_DETAIL_PRODUCT_CODE_XX. PCCharge will support 1 – 10. • NBS supports using 1 – 10. • BAMS Tandem (NPC) supports using 1 – 3.
ProductDetailQuantity_XX	String	**Note:** Only used for the processor NBS. This is the unit price for PRODUCT_DETAIL_PRODUCT_CODE_XX. This is only used for Fleet One and Fuelman. Currently, PCCharge will support 1 – 10.
Reference	String	Required by some processors for returns. The reference number is returned with the original transaction. **Note:** NBS/ Fleet One cards require a Reference Number to be sent with each transaction. This is a minimum of 2 digits and a max of 15. This must be all numeric.
RFID	String	Set to 1 if card information was read from RFID (Radio Frequency Identification) device. If card was read from from RFID, track data must be populated and manual flag must be set to 1. Set to 0 otherwise.
ShiftID	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. The Shift ID. This value is optional. **Format:** Alphanumeric **Max Length:** 1 character.
Ticket	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.
TimeOut	Long	The number of seconds after which a timeout error will be returned from the control. The count will start when the Send method is called. The default timeout value is 90 seconds. It is highly recommended that integrators review the section **Timeouts** (see page 59). Setting the TimeOut value improperly could cause reconciliation issues and problems such as double-charging a customer's account.
Track° °°	String	The track II data captured from the magnetic strip of the card. The track II data is required for Debit transactions and for swiped EBT transactions. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in.
TransNum	*String*	*No longer needed*
TroutD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
User°° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).
Voucher	String	**EBT Only:** The voucher number for an EBT force transaction. The voucher is provided by the processor at the time of authorization and must be supplied to clear the voucher.

Property Name	Data Type	Description - Debit.OCX Properties
XMLtran	Boolean	*Set to True to activate the XML message format. It is recommended that the "3" parameter be passed to the Send method to activate the XML message format instead of using the XMLtran property. See the description for the Send method for more information.*
CashierName	String	Submit Cashier's login name.
CashierPassword	String	Submit Cashier's password.

° These properties are required to process a Debit Sale transaction.

°° These properties are required to process a Canadian Debit Sale transaction using Global Payments East (NDC) and the SC5000 PINpad.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

*** If the "Use Default Processor" option is enabled in the **PC**Charge preferences, and the Processor and MerchantNumber properties are omitted from the transaction request, **PC**Charge will process all transactions using the "Default Processor". The "Default Processor" is defined as the first merchant number that is set up **PC**Charge. Consult the **Multi-Merchant Support** section (see page 68) for more information on the "Use Default Processor" option. In addition, Processor and MerchantNumber should not be set when doing follow-on transactions. Refer to the section **Follow On Transactions** (see page 70) for more information.

Debit.OCX Methods

Method Name	Returned Value	Description - Debit.OCX Methods
About	Boolean	The About method will display the About box associated with the control.
Cancel	None	The Cancel method attempts to cancel the transaction in progress. Calling the Cancel method does not guarantee that the transaction will be canceled; it simply attempts to cancel the transaction.
Clear	None	The Clear method will clear the values in all properties and methods. It is recommended that this method be called: a. after the transaction results have been retrieved by using the various .get methods b. after the DeleteUserFiles method has been called c. prior to running the next transaction
Connect	*Boolean*	*No longer supported*
DeleteUserFiles	None	The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files, the Error event will be triggered and the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150). For use with file transfer CommMethod only
Disconnect	*None*	*No longer supported*
GetApproved	Boolean	The GetApproved method returns TRUE if **PC**Charge returns "APPROVED" as the result of the transaction. Otherwise, FALSE will be returned.
GetAuth	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
GetAuxRespCode	String	When using the SC5000 PINpad, returns the ISO response code
GetAvlBalance	String	**EBT Only:** The GetAvlBalance method returns the available balance on the EBT card. This value is not returned by all processing companies.

Method Name	Returned Value	Description - Debit.OCX Methods
GetCaptured	Boolean	The GetCaptured method returns TRUE if **PC**Charge returns "CAPTURED" as the result of the transaction. Otherwise, FALSE will be returned. The GetCaptured method is used to determine if a transaction that will result in a monetary transfer (Sale, Credit, Post-Authorization, etc.) is approved or declined. A "CAPTURED" response indicates that the transaction has been approved, and that the transaction has been placed in the open batch.
GetEBTCashBalance	String	**EBT Only:** Returns the remaining balance on a Cash Benefits card. This value is not returned by all processing companies.
GetEBTFoodBalance	String	**EBT Only:** Returns the remaining balance on a Food Stamp card. This value is not returned by all processing companies.
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetMSI	String	**For Debit Master Session:** Returns the new master key (if one exists) sent by the processor that should be passed to the PINpad.
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetPOSSequenceNumber	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Returns the current POS Sequence Number for the PINpad. The Path property must be set to the **PC**Charge directory and the PINpad's Chip Serial Number must be passed as a parameter when calling the GetPOSSequenceNumber method.
GetRefNumber	String	Returns the reference number associated with the transaction. The reference number is used to help identify the transaction and is useful for the cardholder and merchant when doing research. This value is not returned with all transactions.
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
GetSurchargeAmount	String	**For GSAR Debit US only:** Returns the surcharge amount that was charged by the bank when using debit with GSAR. This value has to be called in order for the developer to know how much the card was actually charged in addition to the transaction amount and cash back.
GetTermFee	String	**For GSAR EBT:** Returns terminal fee sent back by processor. The terminal fee is any surcharge defined and set up by Chase Paymentech and the EBT network of the cardholder's EBT card.
GetTI	String	**For Debit Master Session:** Returns the new working key (if one exists) sent by the processor that should be passed to the PINpad for the next transaction. **For GSAR EBT:** Returns the ledger balance.
GetTraceNum	String	**For GSAR EBT:** Returns the trace number sent back by processor. The trace number is a reference number generated by Chase Paymentech.
GetTransNum	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
GetTroutD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.

Method Name	Returned Value	Description - Debit.OCX Methods
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved can be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.
GetXMLRequest	String	This method is used to echo the text that is sent in the request file associated with the transaction. The request (.inx) file contains XML string data. The text that is sent in the .inx file can be used to view the message of any transaction sent to the server. Note: This method must be called after the calling send and before DeleteUserFiles method.
PccSysExists	Boolean	The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions. For use with File_Transfer CommMethod only
Send	None	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the control will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has an optional parameter that indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send 3 **Note:** The other values that appear in the enumerated list are for internal use only-- do not attempt to use any values other than the ones listed above.
VerifyAmount	Boolean	The VerifyAmount method returns TRUE if the amount provided in the Amount property is in a valid format (DDDDDD.CC), or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
SetSSLCertificate	None	User can set the local certificate to authenticate the SSL TCP/IP request. **Note:** This routine has four parameters. CertificateName : this will store the Store name of the store CertificateLocation : this will assign the store location CertificateIssuesby : This will assign Certificate issued by CertificateSerialNumber : This will assign certificate serial number **Note**: See SSL Information on page 70 for more information.

Debit.OCX Events

Event	Description - Debit.OCX Events
Error	The Error event is fired any time an error occurs in the control. Once an Error event has fired, call GetErrorCode and GetErrorDesc to determine what kind of error has occurred. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
Finish	The Finish event will fire when the transaction has been completed. This means that **PC**Charge has processed the transaction successfully and has placed a file with the extension of .oux in the **PC**Charge directory. The name of the .oux file will be what was set in the User property of the transaction request. Call the GetResult method to determine whether or not the transaction was approved. A list of valid results can be found in the **Transaction Result Constants** section (see page 156).

Note: When doing asynchronous transactions in an event-driven programming model, it is important to place all result or error routines in either the Finish or Error events. Do not place any code that uses the .get methods after invoking the Send method.

Check.OCX

Check.OCX provides integrators with properties and methods used to submit check verification, guarantee, and conversion transactions to **PC**Charge. To use Charge.OCX to integrate transaction processing, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to process transactions by using the PccSysExists method.

2. Assign the appropriate values to the properties required for the transaction to be performed. (The properties marked with a ° in the **Check.OCX properties** table are the minimum required to process a check verification transaction.)

3. Call the Send method. (**Note:** When calling the Send method, it is recommended that "3" is passed as a parameter to activate the XML message format)

4. Wait for the Error or Finish event to occur.

5. Call the various .Get methods to determine the outcome of the transaction. The most important information can be acquired by calling the GetResult and GetAuth methods. If an error occurs, call the GetErrorCode and GetErrorDesc methods to determine the nature of the error.

6. Call the DeleteUserFiles method to delete all files related to the transaction.

7. Call the Clear method to reset all the properties and methods related to the transaction or destroy the object.

Note: **PC**Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

Check.OCX Properties

Property Name	Data Type	Description - Check.OCX Properties
Account_Number°°	String	For Check, MICR, or Double ID: The account number that will be used when processing the transaction. **Max Length:** 20 characters.
Action°	String	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
Amount°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
AdjustmentAmount	String	Total amount of the transaction after adjustment (i.e. if the original transaction was $5.00 and it should have been $50.00, the adjustment transaction request should have the .Amount property set equal to 50.00).
Birth_Date°°	String	The date of birth of the check writer. **Length:** Exactly six characters. **Format:** MMDDYY. The birth date is required for DL (Driver's License) check transactions.
Cash_Back	String	The amount of cash back that the customer will receive. This amount is in addition to value entered in Amount property. For example, if the total amount of the sale is $10 and the customer has requested $5 cash back, Amount should be set to $10 and CashBack should be set to $5. The CashBack property should be formatted the same the Amount property. **Max Length:** 9 characters. **Note:** Some processors do not support the cash back feature.
CashierNum	String	The Cashier Number
CheckType	String	**Valid Values:** 0 = Personal check, 1 = Business check **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
Check_Number°	String	The check number of the check that will be used when processing the transaction. **Max Length:** 10 characters.
CommMethod	Enum	Specifies which communication method will be used. 0 – File_Transfer 1 – TCP/IP Please refer to page 20 for a description of these methods. If TCP/IP is selected, the IPAddress, Port and EnableSSL properties must also be set. If File_Transfer is set then the Path property must be set.
CustomerCity	String	The customer's city. **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
CustomerName	String	The first and last name of the customer. **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.

Property Name	Data Type	Description - Check.OCX Properties
CheckReaderCode	Enum	Passes the type of Check Reader that is being used. Currently only used by Telecheck and will only be set if TECK is the set processor. Cannot be configured in the PCCharge GUI. **Valid Values:** 1 - Magtek_mini_micr 2 - EnCheck_3000 3 - IVI_2500 4 - IVI_430 5 - IVI_431 6 - ICE_5700 7 - MagtekImager 8 - VeriFone_CR1000i 9 - Epson_TMH6000 10 - Epson_TMH6000Imager 11 - WelchAllyn_ScanTeam 8300 12 - VeriFone_CR600 13 - Magtek_Imager_with_Modem 14 - IBM_4610_reader_printer 15 - Ingenico_EC2600 16 - RDM_EC5000 17 - RDM_EC6000 18 - NCR_7158_and_7167 19 - LS_100 20 - Magtek_Excella 21 - Magtek_Excella_DLCapture_FBChkImg 22 - Verifone_Model_Quartet
CustomerStreet	String	The street address of the customer. **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
DLTrackII	String	The parsed TrackII data from the driver's license. **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
Drivers_License°°	String	The driver's license number of the individual writing the check. **Max Length:** 20 characters. The driver's license is required for DL (Driver's License) transactions and when performing Double ID transactions.
EnableSSL	Boolean	**For use with TCP/IP CommMethod only**. SSL is available starting with PCCharge version 5.8. Set to `True` to enable SSL Socket integration. **Note**: See SSL Information on page 70 for more information.
IPAddress	String	**For use with TCP/IP CommMethod only**. The IPAddress of the computer on which PCCharge is running.
ManagerNum	String	Used for BPS Double ID transactions. Optional Manager Number for manager override.
Manual°	String	Flag that indicates whether the transaction was manually entered or swiped. **Valid values:** `0` = manual transaction, `1` = swiped transaction
MerchantNumber°	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Check Services Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
MICR_DATA	String	The raw MICR data from the bottom of the check. Used for conversion transactions.
MICRStatus	String	**Valid Values:** 15 = Valid read by MICR reader, 15I = Valid read by MICR reader with imaging capability, 9 = Manual only **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
Multi	String	Flag that indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PC**Charge GUI. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid values:** `1 = TRUE, 0 = FALSE` **Default value:** `0`. See the section **Multi-trans Wait** for more information (see page 67). This Flag has no effect if processing will occur over IP or leased line.
OutDelay	Single	The delay time before the **PC**Charge directory is polled for a transaction response file (`.oux` file). The default is 0.25 seconds. This value should only be modified if the integration is not performing properly. This could be caused if the client machine is slow or there is network lag that causes the server to spend more time checking for `.oux` files than processing transactions.

Property Name	Data Type	Description - Check.OCX Properties
Path°	String	**For use with File_Transfer Method only**. The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ (**default**) **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Phone_Number°°	String	The phone number of the individual writing the check. **Max Length:** 7 digits. **Format:** digits only. The phone number is required for COD (Checks On Delivery).
Port	String	**For use with TCP/IP CommMethod only.** Open port of PCCharge.
Processor°	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
Services	ServicesType	The type of check verification to be performed. This property may be specified by using a numerical value or an enumerated value if the programming language being used supports enumerated values. **Valid values:** 0 (MICR) – MICR 1 (COD) – Checks-On-Delivery 2 (DL) – Driver's License 3 (DI) – Double ID 4 (SPS) – Use if Check processor is SPS **Note:** The value set in the Services property overrides the value set in the Action property.
PrintReceipts	String	The number of receipts that **PC**Charge should print for the transaction. This value will override the corresponding value in the **PC**Charge GUI. **PC**Charge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.
State°°	String	The state code of the state that issued the check writer's driver's license. The state code is required for DL (Driver's License). **Format:** 2 characters.
Ticket	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.
TimeOut	Long	The number of seconds after which a timeout error will be returned from the control. The count will start when the Send method is called. The default timeout value is 90 seconds. It is highly recommended that integrators review the section **Timeouts** (see page 59). Setting the TimeOut value improperly could cause reconciliation issues and problems such as double-charging a customer's account.
Transit_Number°°	String	The Transit Routing Number / ABA number that will be used when processing the transaction. This value indicates which bank issued the check. **Max Length:** 9 characters. This value is required for MICR transactions and when performing Double ID transactions.
TroutD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
User° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).

Property Name	Data Type	Description - Check.OCX Properties
XMLtran	*Boolean*	*Set to True to activate the XML message format. It is recommended that the "3" parameter be passed to the Send method to activate the XML message format instead of using the XMLtran property. See the description for the Send method for more information.*
Zip_Code°°	String	The check writer's ZIP code. **Max Length:** 9 characters. **Format:** digits only. This value is required for COD transactions. **Note:** If submitting the 9-digit zip, do not include the dash.
CashierName	String	Submit Cashier's login name.
CashierPassword	String	Submit Cashier's password.

Note: To perform Double ID, both the MICR_DATA and Drivers_License fields must be populated.

° These properties are required, regardless of service type.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

°° COD -- required for Checks-On-Delivery
DL -- required for Driver's License
MICR -- required for MICR

Check.OCX Methods

Method Name	Returned Value	Description - Check.OCX Methods
About	None	The About method will display the About box associated with the control.
Cancel	None	The Cancel method attempts to cancel the transaction in progress. Calling the Cancel method does not guarantee that the transaction will be canceled; it simply attempts to cancel the transaction.
Clear	None	The Clear method will clear the values in all properties and methods. It is recommended that this method be called: a. after the transaction results have been retrieved by using the various .get methods b. after the DeleteUserFiles method has been called c. prior to running the next transaction
DeleteUserFiles	None	The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files, the Error event will be triggered and the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150). For use only with File_transfer CommMethod
GetApproved	String	The GetApproved method returns TRUE if **PC**Charge returns "APPROVED" as the result of the transaction. Otherwise, FALSE will be returned. An "APPROVED" response indicates that a Verification has been approved.
GetAuth	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
GetCaptured	Boolean	The GetCaptured method returns TRUE if **PC**Charge returns "CAPTURED" as the result of the transaction. Otherwise, FALSE will be returned. The GetCaptured method is used to determine if a conversion transaction that will result in a monetary transfer is approved or declined. A "CAPTURED" response indicates that the transaction has been approved, and that the transaction has been placed in the open batch.

Method Name	Returned Value	Description - Check.OCX Methods
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetMerchantNumher	String	Returns the merchant number that was specified in the MerchantNumber property.
GetRespCode	String	Returns the response code that is provided by the processor. This value is not returned by all processing companies.
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
GetResultCode	String	Returns the result code that is provided by the processor. This value is not returned by all processing companies.
GetReturnCheckFee	String	Returns the response from the processor which indicates the fee for returned checks. **Note**: Only used for the processor TECK
GetReturnCheckNote	String	Returns the response from the processor which displays a note for returned checks. **Note**: Only used for the processor TECK
GetReference	String	Returns the reference number that is provided by the processor. This value is not returned by all processing companies
GetTransNum	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
GetTroutD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
GetTraceID	String	**Only for TECK.** Returns the Trace ID associated with the transaction.
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved from the .oux file can be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.
GetXMLRequest	String	This method is used to echo the text that is sent in the request file associated with the transaction. The request (.inx) file contains XML string data. The text that is sent in the .inx file can be used to view the message of any transaction sent to the server. Note: This method must be called after the calling send and before DeleteUserFiles method.
PccSysExists	Boolean	The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions. For use only with File_Transfer CommMethod

Method Name	Returned Value	Description - Check.OCX Methods
Send	None	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the control will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has an optional parameter that indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send 3 **Note:** The other values that appear in the enumerated list are for internal use only—do not attempt to use any values other than the ones listed above.
VerifyAmount	Boolean	The VerifyAmount method returns TRUE if the amount provided in the Amount property is in a valid format (DDDDDD.CC), or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
SetSSLCertificate	None	User can set the local certificate to authenticate the SSL TCP/IP request. **Note:** This routine has four parameters. CertificateName : this will store the Store name of the store CertificateLocation : this will assign the store location CertificateIssuesby : This will assign Certificate issued by CertificateSerialNumber : This will assign certificate serial number **Note**: See SSL Information on page 70 for more information.

Check.OCX Events

Event	Description - Check.OCX Events
Error	The Error event is fired any time an error occurs in the control. Once an Error event has fired, call GetErrorCode and GetErrorDesc to determine what kind of error has occurred. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
Finish	The Finish event will fire when the transaction has been completed. This means that **PC**Charge has processed the transaction successfully and has placed a file with the extension of .oux in the **PC**Charge directory. The name of the .oux file will be what was set in the User property of the transaction request. Call the GetResult method to determine whether or not the transaction was approved. A list of valid results can be found in the **Transaction Result Constants** section (see page 156).

Note: When doing asynchronous transactions in an event-driven programming model, it is important to place all result or error routines in either the Finish or Error events. Do not place any code that uses the .get methods after invoking the Send method.

EBT

The structure of EBT transactions is similar enough to debit transactions that both are handled via the `Debit.OCX`. Consult the section **Debit.OCX** for more information on this ActiveX Control (see page 208).

GiftCard.OCX

GiftCard.OCX provides integrators with properties and methods used to submit gift card transactions to **PC**Charge. To use GiftCard.OCX to integrate transaction processing, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to process transactions by using the PccSysExists method.

2. Assign the appropriate values to the properties required for the transaction to be performed and validate the values using the various .Verify methods. (The properties marked with a ° in the **GiftCard.OCX properties** table are the minimum required to process a Gift Card Redemption / Sale transaction.)

3. Call the Send method. (**Note:** When calling the Send method, it is recommended that "3" is passed as a parameter to activate the XML message format)

4. Wait for the Error or Finish event to occur.

5. Call the various .Get methods to determine the outcome of the transaction. The most important information can be acquired by calling the GetResult and GetAuth methods. If an error occurs, call the GetErrorCode and GetErrorDesc methods to determine the nature of the error.

6. Call the DeleteUserFiles method to delete all files related to the transaction.

7. Call the Clear method to reset all the properties and methods related to the transaction or destroy the object.

Note: **PC**Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

VeriFone Stored Value API (GAPI)

The VeriFone Stored Value API (GAPI) is a proprietary specification that allows for stored value card processors to add themselves to **PC**Charge. Applications using GAPI can also integrate with **PC**Charge using the various integration methods. For more information on adding a stored value card processor to **PC**Charge, and how to obtain the VeriFone Stored Value API, please contact VeriFone sales at 1-800-725-9264.

GiftCard.OCX Properties

Property Name	Data Type	Description - GiftCard.OCX Properties
Action°	String	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
Amount°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00". For Valuelink (VLNK) Balance Adjustment: **Format:** +/-DDDDD.CC.
Authcode	String	Used for VTEC, VLNK and GSAR **Void** transactions. For VTEC and VLNK, set to auth code of original transaction (the one to be voided). For GSAR and MELL, set to reference number of the original transaction (the one to be voided). For BPS, set to retrieval reference number of original transaction (the one to be voided). For BPS, set to retrieval reference number of original transaction (the one to be voided).
Card°	String	The gift card number that will be used when processing the transaction. **Max Length:** 20 characters.
CardSeqNum	String	For GSAR multi **Issuance**, sequence number of cards issued at time of transaction. **Example:** Ten cards are being issued. The fifth is being sent, so set CardSeqNum to 5.
CashierID	String	VTEC and VLNK – (optional) – numeric value that identifies the cashier performing the transaction.
CheckCard	Boolean	Flag that indicates whether to activate gift card validity testing. **Valid Values:** TRUE; FALSE. **Default value:** TRUE. This value must be set to FALSE when performing Follow on transactions because the card number is omitted from these transaction requests.
CommMethod	Enum	Specifies which communication method will be used. 0 – File_Transfer 1 – TCP/IP Please refer to page 20 for a description of these methods. If TCP/IP is selected, the IPAddress, Port and EnableSSL properties must also be set. If File_Transfer is set then the Path property must be set.
Demo	String	The demo mode flag. In demo mode, a simulated response is returned in which even amounts return approved, and odd amounts return declined. **Valid Values:** 1 – Activates demo mode 0 – Deactivates demo mode (default)
EnableSSL	Boolean	**For use with TCP/IP CommMethod only**. SSL is available starting with PCCharge version 5.8. Set to True to enable SSL Socket integration. **Note**: See SSL Information on page 70 for more information.
ExpDate	String	The expiration date associated with the gift card that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208 **Note:** Most gift cards do not have an expiration date.
FORCE	Boolean	Set to TRUE to process a transaction for which an approval code has already been issued -- only valid for a GSAR **Redemption** transaction or a single GSAR **Issuance/Add Value** transaction.
GratuityAmount	String	The gratuity amount for the transaction. Tip should be no more than 9 characters long (including the decimal). **Format:** DDDDDD.CC.
GiftPin	String	Only used for the processor **SVS**. To retrieve pin, call GetGfitPin upon activation. Used for only for virtual gift card transactions.

Property Name	Data Type	Description - GiftCard.OCX Properties
Industry°°	String	Indicates industry type (1 = retail, 2 = restaurant). For VLNK (0 = retail, 1 = restaurant, 2 = e-Commerce).
IPAddress	String	**For use with TCP/IP CommMethod only.** IPAddress of machine where PCCharge is running. Defaults to 127.0.0.1
LastValidDate	String	The last year that will be considered a valid expiration date. Currently, the default value in the control is "09". It is recommended that a setting is provided by which the end-user can change this property; otherwise, in the future, end users will require a new control to be distributed to resolve expiration date issues. **Length:** 2 digits. **Format:** YY **Example:** If LastValidDate is set to 05, then cards between 06 and 99 are considered to be 1906 to 1999, and cards between 00 and 05 are 2000 to 2005.
Loyalty	Boolean	VTEC loyalty transaction flag (0 = non-loyalty, 1 = loyalty).
Manual°	Long	Flag that indicates whether the transaction was manually entered or swiped. If the transaction was swiped, the Track property must also be set. **Valid values:** 0 = manual transaction, 1 = swiped transaction
Multi	String	Flag that indicates whether **PCC**harge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PCC**harge GUI. Note that **PCC**harge can only keep the connection open as long as is allowed by the processing company. **Valid values:** 1 = TRUE, 0 = FALSE **Default value:** 0. See the section **Multi-trans Wait** for more information (see page 58). This Flag has no effect if processing will occur over IP or leased line.
MerchantNumber°	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Gift Card Setup** window of **PCC**harge. **Max Length:** 32 characters. This value can be alphanumeric.
OldCard	String	For VTEC **Replace** transaction. Set to account number of old card. For VLNK, **Balance Merge** and **Balance Transfer**.
OutDelay	Single	The delay time before the **PCC**harge directory is polled for a transaction response file (.oux file). The default is 0.25 seconds. This value should only be modified if the integration is not performing properly. This could be caused if the client machine is slow or there is network lag that causes the server to spend more time checking for .oux files than processing transactions.
Partial	Boolean	For GSAR: Flag indicating whether the transaction is a partial redemption transaction.
Path°	String	**For use with File_Transfer CommMethod method only**. The path to the directory in which the **PCC**harge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PCC**harge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ (**default**) **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Points	String	For GVEX **Points** transactions. Set to number of loyalty points for account.
Port	String	**For use with TCP/IP CommMethod only.** Open port of PCCharge.
PrintReceipts	String	The number of receipts that **PCC**harge should print for the transaction. This value will override the corresponding value in the **PCC**harge GUI. **PCC**harge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.
Processor°	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PCC**harge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
PromoCode	String	Used for GVEX: A code defined by the merchant that affects the calculation from amount and units to points.

Property Name	Data Type	Description - GiftCard.OCX Properties
Refund	String	Flag that indicates whether to provide the customer a refund when performing a VTEC Deactivate transaction. **Valid Values:** 1 – Provide refund 0 – Do not provide refund
RFID	String	Set to 1 if card information was read from RFID (Radio Frequency Identification) device. If card was read from from RFID, track data must be populated and manual flag must be set to 1. Set to 0 otherwise.
Ticket	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all gift processors support ticket numbers.
TimeOut	Long	The number of seconds after which a timeout error will be returned from the control. The count will start when the Send method is called. The default timeout value is 90 seconds. It is highly recommended that integrators review the section **Timeouts** (see page 59). Setting the TimeOut value improperly could cause reconciliation issues and problems such as double-charging a customer's account.
TIP	String	The tip amount for the transaction. TIP should be no more than 9 characters long (including the decimal). **Format:** DDDDDD.CC. Currently, tips are supported via the TIP property only for VTEC and VLNK restaurant transactions.
TotalCardNum	String	For GSAR multi **Issuance**, total number of cards being issued at time of transaction.
Track	String	The track II data captured from the magnetic strip of the card.. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in. Alternatively, the GetParseData method can be used to parse the track data and set the Card, ExpDate, Member, and Track properties automatically.
TroutD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
User° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).
VirtualGiftCardFlag	Boolean	0 - False, 1-True – Only used for the processor **SVS**. Only used on an activation to determine if a pin should be returned.
XMLtran	*Boolean*	*Set to True to activate the XML message format. It is recommended that the "3" parameter be passed to the Send method to activate the XML message format instead of using the XMLtran property. See the description for the Send method for more information.*
TableNumber	String	Only used for GAPI in restaurant mode. This is the table number of the gift card holder
TrackI	String	Only used for GAPI. The Track I information associated with the card
CashierName	String	Submit Cashier's login name.
CashierPassword	String	Submit Cashier's password.

Property Name	Data Type	Description - GiftCard.OCX Properties
SkuLoyalty	String	**GIVEX ONLY** – **Integration Only** Pass this tag in order to send product codes, quantities and amounts for loyalty transactions. **Format:** **Single Product Code:** <SKU_LOYALTY>Product Code,Amount,Quantity</SKU_LOYALTY> **Multi-Product Codes (Separated with a semi-colon):** <SKU_LOYALTY>Product Code,Amount,Quantity; Product Code,Amount,Quantity</SKU_LOYALTY>

° These properties are required to process a gift card redemption or sale transaction.

°° Required for VTEC gift card transactions

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

GiftCard.OCX Methods

Method Name	Returned Value	Description – GiftCard.OCX Methods
Abort	Boolean	The Abort method attempts to cancel the transaction in progress and will return a Boolean value that indicates whether or not the transaction was canceled. **Note:** This method is not available when integrating using FoxPro. Use the Cancel method instead.
About	MsgBox	The About method will display the About box associated with the control.
Cancel	None	The Cancel method attempts to cancel the transaction in progress. Calling the Cancel method does not guarantee that the transaction will be canceled; it simply attempts to cancel the transaction.
Clear	None	The Clear method will clear the values in all properties and methods. It is recommended that this method be called: a. after the transaction results have been retrieved by using the various .get methods b. after the DeleteUserFiles method has been called c. prior to running the next transaction
DeleteUserFiles	None	The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files, the Error event will be triggered and the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150). For use only with File_Transfer CommMethod
GetActivationCount	String	Returns the number of activations in the current batch
GetActivationTotalAmount	String	Returns the total dollar amount of activations in the current batch
GetAddPointsCount	String	Returns the number of AddPoints Transactions in the current batch
GetAddPointsTotalAmount	String	Returns the total dollar amount of AddPoints transactions in the current batch
GetAddValueCount	String	Returns the number of AddValue transactions in the current batch
GetAddValueTotalAmount	String	Returns the total dollar amount of AddValue transactions in the current batch
GetAmountDue	String	Returns the amount due when Partial Auth is enabled. **Note:** With Elavon (NOVA) as the processor, "Enable Pre-Paid Cards" option within PCCharge MUST be checked for this feature to work. The option is located under Setup>Credit Card Company>Extended Data. This option is only configurable in *PCCharge version 5.7.1 release I sp9a* and above.
GetAuth	String	The GetAuth method returns the authorization number for approved transactions or the reason the transaction was declined (if the processor provides one). **For GVEX Balance transaction:** GetAuth will return the balance remaining on an account. **For all other GVEX transactions:** GetAuth will return the transaction's reference/error message. For VTEC, returns the Auth Code. **For a VTEC Batch function:** use this method to retrieve the number of sales done that day and the total amounts of sales in the following format <# of transaction>, <amount>.
GetAuthAmount	String	Used in partial redemption transactions where only part of the amount was authorized. Returns the actual authorized amount.
GetBalanceTransferCount	String	Returns the number of Balance Transfers in the current batch
GetBalanceTransferTotalAmount	String	Returns the total dollar amount of Balance Transfers in the current batch
GetCaptured	Boolean	The GetCaptured method returns TRUE if **PC**Charge returns "CAPTURED" as the result of the transaction. Otherwise, FALSE will be returned. The GetCaptured method is used to determine if a transaction that will result in a monetary transfer (Sale, Credit, Post-Authorization, etc.) is approved or declined. A "CAPTURED" response indicates that the transaction has been approved.

Method Name	Returned Value	Description - GiftCard.OCX Methods
GetCashBack	String	Used in redemption for remaining balance transactions where the transaction amount is so close to the balance of the card that the entire balance is authorized. Returns the remainder that is owed to the customer.
GetCreditCount	String	Returns the number of credits in the current batch
GetCreditTotalAmount	String	Returns the total dollar amount of credits in the current batch
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetExp	String	Returns the expiration date for processors who issue expiration dates in the response.
GetGiftCardBalance	String	Returns the gift card balance.
GetGiftCardIssuer	String	Returns the gift card issuer. Consult the section **DevKit Constants** for description of values (see page 143).
GetGiftCardType	String	Returns type of gift card represented by card property. Consult the section **DevKit Constants** for description of values (see page 143).
GetGiftPin	String	Only used for the processor **SVS**. Returned on activation if the virtual gift card tag is set to "1".
GetLevel	String	Returns the customer's loyalty level. Only used for VTEC loyalty gift cards.
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetParseData	String	The GetParseData method will parse a string (containing credit card track data) passed to it and populate the Card, ExpDate, and Track properties with the appropriate data. GetParseData will return an integer indicating its success. **Valid return values:** 0 (error parsing data), 1 (track I successful), or 2 (track I & II successful).
GetPointsBalance	String	Returns points balance for loyalty cards
GetPointsCount	String	Returns the number of points transactions in the current batch
GetPointsTotalAmount	String	Returns the total dollar amount of points transactions in the current batch
GetProcRespCode	String	The processor response code. Only returned by the processor **SVS**.
GetRefNumber	String	The GetRefNumber returns the Reference field from the .oux file. The Reference field is used for different purposes (depending on the gift card processor). **For GVEX Register transaction:** The first eleven digits of an account number will be returned. **For all VTEC transactions:** The account's remaining balance will be returned. **For a VTEC batch function:** use this method to retrieve the number of activations done that day and the total amounts of activations in the following format <# of transaction>, <amount>.>. For a BPS **Redemption** transaction, returns the retrieval reference number.
GetRespCode	String	Returns the response code that is provided by the processor. This value is not returned by all processing companies.
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
GetRet	String	**For GVEX:** Returns the loyalty balance. **For VLNK:** Returns the trace number. **For a VTEC batch function: :** use this method to retrieve the number of Gift Transactions Voids performed that day. You can call GetVoidBalance to determine the total amount of the voids.
GetSaleCount	String	Returns the number of redemptions in the current batch
GetSaleTotalAmount	String	Returns the total dollar amount of redemptions in the current batch

Method Name	Returned Value	Description - GiftCard.OCX Methods
GetTI	String	The GetTI method returns the TI field from the .oux file. The TI field is used for different purposes (depending on the gift card processor and transaction type). **For GVEX Register:** The remaining digits of an account number will be returned. **For GVEX Redemption, Increment, and Cancel:** The balance remaining on the card will be returned. **For a VTEC batch function: :** use this method to retrieve the number Add Value Transactions done that day and the total amounts of Add Value in the following format <# of transaction>, <amount>>.
GetTicket	String	The GetTicket method returns the Ticket field from the .oux file. The Ticket field will return the ticket for all transactions except for a VTEC batch function. **For a VTEC batch function:** use this method to retrieve the number of gift card that has been de-activated that day and the total amounts of de-activations in the following format <# of transaction>, <amount>.>.
GetTIM	String	Returns the Time of the transaction. This value is not returned by all processing companies. For VTEC, returns the Amount Due.
GetTipCount	String	Returns the number of Tip transactions in the current batch
GetTipTotalAmount	String	Returns the total dollar amount of Tip transactions in the current batch
GetTransDateTime	String	Returns the transaction date and time when passed back by a processor.
GetTransNum	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
GetTroutD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
GetUpdateData	*String*	*Used internally*
GetVoidBalance	String	Returns the Void Balance
GetVoidCount	String	Returns the number of voids in the current batch
GetVoidTotalAmount	String	Returns the total dollar amount of Voids in the current batch
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved from the .oux file can be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.
GetXMLRequest	String	This method is used to echo the text that is sent in the request file associated with the transaction. The request (.inx) file contains XML string data. The text that is sent in the .inx file can be used to view the message of any transaction sent to the server. Note: This method must be called after the calling send and before DeleteUserFiles method.
PccSysExists	Boolean	The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions. For use only with File_Transfer CommMethod

Method Name	Returned Value	Description - GiftCard.OCX Methods
Send	None	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the control will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has an optional parameter that indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) - XML message format - (**RECOMMENDED**) **Example:** Send 3 **Note:** The other values that appear in the enumerated list are for internal use only—do not attempt to use any values other than the ones listed above.
ValidCardLength	Boolean	Returns TRUE for card of correct length
ValidDate	Boolean	The ValidDate method returns TRUE if the expiration date provided in the ExpDate property is valid, or FALSE if it is not.
ValidIssuer	Boolean	Returns TRUE for valid card issuer
VerifyAmount	Boolean	The VerifyAmount method returns TRUE if the amount provided in the Amount property is in a valid format (DDDDDD.CC), or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
VerifyAmount2	Boolean	The VerifyAmount2 method returns TRUE if the amount provided in the Amount property is in a valid format (+/-DDDDD.CC). or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150). The difference between VerifyAmount and VerifyAmount2 is that VerifyAmount2 allows a + or - to be in the first position of the Amount property. This is needed for Balance Adjustment transactions.
VerifyExpDate	Boolean	The VerifyExpDate method returns TRUE if the expiration date provided in the ExpDate property is correct and in the right format, or FALSE if it is not. VerifyExpDate calls the ValidDate function to validate the expiration date. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
VerifyGiftCard	Boolean	The VerifyGiftCard method verifies that a card is provided and the card is the expected length and return TRUE is it passes, otherwise FALSE will be returned.
VerifyMerchantNumber	*Boolean*	*No Longer Supported*
VerifyProcessor	*Boolean*	*No Longer Supported*
GetPreAuthCount	String	Only for GAPI, this returns the total number of gift card pre-auth transactions processed that day.
GetPreAuthAmount	String	Only for GAPI , this returns the total amount of gift card pre-auth transaction processed that day.
GetPostAuthCount	String	Only for GAPI, this returns the number of the gift card post-auth transactions processed that day.
GetPostAuthAmount	String	Only for GAPI, this returns the total amount of the post-auth transactions processed that day.

Method Name	Returned Value	Description - GiftCard.OCX Methods
GetIssuanceCount	String	Only for GAPI, this returns the total number of gift cards issued that day.
GetIssuanceTotalAmount	String	Only for GAPI, returns the total amount of the gift cards issued that day.
GetDeactivateCount	String	Only for GAPI, this returns how many gift cards where deactivated that day.
GetDeactivateTotalAmount	String	Only for GAPI, this returns the total amount of gift card deactivations that day.
GetBalanceAdjustCount	String	Only for GAPI, this returns the total number of gift cards that were balance adjusted that day.
GetBalanceAdjustTotalAmount	String	Only for GAPI, this returns the total amount of balance adjustments on gift cards that day.
GetBalanceMergeCount	String	Only for GAPI, this returns the number of the gift cards that were balance merged that day.
GetBalanceMergeTotalAmount	String	Only for GAPI, this returns the total amount of gift card balance merges that day.
GetReportLostStolenCount	String	Only for GAPI, returns the total reported stolen or lost gift cards that day.
GetReportLostStolenTotalAmount	String	Only for GAPI, returns the total amount of all stolen or reported lost gift cards that day.
GetCashoutTotalAmount	String	Only for GAPI, returns the total amount of all cashout transactions processed that day.
GetCashoutCount	String	Only for GAPI, returns the total number of the cashout transactions processed that day.
GetReactivateCount	String	Only for GAPI, returns the total number of gift cards that have been reactivated that day.
GetReactivateTotalAmount	String	Only for GAPI, the total amount of all gift cards that have been reactivated that day.
SetSSLCertificate	None	User can set the local certificate to authenticate the SSL TCP/IP request. **Note:** This routine has four parameters. CertificateName : this will store the Store name of the store CertificateLocation : this will assign the store location CertificateIssuesby : This will assign Certificate issued by CertificateSerialNumber : This will assign certificate serial number **Note**: See SSL Information on page 70 for more information.

GiftCard.OCX Events

Event Name	Description - GiftCard.OCX Events
Error	The Error event is fired any time an error occurs in the control. Once an Error event has fired, call GetErrorCode and GetErrorDesc to determine what kind of error has occurred. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
Finish	The Finish event will fire when the transaction has been completed. This means that **PC**Charge has processed the transaction successfully and has placed a file with the extension of .oux in the **PC**Charge directory. The name of the .oux file will be what was set in the User property of the transaction request. Call the GetResult method to determine whether or not the transaction was approved. A list of valid results can be found in the **Transaction Result Constants** section (see page 156).

Note: When doing asynchronous transactions in an event-driven programming model, it is important to place all result or error routines in either the Finish or Error events. Do not place any code that uses the .get methods after invoking the Send method.

Batch.OCX

Batch.OCX provides integrators with properties and methods used to perform batch settle, close, and inquire operations. This control may be used with both Host based and Terminal based processors. To use Batch.OCX to perform batch operations, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to perform batch operations by using the PccSysExists method.

2. Assign the appropriate values to the properties required for the batch operation to be performed. (The properties marked with a ° in the **Batch.OCX properties** table are the minimum required to settle a batch.)

3. Call the Send method.

4. Wait for the Error or Finish event to occur.

5. Call the various .Get methods to determine the outcome of the transaction. The most important information can be acquired by calling the GetResult method. If an error occurs, call the GetErrorCode and GetErrorDesc methods to determine the nature of the error.

6. Call the DeleteUserFiles method to delete all files related to the transaction.

7. Call the Clear method to reset all the properties and methods related to the transaction or destroy the object.

Note: When using action code 39, an appropriate timeout value must be set to allow for closing/settling of all merchant numbers installed in **PC**Charge. If action code 39 is chosen, a default value of 1200 seconds is selected. If a longer time is needed, the timeout property must be set to an adequate value.

Note: PC Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

Batch.OCX Properties

Property Name	Data Type	Description - Batch.OCX Properties
Action°	Single	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
BatchCloseType	String	Flag that determines what type of batch close will occur. This flag only supported by FDMS Atlanta and Fifth-Third when using action code 30 or 31 **Valid values:** 1 – Standard End of Day Batch Close (Default) 2 – Shift Close 3 – Fifth-Third Terminal Based Batch Close of Debit, EBT, or Gift
Cancel	Boolean	Set the Cancel property to TRUE to attempt to cancel the settle/close function. Check the GetResult method to see if the function was Canceled.
CommMethod	Enum	Specifies which communication method will be used. 0 – File_Transfer 1 – TCP/IP Please refer to page 20 for a description of these methods. If TCP/IP is selected, the IPAddress, Port and EnableSSL properties must also be set. If File_Transfer is set then the Path property must be set.
EnableSSL	Boolean	**For use with TCP/IP CommMethod only**. SSL is available starting with PCCharge version 5.8. Set to True to enable SSL Socket integration. **Note**: See SSL Information on page 70 for more information.
IPAddress	String	**For use with TCP/IP CommMethod only.** IPAddress of machine where PCCharge is running. Defaults to 127.0.0.1
MerchantNumber°	String	Account number issued to merchant by processor
OutDelay	Single	The delay time before the **PC**Charge directory is polled for a transaction response file (.oux file). The default is 0.25 seconds. This value should only be modified if the integration is not performing properly. This could be caused if the client machine is slow or there is network lag that causes the server to spend more time checking for .oux files than processing transactions.
Path°	String	**For use with File_Transfer CommMethod method only**. The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ (**default**) **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Port	String	**For use with TCP/IP CommMethod only.** Open port of PCCharge.
Processor°	String	The code for the processing company that will be used to perform batch operations. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
SplitProcessor	String	Only used when settling the processor CITI for private label transactions. Set this property to the main credit card processor ID code being used.
TimeOut°	Long	The number of seconds after which a timeout error will be returned from the control. The count will start when the Send method is called. The default timeout value is 90 seconds. It is highly recommended that integrators review the section **Timeouts** (see page 59).
User° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).
XMLtran	*Boolean*	*Set to True to activate the XML message format. It is recommended that the "3" parameter be passed to the Send method to activate the XML message format instead of using the XMLtran property. See the description for the Send method for more information.*

Property Name	Data Type	Description - Batch.OCX Properties
Command	String	
CashierName	String	Submit Cashier's login name.
CashierPassword	String	Submit Cashier's password.
BatchNumber	String	Submit batch number that was returned on the AMEX split dial settlement, needs to be passed when performing the following Action Codes: 35 – Reverse Batch 36 – Resubmit Batch 37 – Get Results 38 – Finalize Batch
CardType	String	Submit with a Batch Inquiry to receive summary results based on Card Type. Valid Values: MC – MasterCard VISA – Visa AMEX – American Express DISC – Discover Card DEBIT – Debit Card EBT – Electronic Benefits Card PL – Private Label JCB – JCB Card

° These properties are required to settle a batch.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

The following properties are no longer available in the Batch.OCX and should be ignored:

AmexAmount	PurchaseCount
AmexCount	Response
Balance	ReturnAmount
BatchDate	ReturnCount
BatchNumber	Store
CIC	Terminal
ItemCount	VisaMCAmount
PurchaseAmount	VisaMCCount

Batch.OCX Methods

Method Name	Returned Value	Description - Batch.OCX Methods
About	Single	The About method will display the About box associated with the control.
Clear	None	The Clear method will clear the values in all properties and methods. It is recommended that this method be called: a. after the transaction results have been retrieved by using the various .get methods b. after the DeleteUserFiles method has been called c. prior to running the next transaction
DeleteUserFiles	None	The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files, the Error event will be triggered and the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150). For use only with File_Transfer CommMethod
GetAccepted(Index)	Boolean	Index is an integer. Returns TRUE if batch settle/close was accepted
GetBalance	String	Returns dollar value of batch
GetBatches	String	Returns number of batches for settlement
GetBatchNumber	String	After a terminal-based batch settles, returns the batch number.
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetItemCount	String	Returns the number of transactions in the batch
GetNumberIndexs	Integer	The GetNumberIndexs method returns the number of merchant numbers that are stored in **PC**Charge. This number indicates how many batches will be settled or closed if an action code of 39 is submitted.
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetProcessed	Boolean	Returns TRUE if inquiry is successful
GetRespCode	String	Returns the response code for the batch if the close batch command was given. The response code indicates whether or not the transaction was successfully closed. If the batch is declined, the GetResult method will provide more information indicating why the transaction was not approved. **Valid Values:** 2 = Settled, 6 = Declined, or 8 = Deferred.
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
GetSettleAmount	*None*	*No Longer Supported*
GetSettleNumber	String	Returns sequence number of batch
GetStatus	String	Returns the status of the batch when performing an inquiry or a batch close/settle operation. If performing a batch close/settlement operation, GetStatus will return a response from the processor that indicates whether or not the batch was successfully closed or settled. **Example:** If TSYS rejects the batch, GetStatus will return the RB (rejected batch) number from TSYS . If TSYS accepts the batch, GetStatus will return the batch number and an "ACCEPTED" response will be returned.

Method Name	Returned Value	Description - Batch.OCX Methods
GetSystemInfo	None	The GetSystemInfo method is used to set the MerchantNumber and Processor properties of Batch.OCX. To use GetSystemInfo, pass the index number of a merchant number that is registered in **PC**Charge as a parameter (for example, the first Merchant number that is set up in **PC**Charge is assigned the index of "1"). Once the index number has been passed to **PC**Charge via GetSystemInfo, the merchant number and processor can be retrieved using the MerchantNumber and Processor properties.
GetTotals	*Boolean*	*No Longer Supported*
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved from the .oux file can be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.
GetXMLRequest	String	This method is used to echo the text that is sent in the request file associated with the transaction. The request (.inx) file contains XML string data. The text that is sent in the .inx file can be used to view the message of any transaction sent to the server. Note: This method must be called after the calling send and before DeleteUserFiles method.
PccSysExists	Boolean	The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions. For use only with File_Transfer CommMethod
Send	None	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the Batch.OCX will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has an optional parameter that indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **Valid values:** 3 (TTYPE_XML) – XML message format – **RECOMMENDED** **Example:** Send 3 **Note:** The other values that appear in the enumerated list are for internal use only-- do not attempt to use any values other than the ones listed above.
SetSSLCertificate	None	User can set the local certificate to authenticate the SSL TCP/IP request. **Note:** This routine has four parameters. CertificateName : this will store the Store name of the store CertificateLocation : this will assign the store location CertificateIssuesby : This will assign Certificate issued by CertificateSerialNumber : This will assign certificate serial number **Note**: See SSL Information on page 70 for more information.
GetRecordCount	String	The number of records matching the inquiry

Method Name	Returned Value	Description - Batch.OCX Methods
GetTerminationStatus	String	Retrieves Termination Status which returns a "6" upon successful settlement. If anything other than a "6" is returned there may be an issue with the settlement file and should be investigated. This applies to : **GSAR**, **Elavon (NOVA)**, and **NBS** **Response:** **6 = Batch Settled and file has been deleted.** **8 = Batch Settled, but the file is locked and cannot be deleted.**
GetEBTCount	String	Number of EBT Transactions
GetEBTAmount	String	Total Dollar Amount of EBT Transactions
GetAMEXCount	String	Number of AMEX Transactions
GetAMEXAmount	String	Total Dollar Amount of AMEX Transactions
GetTotalCount	String	Number of Transactions
GetTotalAmount	String	Total Dollar Amount of Transactions in the batch
GetVISACount	String	Number of VISA Transactions
GetVISAAmount	String	Total Dollar Amount of VISA Transactions
GetMCCount	String	Number of MC Transactions
GetMCAmount	String	Total Dollar Amount of MC Transactions
GetDISCCount	String	Number of DISC Transactions
GetDISCAmount	String	Total Dollar Amount of DISC Transactions
GetPLCount	String	Number of PL Transactions
GetPLAmount	String	Total Dollar Amount of PL Transactions
GetJCBCount	String	Number of JCB Transactions
GetJCBAmount	String	Total Dollar Amount of JCB Transactions

Batch.OCX Events

Event	Description - Batch.OCX Events
Error	The Error event is fired any time an error occurs in the control. Once an Error event has fired, call GetErrorCode and GetErrorDesc to determine what kind of error has occurred. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
Finish	The Finish event will fire when the transaction has been completed. This means that **PC**Charge has processed the transaction successfully and has placed a file with the extension of .oux in the **PC**Charge directory. The name of the .oux file will be what was set in the User property of the transaction request. Call the GetResult method to determine whether or not the transaction was approved. A list of valid results can be found in the **Transaction Result Constants** section (see page 156).

Note: When doing asynchronous transactions in an event-driven programming model, it is important to place all result or error routines in either the Finish or Error events. Do not place any code that uses the .get methods after invoking the Send method.

Note: In the event there are multiple batches waiting to be settled in one settlement, the integrated application will need to be designed to loop through the settlement response to retrieve the response for each batch.

Device.OCX

The `Device.OCX` allows integration to PINpads connected to the serial port. Currently, the `Device.OCX` supports the following devices:

- VeriFone 101/1000 (PIN entry only)
- VeriFone 2000 (Card Swipe and PIN entry only)
- Ingenico eN-Crypt 2100 (Card swipe and PIN entry)
- VeriFone SC5000 (Card Swipe and Pin Entry only. Only SC5000 with VeriFone 2000 emulation)
- VeriFone Everest (Card Swipe, Pin Entry, and Display Manipulation)
- VeriFone Omni 3730 (Card Swipe, Pin Entry, Printing)

Additional devices will be added in the future.

The `Device.OCX` provides the functionality to Initialize a PINpad and retrieve the PIN from the cardholder. If using the Ingenico eN-Crypt 2100, Verifone Everest, 2000, VeriFone Omni 3730, or SC5000, it also allows retrieving the card swipe data. The PINpad must be initialized once prior to retrieving the PIN. If the PINpad is powered off and back on, it must be re-initialized. If the object is destroyed and then instantiated, the PINpad must be re-initialized.

Device.OCX Dependencies

In order for the Device.OCX to function properly, a set of dependencies need to be present and registered in the system32 directory. The dependencies are deployed with the following DevKit(s):

- **IP**Charge DevKit
- **PC**Charge DevKit

Integrators should ensure the following dependencies are part of their deployment package when installing any POS that utilizes the Device.OCX.

- MSCOMM32.OCX
- OPOSLineDisplay.ocx
- OPOSMSR.ocx
- OPOSPINPad.ocx
- OPOSPOSKeyboard.ocx
- SaxComm8.ocx

Device.OCX Properties

Property Name	Data Type	Description - Device.OCX Properties
Amount°°	String	This amount displayed to customer on the PINpad for approval The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
Baud°	String	Baud rate
Card°°	String	The debit card number that will be used when processing the transaction. **Max Length:** 20 characters. **Example:** 5424180279791765 **NOTE:** Is it the integrator's responsibility to remove spaces in the card number if there are spaces in the card number from the track data.
Databits°	String	Databits (**Example:** "7" or "8")
DefaultMessage	String	The default message for the PINpad that is connected to the machine the Device.OCX will be communicating with. For example, if this is set to "Welcome", "Welcome" will appear on the screen at times of inactivity. **Note:** Not applicable for all PinPads.
Demo	Boolean	The Demo mode flag. Indicates whether the Device.OCX should run in demo mode. In demo mode, a simulated response is returned in which the Pin and KeySerialNumber are set to demo values. **Valid Values:** TRUE – Activates demo mode FALSE – Deactivates demo mode (default)
Device°	Enum	The PINpad that will be used. This parameter is specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **Valid values:** 0 (ppdVerifone_101) – Verifone 101 or 1000 1 (ppdIVICheckMate_2100) – Ingenico EN-CRYPT 2100 2 (ppdVerifone_2000) – Verifone 2000 or Verifone SC5000(with Verifone 200 emulator) 3 (ppdVerifone_Everest) – Verifone Everest 4 (ppdVerifone_SC5000) – Verifone SC5000 (w/ 2000 emulation) 5 (ppdVerifone_3730) – Verifone Omni 3730 **Example:** Device = 0
DisplayString	String	**For Verifone Everest only.** Sets message that will display on Pin Pad. Must be set before initializing or calling RefreshDisplay. Maximum length of characters that the Everest can display is 34.
EncryptMethod°	Long	The Encryption method that will be used. This parameter is specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **Valid values:** 0 (ppmMasterSession) – Master Session encryption (only used with Elavon (NOVA)) 1 (ppmDUKPT) – DUKPT (Derived Unique Key Per Transaction **Example:** EncryptMethod = 1
MasterKey	String	Contains the Master Key for Master Session encryption. Not all processors who do Master Session encryption will have a Master Key. Set MasterKey to "0" if no Master Key is present.
Parity°	String	Parity ("E" for even, "O" for odd, "N" for none)
Port°	String	Number of the COM port to be used (**Example:** "1" used for COM port 1)

Property Name	Data Type	Description - Device.OCX Properties
RefreshDisplay	String	**For VeriFone Everest only.** Refreshes display on Pin Pad to the DisplayString property. Must set DisplayString property before calling this method.
ShutDownApp	None	**For VeriFone Omni 3730 only.** Cancels the application and closes the port.
TimeOut	Long	The number of seconds after which a timeout error will be returned from the control. The default timeout value is 15 seconds. It is highly recommended that integrators review the section **Timeouts** (see page 59).
WorkingKey	String	Processor's working key. This value is required for Master Session.

° These properties must be set before `Initialize` can be called.

°° These properties must be set before `GetPin` can be called.

Device.OCX Methods

Method Name	Returned Value	Description - Device.OCX Methods
About	None	The `About` method will display the About box associated with the control.
Cancel	Boolean	The `Cancel` method will close the PINpad's Com port. Only call after the application has been terminated.
Clear	Boolean	The `Clear` method will clear the values in all properties and methods. It is recommended that this method be called after the transaction results have been retrieved by using the various `.get` methods.
GetCard	String	The `GetCard` method returns the card number that was obtained when the `GetCardSwipe` method was called.
GetCardSwipe	Boolean	The `GetCardSwipe` method can only be used for the IVI eN-Crypt 2100 and will wait for a card to be swiped with the PINpad. If the card is not swiped within the timeout value, a timeout will occur.
GetErrorCode	Long	The `GetErrorCode` method returns an error code if an error was encountered during the use of various methods such as the `Cancel`, `Initialize`, or `GetPin`. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The `GetErrorDesc` method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetExpDate	String	The `GetExpDate` method returns the expiration date that was obtained when the GetCardSwipe method was called.
GetKeySerialNumber	String	Gets the Key Serial Number. Only used for DUPKT encryption.
GetMember	String	The `GetMember` method returns the cardholder's name that was obtained when the `GetCardSwipe` method was called.
GetPin	String	Gets the PIN from the PINpad. This will prompt the user to enter his/her PIN. `GetPin` will then return the encrypted PIN. `GetPin` will return nothing if a timeout or cancel occurs.
GetTrack1	String	The `GetTrack1` method returns the track I data of the card that was obtained when the `GetCardSwipe` method was called. Only for the IVI eN-Crypt 2100. Currently, **PC**Charge does not support any Track 1 data for any processors.
GetTrack2	String	The `GetTrack2` method returns the track II data of the card that was obtained when the `GetCardSwipe` method was called.
Initialize	Boolean	Initializes the PINpad. Returns `TRUE` if the initialization was successful, `FALSE` if not. Initialize has an optional parameter than can be passed in that will allow checking of the com port.

Device.OCX Events

Event	Description - Device.OCX Events

Event	Description - Device.OCX Events
`CardSwipe`	Used only for the Verifone 2000(including SC5000 with 2000 emulator), Omni 3730, Everest. Once the PINpad is initialized and a card is swiped, this event will fire. Once the event fires, call the various get methods (`GetCard`, `GetExpDate`, `GetTrack2`, `GetMember`) to retrieve data from the magnetic stripe of the card.
`Error`	The `Error` event is fired any time an error occurs in the control. Once an `Error` event has fired, call `GetErrorCode` and `GetErrorDesc` to determine what kind of error has occurred. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).

SC550.OCX

The SC550.OCX allows integration to the Verifone SC5000 PINpad. The Verifone SC5000 PINpad is only used when performing Canadian Debit transactions with Global Payments East (NDC).

Refer to the section **Canadian (Interac) Debit Transactions** (see page 113) for more information on using SC550.OCX to integrate Canadian Debit transactions.

SC550.PinSC550 Class Properties

Property Name	Data Type	Description – SC550.PinSC550 Class Properties
Account	String	The bank account type
ActionPending	GPSActionPending	ENUM_ACTION_PENDING_INTERAC ENUM_ACTION_PENDING_MSR ENUM_ACTION_PENDING_NONE ENUM_ACTION_PENDING_PRINTER
AppName	String	This property contains the AppName of the PINpad. This data will be available once the GetSerialBlock method has been executed successfully. **Example:** [GPS/CA/CC/1.0b]. Provided for informational purposes only.
AutoInterval°°	String	Sets how often the PinSC550 class polls for the OMC and IMC (Out MAC and In MAC) files **Format:** Milliseconds **Default Value:** "1000".
AutoProcess°°	Boolean	Determines whether the PinSC550 class will poll for and automatically process the OMC and IMC (Out MAC and In MAC) files in the **PC**Charge directory. **Valid Values:** TRUE – enable automatic processing (recommended) FALSE – disable automatic processing (default)
BatchCode	String	Settlement number
Baud°	String	Baud Rate used to communicate with the PINpad. **Default Value:** "9600"
ChipSN	String	This property will contain the Chip Serial Number of the PINpad. This field is populated by running the GetSerialBlock method. The Chip Serial Number is passed as a parameter to the Debit.OCX's GetPOSSequenceNumber method.
ChipStatus	String	This property contains the Chip Status of the PINpad. This data will be available once the GetSerialBlock method has been executed successfully. **Example:** 00. Provided for informational purposes only.
CommVisible	Boolean	Used to determine whether the PINpad communication monitor is displayed or hidden. Set to TRUE to display the monitor, FALSE to hide the monitor (default). The communication monitor is typically used to troubleshoot an integration to the PINpad.
DataBits°	String	DataBits used to communicate with the PINpad. **Default Value:** "8"
DefaultLanguageCode	String	"English" or "French" for Canadian Debit
DeviceData	String	The property will contain string data returned from the PINpad. This property is populated when performing various PINpad operations with the PINpad. For example, when the Interac response data is returned from PINpad, the data is available in this property.
DeviceState	GPSRespStatus	Returns the state of the PINpad. Valid return values are listed in the **GPSRespStatus Values** table (see below)
FacilityCode	String	This property contains the Facility code of the PINpad. This data will be available once the GetSerialBlock method has been executed successfully. **Example:** 002. Provided for informational purposes only.
FirmVersion	String	This property contains the firmware version of the PINpad. This data will be available once the GetSerialBlock method has been executed successfully. **Example:** 1513. Provided for informational purposes only.
KernelID	String	This property contains the Kernel ID of the PINpad. This data will be available once the GetSerialBlock method has been executed successfully. **Example:** K1N8LE10. Provided for informational purposes only.
KeyType	GPSKeyManagementType	ENUM_DUKPT ENUM_MAC ENUM_MASTERSESSION

Property Name	Data Type	Description - SC550.PinSC550 Class Properties
LanguageCode°	GPSLangCode	Language Code used when communicating with and displaying messages on the PINpad. **Valid Values:** 0 – English (**default**) 1 – French
MacBlock	String	The MAC block returned from the PINpad. Once the RequestMAC method is called, this property will be populated with the MAC Block information that is returned from the PINpad.
Parity°	String	Parity used to communicate with the PINpad. **Valid Values:** E – even O – odd N – None (default)
PinBlock	String	PinBlock information from the PINpad.
PinPadType	String	This property contains the PINpad type. This data will be available once the GetSerialBlock method has been executed successfully. **Example:** 001A. Provided for informational purposes only.
Port°	String	Port Number to be used to communicate with PINpad. **Default Value:** "COM1"
PortState	GPSRespStatus	Returns the state of the PINpad's Port. Valid return values are listed in the **GPSRespStatus Values** table (see below)
ProdData	String	This property contains the Prod Data of the PINpad. This data will be available once the GetSerialBlock method has been executed successfully. **Example:** 020605. Provided for informational purposes only.
ReMacData	String	If ReMACing will need to occur, this property will contain the string that needs to be sent to the PINpad to retrieve the new MAC block. Use the RequestMAC method to request the new MAC block.
RequireReMac	Boolean	If ReMACing is required this property will return TRUE, otherwise, this property will return FALSE. If TRUE, use the RequestMAC method to pass the string that appears in the ReMacData property to retrieve a new MAC block.
RespCode	GPSRespCode	Returns a value that indicates the status of various operations. Valid return values are listed in the **GPSRespCode Values** table (see below)
ResponseAsString	String	Returns a string that is the English text that describes the GPSRespStatus enumerated value or code. Pass, as a parameter, either an enumerated value or code as defined in the **GPSRespStatus Values** table (see below).
ResponseCodeAsString	String	Returns a string that is the English text that describes the GPSRespCode enumerated value or code. Pass, as a parameter, either an enumerated value or code as defined in the **GPSRespCode Values** table (see below).
ROMVersion	String	This property contains the ROM Version of the PINpad. This data will be available once the GetSerialBlock method has been executed successfully. **Example:** 01. Provided for informational purposes only.
ServerPath°°	String	The path to the directory in which the **PC**Charge executable resides. This property must be set prior to setting the AutoProcess property to TRUE. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
TimeOut	String	The Timeout value **Default Value:** "5"

Property Name	Data Type	Description - SC550.PinSC550 Class Properties
TrackII	String	This property contains the Track II data from the card. This data will be available once the customer swipes a card in the PINpad. **Note:** The value will contain the sentinels (";" and "?") that are captured from the magnetic stripe of the card. **Example:** ;1234123412341234=08121234123412340001? **Note:** The value between the ; and = is the card number. The first four digits after the = is the expiration date in YYMM format (**Note:** The expiration date must be sent to **PC**Charge in MMYY format). The last digit (the digit before the ?) is the language code.
VFIBlock	String	Chip Serial Number of the PINpad

° Must be set correctly prior executing the Initialize method

°° Properties used to enable the automatic processing of OMC files.

SC550.PinSC550 Class Methods

Method Name	Returned Value	Description - SC550.PinSC550 Class Methods
Base64Decode	String	Used to Base 64 decode a string that comes back from **PC**Charge. Only used if AutoProcess is disabled (not recommended).
Base64Encode	String	Used to Base 64 encode a string. Use this prior to sending info to **PC**Charge. Only used if AutoProcess is disabled (not recommended).
Cancel	None	Cancels the transaction.
ClosePort	None	Closes the port that the PINpad is connected to.
DispBankResponse	None	Displays the bank response.
DispGenMsg	None	Displays a general message on the PINpad, 3 parameters, string, string, language code
DispObtainCard	None	Displays text message onto the PINpad "Obtain Card'.
DispPrinterDown	None	Displays message onto the PINpad "Printer Down".
GetSerialBlock°	None	Obtains the Chip Serial Number from the PINpad. This method populates the ChipSN property with the PINpad's Chip Serial Number.
Initialize°	None	Initializes PINpad. The properties in the **SC550.PinSC550 Class Properties** table that are marked with a ° must be set correctly prior to calling the OpenPort and Initialize methods.
InteracAnalysis	None	If the AutoProcess property is set to FALSE (meaning that the integrator has chosen to manually process OMC and IMC files—NOT recommended), this method is used to verify the transaction response from the processor. If AutoProcess is set to TRUE, this method is not needed to process transactions. This method sends the data contained in the OMC file in the **PC**Charge directory to the PINpad for verification. Two parameters, the string from OMC file, and language code must be passed when calling this method. Before passing the string to this method, it must be decoded using the Base64Decode method.
LoadKey	None	Used to load the key into the PINpad. Pass the key as a parameter.
OpenPort°	None	Enables the port for PINpad. This method must be called first. The properties in the **SC550.PinSC550 Class Properties** table that are marked with a ° must be set prior to calling the OpenPort and Initialize methods.
RequestInterac	None	This method is used to instruct the PINpad to prompt the customer to confirm the amount of the transaction, indicate the tip amount, indicate the bank account type, and enter their PIN. This method requires two parameters: The **Interac request string** (the string returned from clsInteracReq.BuildInteracRequest method) and the **Language code** of the card. Once the customer enters the data, the ActionEvent event will fire indicating the PINpad has returned the transaction-specific data such as the MAC block, the PIN block, the bank account type, and the tip amount.

Method Name	Returned Value	Description – SC550.PinSC550 Class Methods
RequestMAC	None	If ReMACing is required, this method is used to request a new MAC block from the PINpad. This method requires a parameter, the MAC data portion of the Interac request string, to be sent as a variable. Once the PINpad returns the new MAC block, the MacBlock property will be populated with the new MAC block value. Use the RequireReMac property in the PinSC550 class to determine if ReMACing must occur. If ReMACing is required, pass the data in the ReMacData property as a variable when using this method.
StartMSR	None	Sends a command to the PINpad to prompt the customer to swipe their card. Once the card is swiped, the PINpad will read the magnetic data on the debit card, and then populate the TrackII property with the track II data from the card.

° Must be called prior to executing the StartMSR method

SC550.PinSC550 Class Events

Event	Description – SC550.PinSC550 Class Events
ActionUpdate	The ActionUpdate event will fire when the state of PINpad changes. Once the ActionUpdate event has fired, two values, eAction and eResp, are set and can be used to determine the current state of the PINpad. EAction will contain one of the values defined in the **GPSPinPadAction Values** table (see below) and eResp will contain one of the values defined in the **GPSRespStatus Values** table (see below). The value returned in eResp can be passed to the ResponseAsString method to provide an English description of the code.

GPSPinPadAction Values

Code	Enumerated Value – GPSPinPadAction Values
1	ENUM_ACTION_PORT_STATE_CHANGE
2	ENUM_ACTION_INIT_PINPAD
3	ENUM_ACTION_MSR_START
4	ENUM_ACTION_MSR_STOP
5	ENUM_ACTION_REQ_SERIAL
6	ENUM_ACTION_DEVICE_STATE_CHANGE
7	ENUM_ACTION_MSR_RECEIVED_DATA
8	ENUM_ACTION_MSG_DISP_GEN
9	ENUM_ACTION_MSG_DISP_OBTAIN
10	ENUM_ACTION_MSG_DISP_BANKRESP
11	ENUM_ACTION_MSG_DISP_PRINTERDOWN
12	ENUM_ACTION_PTR_RECEIVED_DATA
13	ENUM_ACTION_DEVICE_ACTION_CHANGE
14	ENUM_ACTION_REQ_MAC
15	ENUM_ACTION_REQ_INTERAC
16	ENUM_ACTION_INTERAC_RECEIVED_DATA
17	ENUM_ACTION_ANALYSIS_INTERAC
18	ENUM_ACTION_KEY_LOAD
19	ENUM_ACTION_AUTOPROCESS_CHANGE

GPSRespCode Values

Code	Enumerated Value – GPSRespCode Values
-1	ENUM_RCODE_CLEAR
0	ENUM_RCODE_SUCCESSFUL
1	ENUM_RCODE_UNSUCCESSFUL
2	ENUM_RCODE_TIMEOUT
3	ENUM_RCODE_CANCELED
4	ENUM_RCODE_CORRKEY
6	ENUM_RCODE_INVALID_ACCTLEN
7	ENUM_RCODE_MAC_NOVERIFY
9	ENUM_RCODE_MAC_NOBLOCK
11	ENUM_RCODE_KEY_DECRYPT_ERROR
13	ENUM_RCODE_KEY_LOCATION_ERROR
14	ENUM_RCODE_KEY_MTK_NOSELECT
16	ENUM_RCODE_KEY_LOAD_FAIL_REVERT
98	ENUM_RCODE_KEY_LOAD_FAIL_NOREVERT

GPSRespStatus Values

Code	Enumerated Value - GPSRespStatus Values
-35	ENUM_STATUS_AUTOPROCESS_ON
-34	ENUM_STATUS_AUTOPROCESS_OFF
-30	ENUM_STATUS_PENDING_NONE
-29	ENUM_STATUS_PENDING_MSR
-28	ENUM_STATUS_PENDING_PRINTER
-27	ENUM_STATUS_PENDING_INTERAC
-20	ENUM_STATUS_DEVICE_IDLE
-19	ENUM_STATUS_DEVICE_PROCESS
-18	ENUM_STATUS_DEVICE_CANCEL
-17	ENUM_STATUS_DEVICE_WAITING
-16	ENUM_STATUS_DEVICE_INPUT_PENDING
-10	ENUM_STATUS_PORT_OPEN
-9	ENUM_STATUS_PORT_CLOSED
0	ENUM_STATUS_OK
1	ENUM_STATUS_INVALID_ACTION
2	ENUM_STATUS_DEVICE_INPROCESS
3	ENUM_STATUS_DEVICE_TIMEOUT
4	ENUM_STATUS_UNKNW_ERROR

SC550.clsInteracReq Class Properties

Property Name	Data Type	Description – SC550.clsInteracReq Class Properties
Account°°	String	After the ParseResponseData method has been called, this property will contain the customer-specified bank account type. Possible return values: A – Chequing B – Savings If this customer-specified bank account type is different that the bank account type specified in the AccType when building the Interac Request, a ReMAC must occur. Use the RequireReMac property in the PinSC550 class to determine if ReMACing must occur.
AccType°	String	The customer's bank account type used when processing the transaction. 1 or ENUM_ACC_CHEQ – Chequing 2 or ENUM_ACC_SAV – Savings **Note:** When populating this property to build the initial Interac request string, the customer's bank account type will not be known by the merchant (the customer will enter it once prompted). This value must be hard-coded. It is suggested to hard-code this value to "1" if most of the merchant's customers use their checking accounts when purchasing products or "2" if most of the merchant's customers use their saving account when purchasing products. **Note:** A new MAC value must be requested from the PINpad if the account chosen by the merchant differs from the account chosen by the customer. Use the RequireReMac property in the PinSC550 class to determine if ReMACing must occur.
Amount°	String	Set this property to the Amount of the transaction to be processed. This property should not contain any decimals or commas. **Example:** to specify a $1.00 transaction, set this property to "100"
DisplayAmount°	String	Set this property to the amount of the transaction. The amount specified is displayed to the customer for confirmation on the PINpad. This amount should include the decimal point and trailing zeros, if applicable. **Example:** to specify a $1.00 transaction, set this property to "1.00"
MacBlock°°	String	After the ParseResponseData method has been called, this property will contain the MAC block information that was returned from the PINpad.
PinBlock°°	String	After the ParseResponseData method has been called, this property will contain the customer's encrypted PIN that was returned from the PINpad.
SequenceNum°	String	The POS Sequence number. Set this property to the value that was returned by **PC**Charge when calling the GetPOSSequenceNumber method in the Debit.OCX control.
TermID°	String	Set this property to the Chip Serial Number of the PINpad. Use the GetSerialBlock method in the PinSC550 class to acquire the Chip Serial Number of the PINpad.
TipAmount°°	String	After the ParseResponseData method has been called, this property will contain the optional customer-specified tip amount. If this property is populated with a value greater than 0, a ReMAC must occur. Use the RequireReMac property in the PinSC550 class to determine if ReMACing must occur.
TipType°	Integer	Determines if the PINpad will prompt the customer to enter a tip amount and also determines if the PINpad will display a "suggested" tip amount prior to prompting for the tip amount. **Valid Values:** 0 – The customer is not prompted to enter a tip amount (Default) 1 – The customer is prompted to enter a tip amount. 2 through 99 – The PINpad will calculate a "suggested" tip amount and display it on the PINpad prior to prompting the customer for the tip amount. The integer value passed in represents the percentage that will be used to calculate the tip. For example, 20 would calculate a 20% tip amount. The customer would see this "suggested" tip amount and would then be prompted to key in the actual tip amount.

Property Name	Data Type	Description - SC550.clsInteracReq Class Properties
TrackII°	String	Set this property to the track II data from the magnetic stripe of the card. For example: ;1234123412341234=08121234123412340001?
TransCode°	GPSTransCode	Set this property to the type of transaction being processed. This should be set to the same transaction type specified in the TransNameID property. 0 or ENUM_TCODE_PURCH_NORM – Purchase (default) 4 or ENUM_TCODE_REFUND – Refund
TransNameID°	GPSTransID	Set this property to the type of transaction being processed. The property indicates to the PINpad to displays the type of transaction being processed to the customer. This should be set to the same transaction type specified in the TransCode property. **Valid Values:** 0 or ENUM_TID_PURCH – Displays "PURCHASE" (default) 1 or ENUM_TID_REFUND – Displays "REFUND"
VFISerial°°	String	After the ParseResponseData method has been called, this property will contain the PINpad's Chip Serial Number.

° These properties must be set prior to calling the BuildInteracRequest method.
°° These properties will be set after the ParseResponseData method completes successfully.

SC550.clsInteracReq Class Methods

Property Name	Returned Value	Description - SC550.clsInteracReq Class Methods
BuildInteracRequest	String	Builds and returns the Interac request string that will be sent to the PINpad. The properties in the **SC550.clsInteracReq Class Properties** table that are marked with a ° must be set prior to calling this method.
BuildRemacData	*String*	*Used internally*
ClearData	None	The ClearData method will clear the values in all properties.
ParseResponseData	Boolean	Parses the Interac response string returned by the PINpad. The response string is returned by the PINpad in the DeviceData property of the PinSC550 class. If the string is parsed successfully, TRUE is returned, FALSE otherwise. When the string is parsed successfully, the properties marked with a ° in the **SC550.clsInteracReq Class Properties** table will be populated with the customer entered data.

SC5X.OCX

The SC5X.OCX allows integration to the Verifone SC5000 PINpad. The Verifone SC5000 PINpad is only used when performing Canadian Debit transactions with Chase Paymentech (GSAR).

Refer to the section **Canadian (Interac) Debit Transactions** (see page 113) for more information on using SC5X.OCX to integrate Canadian Debit transactions.

SC5X.OCX Properties

Property Name	Data Type	Description – SC5X.OCX Properties
Action	String	Transaction Action Code
Amount	String	Transaction Base Amount
Baud°	String	Baud Rate used to communicate with the PINpad. **Default Value:** "9600"
Command	String	Transaction Action Code
CommMethod	Integer	Defines the Method that the OCX will send the transaction to PCCharge 0 – INX File Method 1 – TCP/IP (For Future Use)
DataBits°	String	DataBits used to communicate with the PINpad. **Default Value:** "8"
Parity°	String	Parity used to communicate with the PINpad. **Valid Values:** E – even O – odd N – None (default)
PinPadTimeout	String	Timeout value for the PinPad **Default Value:** "5"
Port°	String	Port Number to be used to communicate with PINpad. **Default Value:** "COM1"
IPAddress	String	The IP address of PCCharge. Only used when using the TCP/IP portion of the SC5X.OCX – For Future Use
TcpIPPort	String	The port used to communicate with PCCharge. Only used when using the TCP/IP portion of the SC5X.OCX – For Future Use
ServerPath°°	String	The path to the directory in which the **PC**Charge executable resides. This property must be set prior to setting the AutoProcess property to TRUE. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
SurchargeAmount	String	Surcharge Amount assigned to the transaction. This is inputted for each transaction and is not to be included in the amount.
SwipeTimeout	String	The amount of time the pinpad will wait for a card swipe. **Default Value**: 255 (seconds)
Ticket	String	Ticket
TimeOut	String	The Timeout value for the Transaction **Default Value:** "90"
RequestTip	Boolean	Used to determine if PINpad will prompt for gratuity. Only used with the S21 action code. **Valid Values:** True - Prompt for gratuity False - Do not prompt for gratuity
RequestCashBack	Boolean	Used to determine if PINpad will prompt for cash back. Only used with the S21 action code. **Valid Values:** True - Prompt for cash back False - Do not prompt for cash back
CommVisible	Boolean	Used to determine whether the PINpad communication monitor is displayed or hidden. Set to TRUE to display the monitor, FALSE to hide the monitor (default). The communication monitor is typically used to troubleshoot an integration to the PINpad. **Note:** Only for use in development environment.

° Must be set correctly prior executing the Initialize method

SC5X.OCX Methods

Method Name	Returned Value	Description – SC5X.OCX Methods
Cancel	None	Cancels the transaction.
CancelPin	None	Cancels the pin pad request
Initialize°	None	Initializes PINpad. The properties in the **SC5X.OCX Class Properties** table that are marked with a ° must be set correctly prior to calling the OpenPort and Initialize methods. The initialize command will attempt to initialize the PINpad 3 times prior to returning a response.
GetResult	String	Returns the result, which indicates the transaction's status upon completion.
GetApproved	Boolean	The GetApproved method returns TRUE if **PC**Charge returns "APPROVED" as the result of the transaction. Otherwise, FALSE will be returned.
GetAuth	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
GetCaptured	Boolean	The GetCaptured method returns TRUE if **PC**Charge returns "CAPTURED" as the result of the transaction. Otherwise, FALSE will be returned. The GetCaptured method is used to determine if a transaction that will result in a monetary transfer (Sale, Credit, Post-Authorization, etc.) is approved or declined. A "CAPTURED" response indicates that the transaction has been approved, and that the transaction has been placed in the open batch.
GetCashBack	String	Returns the cash back entered by the card holder.
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetReceipt	String	Returns the required receipt data, including tip amount.
GetRefNumber	String	Returns the reference number associated with the transaction. The reference number is used to help identify the transaction and is useful for the cardholder and merchant when doing research. This value is not returned with all transactions.
GetTipAmount	String	Returns the tip amount entered by the card holder.
GetTroutD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** for more information.
GetXMLResponse	String	Returns the Raw XML from the oux response
OpenPort°	None	Enables the port for PINpad. This method must be called first. The properties in the **SC5X.OCX Class Properties** table that are marked with a ° must be set prior to calling the OpenPort and Initialize methods.
Shutdown	None	Closes the port and shuts down the OCX timer.

Method Name	Returned Value	Description - SC5X.OCX Methods
Send	None	This method communicates with the Pinpad. Card number and expiration date are not required to be assigned to the pinpad before calling this method. This method requires the following properties of the OCX to be set: • MerchantNumber • Processor • Action/Command • Amount • Path The following properties are optional but must be set if used. • Ticket • CashBack • SurchargeAmount • CommMethod • Language • TroutD • Timeout • Baud • Parity • PinPadTimeout **Example:** `Send 3`
RetrieveCreditSwipe	Boolean	Once this is called, the Pinpad will have the MSR function called. The merchant/customer swipes the card, then the pinpad gets reset to "ready". If successful, the function returns true. Then, the integrator can call: • GetCard • GetMember • GetExpDate • GetTrack If unsuccessful, the function returns false. Then, the integrator can call: • GetErrorCode • GetErrorDesc
RetrieveGiftCardSwipe	Boolean	This is the same as `RetrieveCreditSwipe`, yet the processor code must be passed in. **Example:** .RetrieveGiftCardSwipe("VLNK"). Once this is called, the Pinpad will have the MSR function called. The merchant/customer swipes the card, then the pinpad gets reset to "ready". If successful, the function returnes true. Then, the integrator can call: • GetCard • GetMember • GetExpDate • GetTrack If unsuccessful, the function returns false. Then, the integrator can call: • GetErrorCode • GetErrorDesc

SC5X.OCX Error Codes

ErrorCode	ErrorDescription	Description
-1	Port Open Error	The OCX could not open the port to the pinpad
-2	Invalid Command	The Command is not currently supported by the ocx
-3	Invalid Amount	The amount was not set or was in an incorrect format
-4	Invalid Baud	The baud was not set or was in an incorrect format
-5	Invalid Port	The pinpad port was not set or was in an incorrect format
-6	Invalid Parity	The pinpad parity was not set or was in an incorrect format
-7	Invalid Databits	The pnpad databits were not set or was in an incorrect format
-8	Invalid Path	The Path was not set
-9	Invalid Processor	Processor is not supported for Interac with this OCX (currently only Chase Paymentech)
-10	Invalid Merchant Number	Merchant Number is not a valid Merchant Number for the selected processor (currently only Chase Paymentech)
-11	Invalid Card	Credit Only. This card does not pass the Luhn Check.
-12	Invalid Expiration Date	Expiration Date is expired or invalid
-13	Chip Serial Number Error	The OCX could not obtain the Chip Serial Number from the Pinpad
-14	Key Change Request Was Required. Retry Transaction.	A Current Key Request was required and has finished. Continue with the original transaction.
-15	Card Swipe Error	The card swipe did not return the correct information
-16	Confirm Amount Error	There was an error in the Confirm Amount dialogue of the pinpad.
-17	Surcharge Was Rejected	The customer rejected the surcharge amount.
-18	Pin Error	There was an error obtaining the Pin Block
-19	Initialization Error	The OCX was unable to initialize the pinpad
-20	Invalid Swipe Timeout	The passed in Swipe Timeout exceeds 255 seconds. This is the length of time that the pinpad will sit at "Swipe Card"
-21	Card Swipe Timeout	Card was not swiped within timeout value

Notes: In order to test Canadian Debit with Chase Paymentech (GSAR), the integrator will need to obtain a test merchant account directly from Chase Paymentech (GSAR) and a VeriFone SC5000 PINpad that is configured properly for use with Canadian Debit and the Chase Paymentech (GSAR) test merchant account. MAC data is specific to the PINpad and merchant number. If EBT transactions will be supported, a separate PINpad device is required.

Reporting

The Charge.OCX control may be used by integrators to submit report requests. A report request can have **PC**Charge print a report to it's default report printer or have **PC**Charge generate a file containing the report output. If generating a file, the **PC**Charge reporting interface supports three different file types:

1. Portable Document Format (.pdf)
2. Rich Text Files (.rtf)
3. Standard Text files (.txt)

Note: The reporting interface cannot be configured to send reports directly to the screen.

The following outlines the properties used for submitting report requests to PCCharge with the Charge.OCX control. The properties in Charge.OCX that are not documented below should be left blank when submitting report requests.

Property	Data Type	Description – Charge.OCX Reporting Properties
Action°	Long	The action code that identifies what type of report will be requested. **Valid Values:** 81-84. **Example:** If running a credit card detail report, set the action code to "81". Consult the section **DevKit Constants** for a list of valid values (see page 143).
Card	String	**User name filter**. If a valid user name is set in the Card property, the report will be filtered by that user name. The report returned will consist of only those transactions processed by the user name specified. **Example:** "User1". If this property is left blank, the report will show transactions processed by all users.
CheckCard°	Boolean	Flag that indicates whether to activate credit card validity testing. **Valid Values:** TRUE; FALSE. **Default value:** TRUE. **This value must be set to FALSE when submitting a report request.**
CommMethod	Enum	Specifies which communication method will be used. 0 – File_Transfer 1 – TCP/IP Please refer to page 20 for a description of these methods. If TCP/IP is selected, the IPAddress, Port and EnableSSL properties must also be set. If File_Transfer is set then the Path property must be set.
EnableSSL	Boolean	**For use with TCP/IP CommMethod only**. SSL is available starting with PCCharge version 5.8. Set to True to enable SSL Socket integration. **Note**: See SSL Information on page 70 for more information.
IPAddress	String	**For use with TCP/IP CommMethod only.** IPAddress of machine where PCCharge is running. Defaults to 127.0.0.1
Manual	Long	**Result filter**. Use this filter to create a report consisting of only those transactions with the result specified. **Valid Values:** 0 = all (default), 1 = approved, 2 = declined **Example:** 1
Member	String	**Ending Date/Time filter**. Specifies the end date and end time of the report. **Format**: Date: MM/DD/YY Time: HH:MM:SS PM. When used in conjunction with Street; will create a report consisting of only those transactions processed between the start and end date/time specified (inclusive). If an end date is not specified, today's date is assumed. If an end time is not specified, 11:59:59 PM is assumed. The end date can be passed without the end time. However, the end time cannot be passed without the end date. **Examples:** "07/06/05 06:00:00 PM" or "07/06/05"
MerchantNumber	String	**Merchant Number filter**. Set this property to filter the report by the merchant number specified. Setting this property will generate a report consisting of only those transactions processed via the merchant number specified. To generate a report that includes all merchant numbers in **PC**Charge, set this property to "ALL" or leave blank. **Example:** "99999999911"

Property	Data Type	Description - Charge.OCX Reporting Properties
Path°	String	**For use with File_Transfer CommMethod only**. The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ (**default**) **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
PeriodicPayment	String	**Report Output setting**. Determines if the report will be printed by **PC**Charge or written to a file. **Valid Values:** "0" = print to default printer specified in **PC**Charge (default). "1" = print to file using filename specified in TransID and path specified in TRACK.
Port	String	**For use with TCP/IP CommMethod only.** Open port of PCCharge.
Street	String	**Starting Date/Time Filter** (Optional) Specifies the start date and start time of the report. **Format**: Date: MM/DD/YY Time: HH:MM:SS PM. Use to create a report consisting of only those transactions processed on or after the date specified. If a start date is not specified, today's date is assumed. If a start time is not specified, 12:00:00 AM is assumed. The start date can be passed without the start time. However, the start time cannot be passed without the start date. **Examples:** "03/04/05 09:00:00 AM" or "03/04/05"
TimeOut	Long	The number of seconds after which a timeout error will be returned from the control. The count will start when the Send method is called. The default timeout value is 90 seconds. It is highly recommended that integrators review the section **Timeouts** (see page 59).
Track	String	**Destination Directory for Report File**. Specifies the destination directory where the report file will be generated by **PC**Charge (if PeriodicPayment is set to "1"). **Example:** "C:\My Documents\PCCReports\" **Path Formats:** UNC, MS-DOS(8 Characters) and Long. **Max Length:** 40 characters (if the Destination Directory is longer than 40 characters, use CustCode for the additional characters. Must end with a "\" unless the directory name will be continued in the CustCode property. **Note:** If running in a Client/Server environment, this property is the path from the server running **PC**Charge, not the client. For example, if a client submitted a report request that specified "C:\" as the destination directory, the report would be written to the local hard drive of the server running **PC**Charge, not to the client's hard drive.
CustCode	String	**Destination Directory for Report File (continued)**. Continuation of the destination directory (if the directory name is greater than 40 characters). **Max Length:** 25 characters. Must end with a "\"
TransID	String	**Report File name/Report File Type**. Specifies the filename and extension of the report file generated by **PC**Charge (if PeriodicPayment is set to "1"). Also determines what file type will be used when **PC**Charge generates the report. To specify the file type, set the extension to one of the following: **.pdf** – Create the report file in the Portable Document Format. **Ex.** Report.pdf **.rtf** – Create the report file in Rich Text. **Ex.** Report.rtf **.txt** – Create a report file in flat text. **Ex.** Report.txt **Default: .txt** (If an extension other than the ones listed is passed, the report will be returned as flat text and a .txt extension will be added to the filename)
User°	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).

° These properties are required to submit a report request.

The following outlines the methods used to process report requests. The methods in `Charge.OCX` that are not documented below will not be used when processing report requests.

Method	Returned Value	Description - Charge.OCX Reporting Methods
`Abort`	Boolean	The `Abort` method attempts to cancel the transaction in progress and will return a Boolean value that indicates whether or not the transaction was canceled. **Note:** This method is not available when integrating using FoxPro. Use the `Cancel` method instead.
`About`	MsgBox	The `About` method will display the About box associated with the control.
`Cancel`	None	The `Cancel` method attempts to cancel the transaction in progress. Calling the `Cancel` method does not guarantee that the transaction will be canceled; it simply attempts to cancel the transaction.
`Clear`	None	The `Clear` method will clear the values in all properties and methods. It is recommended that this method be called: a. after the transaction results have been retrieved by using the various `.get` methods b. after the `DeleteUserFiles` method has been called c. prior to running the next transaction
`DeleteUserFiles`	None	The `DeleteUserFiles` method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the `User` property. The `DeleteUserFiles` method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files, the `Error` event will be triggered and the `GetErrorDesc` method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150). For use only with File_Transfer CommMethod
`GetAuth`	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
`GetErrorCode`	Long	The `GetErrorCode` method returns an error code if an error was encountered during the use of various methods such as the `Send`, `Cancel`, `DeleteUserFiles`, and `PccSysExists`. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
`GetErrorDesc`	String	The `GetErrorDesc` method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
`GetResult`	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
`PccSysExists`	Boolean	The `PccSysExists` method is used to determine if **PC**Charge is available to process transactions. If `PccSysExists` returns `TRUE`, the file `SYS.PCC` exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. `TRUE` usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The `GetErrorCode` and `GetErrorDesc` methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If `PccSysExists` returns `FALSE`, then **PC**Charge is ready to process transactions. For use only with File_Transfer CommMethod

Method	Returned Value	Description - Charge.OCX Reporting Methods
Send	Integer	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the control will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has an optional parameter that indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send 3 **Note:** The other values that appear in the enumerated list are for internal use only-- do not attempt to use any values other than the ones listed above.
VerifyMerchantNumber	Boolean	The VerifyMerchantNumber method returns TRUE if the merchant number that is passed to it is set up in **PC**Charge, otherwise, FALSE is returned. Specifically, this method checks for the merchant number in the file TID.PCC, which is located in the **PC**Charge directory. The Path property must be set before calling this Method.

Charge.OCX Events

Event	Description - Charge.OCX Events
Error	The Error event is fired any time an error occurs in the control. Once an Error event has fired, call GetErrorCode and GetErrorDesc to determine what kind of error has occurred. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
Finish	The Finish event will fire when the transaction has been completed. This means that **PC**Charge has processed the transaction successfully and has placed a file with the extension of .oux in the **PC**Charge directory. The name of the .oux file will be what was set in the User property of the transaction request. Call the GetResult method to determine whether or not the transaction was approved. A list of valid results can be found in the **Transaction Result Constants** section (see page 156).

Note: When doing asynchronous transactions in an event-driven programming model, it is important to place all result or error routines in either the Finish or Error events. Do not place any code that uses the .get methods after invoking the Send method.

DLL (ActiveX) Method

A DLL called `PSCharge.dll` is included in the DevKit. This DLL allows developers to access various processing functions using any Windows-based programming environment that supports referencing ActiveX DLLs.

Before `PSCharge.dll` can be used, it must be added as a reference in the programming environment. For example, in Microsoft Visual Basic 6, follow the procedure below:

1. Choose **Project** | **References** from the Visual Basic menu bar.

2. After the References window opens, from the list, scroll to and check the box next to:

    - **VeriFone's ActiveX Payment Server DLL**

    and click **OK**. (The DevKit installation registers the `PSCharge.dll` by default. If `PSCharge.dll` is not yet registered on the system, use `regsvr32.exe` to register it, or use the Browse button on the References window to register `PSCharge.dll` and add it as a reference to the project).

Once registered, `PSCharge.dll` provides the developer with several classes to allow the integration of payment processing:

- **`Charge`** – contains properties and methods for credit card processing
- **`Debit`** – contains properties and methods for debit and EBT processing.
- **`Check`** – contains properties and methods for check processing.
- **`Gift`** – contains properties and methods for gift/loyalty processing.
- **`Batch`** – contains properties and methods for performing end-of-day inquiry and batch settlement.
- **`Offline`** – contains properties and methods for processing offline transactions.

The properties and methods of the DLL's various classes can be viewed through the object browser. If MS VB6 is not being used, refer to the language documentation for instructions on using ActiveX DLLs.

Note: The additional classes in `PSCharge.dll` that do not appear in the list above are currently not supported for transaction processing.

Using PSCharge.dll

Create an instance of any of the DLL's classes by using the following line of code:

```
Set <instance name>  = New <object name>
```

For example, to create an instance based on the `Charge` class, the following line of code would be used in MS VB6:

```
Set Charge = New PSCharge.Charge
```

Charge Class

The `Charge` class provides integrators with properties and methods used to submit credit card transactions to **PC**Charge. To use the `Charge` class to integrate transaction processing, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to process transactions by using the `PccSysExists` method.

2. Assign the appropriate values to the properties required for the transaction to be performed and validate the values using the various `.Verify` methods. (The properties marked with a ° in the **Charge Class properties** table are the minimum required to process a Sale or Pre-Authorization transaction.)

3. Call the `Send` method. (**Note:** When calling the `Send` method, it is recommended that "3" is passed as a parameter to activate the XML message format)

4. Wait for the transaction to process and then call the various `.Get` methods to determine the outcome of the transaction (code using the `.Get` methods may be placed immediately after the `Send` method). The most important information can be acquired by calling the `GetResult` and `GetAuth` methods. If an error occurs, call the `GetErrorCode` and `GetErrorDesc` methods to determine the nature of the error.

5. Call the `DeleteUserFiles` method to delete all files related to the transaction.

6. Call the `Clear` method to reset all the properties and methods related to the transaction or destroy the object.

Consult the section **Pseudo-code** (see page 158) for various examples that may be followed when using the `Charge` class to perform transaction processing.

Note: **PC**Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

Charge Class Properties

Property	Data Type	Description - Charge Class Properties
Action°	Long	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
Amount°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
AmxChargeDescription	String	The American Express Charge Description. This is a general description describing merchandise: the AMEX representative and the merchant will decide on an appropriate description. **Note:** Only Required for Retail, MOTO and Restaurant transactions when using AMEX direct settlement or TSYS **Max Length:** 23 bytes
AmxDescription_1	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AmxDescription_2	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AmxDescription_3	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AmxDescription_4	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AuthCode	String	The Authorization code. This value is returned by the issuing bank and should only be set in a transaction request if processing a Post-Authorization and the Post-Authorization is being used to add a Voice-Authorization to the batch or to "store" a Voice-Authorization. (For information on stored Voice-Authorizations, see page 91). The AuthCode property does not need to be set if the Post-Authorization completes a standard Pre-Authorization using the TroutD value of the Pre-Authorization. See the section **Follow On Transactions** for more information (see page 70).
Billpay	String	Only valid for Visa debit and credit transactions. It is used to indicate the transaction is being ran for payment of a bill (ultilty, monthly gym dues, etc.) **Valid values:** 0 – Non-Bill payment transaction 1 – Bill payment transaction

Property	Data Type	Description - Charge Class Properties
Card°	String	The credit card number that will be used when processing the transaction. **Max Length:** 20 characters. **Example:** 5424180279791765 **NOTE:** Is it the integrator's responsibility to remove spaces in the card number if there are spaces in the Track II card number data.
CardPresent	String	**For Retail or Restaurant transactions**: Flag that indicates whether the card was present. **For eCommerce transactions**: Flag that indicates what type of transaction occurred. **Valid values:** 0 = Card not present, 1 = Card present (for Retail, MOTO, or Restaurant); D = Digital goods, P = Physical goods (for eCommerce)
CheckCard	Boolean	Flag that indicates whether to activate credit card validity testing. **Valid Values:** TRUE; FALSE. **Default value:** TRUE. This value must be set to FALSE when performing Follow on transactions such as Voids or Gratuities because the card number is omitted from these transaction requests.
Command	String	The action code that identifies what type of transaction will be performed. **Valid Values:** 1-10, 13-15, ZI, ZH. **Example:** If running a credit card sale, set the action code to "1". Consult the section **DevKit Constants** for a list of valid values (see page 143). **Note:** Because the Action property is defined as "long", this property was added to allow action codes that contain strings (such as Transaction Inquiry - ZI). If the Command property is set, it's value will override the value set in Action.
CommercialCardFlag	String	The type of commercial card being submitted. The getCommercialCardType method should be used to retrieve the 1 character value from **PC**Charge that indicates what type of commercial card will be submitted. See the section **Commercial Card Transactions** (see page 93) for more information. **Max Length**: 1 character **Valid values:** B – Business P,L,G -- Purchase C – Corporate F – Fleet
CommMethod	Enum	Specifies which communication method will be used. 0 – File_Transfer 1 – TCP/IP Please refer to page 20 for a description of these methods. If TCP/IP is selected, the IPAddress, Port and EnableSSL properties must also be set. If File_Transfer is set then the Path property must be set.
CustCode	String	Customer code for purchasing/commercial cards. This property must be set for commercial card transactions in order to get the best discount rate. Additionally, the transaction's action code must indicate that the transaction is a commercial card transaction. **Note:** Global East (NDC), terminal based, requires the customer code be all upper case. **Max Length:** 25 characters, alphanumeric only.
CreditPlanNumber	String	The credit plan number, only applicable when using Citi as the processor for private label cards.
CVV2	String	The CVV2 value for the transaction. The card verification value (CVV2 for Visa, CVC2 for MasterCard, and CID for AMEX and Discover) is a 3 or 4 digit number that is embossed in the signature panel for Visa, MasterCard, and Discover and on the front of the card for AMEX. All AMEX cards utilize a 4 digit CID. **Max Length:** 4 characters. CVV2 should only be passed on non-swiped transactions.
Demo	Boolean	The demo mode flag. In demo mode, a simulated response is returned in which even amounts return approved, and odd amounts return declined. **Valid Values:** TRUE – Activates demo mode FALSE – Deactivates demo mode (default)
DEST_ZIP_CODE	String	Destination Zip Code for American Express purchasing/commercial cards. This property must be set for American Express commercial card transactions when using American Express as the processor (or via split dial) in order to get the best discount rate. Additionally, the transaction's action code must indicate that the transaction is a commercial card transaction.

Property	Data Type	Description - Charge Class Properties
DriverID	String	Driver identification field. Only required for Wright Express, Voyager and Fleet One cards.
DriverPIN	String	Driver personal identification number. Only required for Fuelman cards.
EstGratuityAmount	String	**For use with Restaurant transactions only.** The estimated gratuity amount for a Sale (action code 1) or Pre-Authorization (action code 4) transaction. If the EstGratuityAmount is populated, **PC**Charge will submit the sum of the values in the Amount and EstGratuityAmount fields for authorization. If the transaction is authorized, only the value in the Amount field will be placed in the **PC**Charge settlement file (if running a Sale). By using the EstGratuityAmount, the merchant can help ensure that the customer has enough available credit on their card to leave a tip. Once the customer indicates the amount of the tip that will be left, a gratuity transaction (action code 13) must be performed on the sale prior to settlement in order to add the actual gratuity to the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. See the section **Restaurant Transactions** (see page 105) for more information. **Note:** It is recommended to check with the processor or merchant service provider for guidance on what amount to set this value to. Incorrectly setting this value can result in downgrades.
ExpDate°	String	The expiration date associated with the credit card number that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208
EnableSSL	Boolean	**For use with TCP/IP CommMethod only**. SSL is available starting with PCCharge version 5.8. Set to True to enable SSL Socket integration. **Note**: See SSL Information on page 70 for more information.
GratuityAmount	String	**For use with Restaurant transactions only.** The actual gratuity amount for a Sale with Gratuity (action code 14) , Gratuity (action code 13) , or Post-Authorization (action code 5) transaction. See the section **Restaurant Transactions** (see page 105) for more information.
IDNumber	String	Only required for Voyager cards, dependant on Restriction Code. Four to six digits. **Note:** Only used for Pre-Authorization transactions **Update:** Beginning with PCCharge version 5.8, this is also used for Citi© Private Label.
Index	Long	The Merchant Number index. If Index is set to a value greater than 0, the Charge class will access the file tid.pcc file and use the merchant number at that index in the file. Index and Path should be set prior to calling the GetCompanyCity, GetCompanyName, GetCompanyState, GetCompanyStreet, or GetCompanyZip methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
IPAddress	String	**For use with TCP/IP CommMethod only.** IPAddress of machine where PCCharge is running. Defaults to 127.0.0.1
ItemID	String	The Item ID for the transaction. This field is only used for Chase Paymentech (GSAR) and can store five (5) four-digit codes that are defined by Chase Paymentech. **Example:** If the ItemID is set to 0001000020003000400005, it stores 5 item IDs (0001, 0002, 0003, 0004, and 0005). These numbers must be obtained from Chase Paymentech.
LastValidDate	String	The last year that will be considered a valid expiration date. Currently, the default value in the charge class is "09". It is recommended that a setting is provided by which the end-user can change this property; otherwise, in the future, end users will require a new PSCharge.dll to be distributed to resolve expiration date issues. **Length:** 2 digits. **Format:** YY **Example:** If LastValidDate is set to 05, then cards between 06 and 99 are considered to be 1906 to 1999, and cards between 00 and 05 are 2000 to 2005.
Manual°	Long	Flag that indicates whether the transaction was manually entered or swiped. If the transaction was swiped, the Track property must also be set. **Valid values:** 0 = manual transaction, 1 = swiped transaction

Property	Data Type	Description - Charge Class Properties
MCSC	String	The Multiple Count Sequence Count. This is the total number of installments that will be charged in a non-restaurant recurring billing scenario. **Max Length**: 2 characters. **Example:** If there are 5 payments to be made, set this property to "5".
MCSN	String	**In a restaurant environment:** The server or cashier id. **Max Length:** 2. This field should be passed for reporting and reconciliation purposes. See the section **Restaurant Transactions** (see page 105) for more information. **Processor specific note:** The Server ID is required for AMEX card transactions. Also required when using the processor NB and GSAR in restaurant business type. **In a non-restaurant environment**, this field is the Multiple Count Sequence Number. This is the transaction number within the total number of payment installments in a recurring billing scenario. **Max Length**: 2 characters. **Example:** If there are 5 payments to be made and this transaction is the first transaction, set this property to "1". The first transaction should also include the CVV property, but this value should **not** be stored or sent for subsequent transactions.
Member	String	The cardholder's name. **Max Length:** 20 characters.
MerchantNumber° ***	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Credit Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
MTS	*Boolean*	*No Longer Supported.*
Multi	String	Flag that indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PC**Charge GUI. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid values:** 1 = TRUE, 0 = FALSE **Default value:** 0. See the section **Multi-trans Wait** for more information (see page 67). This Flag has no effect if processing will occur over IP or leased line.
Odometer	String	The odometer reading. Only required for Fleet One (7 digits), Voyager (7 digits), and Fuelman (6 digits) cards.
OffLine	String	Flag that indicates whether **PC**Charge should process the transaction offline. If the offline flag is set, **PC**Charge will put the transaction into a .BCH file that resides in the **PC**Charge directory for importing at a later time. The file can only be imported from the **PC**Charge GUI. **Valid values:** 1 = TRUE, 0 = FALSE
OutDelay	Single	The delay time before the **PC**Charge directory is polled for a transaction response file (.oux file). The default is 0.25 seconds. This value should only be modified if the integration is not performing properly. This could be caused if the client machine is slow or there is network lag that causes the server to spend more time checking for .oux files than processing transactions.
Path°	String	**For use with File Transfer CommMethod only.** The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ (**default**) **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
PeriodicPayment	String	Flag that indicates whether the transaction is a recurring transaction. **Valid values:** 1 = TRUE, 0 = FALSE **Note:** If periodic payment is set to true, the recurring billing properties must also be set to achieve the best processing rates.
Port	String	**For use with TCP/IP CommMethod only.** Open port of PCCharge.

Property	Data Type	Description - Charge Class Properties
PrintReceipts	String	The number of receipts that **PC**Charge should print for the transaction. This value will override the corresponding value in the **PC**Charge GUI. **PC**Charge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.
Processor° ***	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
ProductDetailAmount_XX	String	**Note:** Only required for the processor NBS. This is the total dollar amount for PRODUCT_DETAIL_PRODUCT_CODE_XX being authorized. For example, PRODUCT_DETAIL_PRODUCT_CODE_1 has a PRODUCT_DETAIL_QUANTITY_1 = 2 and a PRODUCT_DETAIL_UNIT_PRICE_1 = $2.00, therefore the PRODUCT_DETAIL_AMOUNT_1 = $4.00
ProductDetailCount	String	**Note:** Only required for the processor NBS. All card types are configurable except for Fleet One which is limited to 7 records. Only 1 – 10 records are currently supported through PCCharge for all card types.
ProductDetailCode_XX	String	**Note:** This is the number of items for RODUCT_DETAIL_PRODUCT_CODE_XX. PCCharge will support 1 – 10. • NBS supports using 1 – 10. • BAMS Tandem (NPC) supports using 1 – 3.
ProductDetailQuantity_XX	String	**Note:** Only used for the processor NBS. This is the unit price for PRODUCT_DETAIL_PRODUCT_CODE_XX. This is only used for Fleet One and Fuelman. Currently, PCCharge will support 1 – 10.
Reference	String	The reference number from the original transaction (returned by the processor). Set this property only if processing a Post-Authorization and the Post-Authorization is being used to add a Voice-Authorization to the batch or to "store" a Voice-Authorization. (For information on stored Voice-Authorizations, see page 91). The Reference property does not need to be set if the Post-Authorization completes a standard Pre-Authorization using the TroutD value of the Pre-Authorization. See the section **Follow On Transactions** for more information (see page 70). **Max Length:** 8 characters. **Note:** NBS/ Fleet One cards require a Reference Number to be sent with each transaction. This is a minimum of 2 digits and a max of 15. This must be all numeric.
RestrictionCode	String	Only required for Voyager cards. This is used to determine the level of identification and which fields are required. Two digits. **Valid Values:** 00 - No ID Number or Odometer required. Fuel and Other allowed. 01 - No ID Number or Odometer required. Fuel only allowed. 10 - ID Number only required. Fuel and Other allowed. 11 - ID Number only required. Fuel only allowed. 20 - Odometer only required. Fuel and Other allowed. 21 - Odometer only required. Fuel only allowed. 30 - ID Number and Odometer required. Fuel and Other allowed. 31 - ID Number and Odometer required. Fuel only allowed. **Note:** Required for both manual and swiped transactions.
RFID	String	Set to 1 if card information was read from RFID (Radio Frequency Identification) device. If card was read from from RFID, track data must be populated and manual flag must be set to 1. Set to 0 otherwise.
Store	String	Flag indicating whether a Voice Authorization transaction should be stored. This flag should only be submitted when performing a Post-Authorization transaction (action code 5) that includes an authorization code from the voice operator. For more information on stored Voice-Authorizations, see page 91. **Valid Value:** 1 - Store the Voice Authorization transaction.
Street	String	The cardholder's billing street address. The Street property is used for address verification. Address verification can only be performed on non-swiped transactions. **For FDC:** Use first 5 digits only. **Note:** For manually keyed transactions, Street is required to qualify for the lowest transaction rates. **Max Length:** 20 characters **Citi**©- When used with Citi Plan PO Box's are not allowed.

Property	Data Type	Description - Charge Class Properties
TaxAmt	String	The tax amount. This is the portion of the amount that is tax. Providing the tax amount is required to obtain the best rate on commercial card transactions. **Max Length**: 9 characters (including the decimal). **Format:** DDDDDD.CC. The transaction's action code must indicate that it is a commercial transaction. Tax amount should be included in the amount field.
TaxExempt	Boolean	Tax Exempt Flag. This flag is used to indicate if the purchase is tax exempt. Used only for Commercial Card Transactions. **Valid Values:** 1 – Purchase is tax exempt; 0 – Purchase is not tax exempt.
Ticket	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** For manually keyed transactions, Ticket is required to qualify for the lowest transaction rates. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.
TimeOut	Long	The number of seconds after which a timeout error will be returned from the class. The count will start when the Send method is called. The default timeout value is 90 seconds. It is highly recommended that integrators review the section **Timeouts** (see page 59). Setting the TimeOut value improperly could cause reconciliation issues and problems such as double-charging a customer's account.
TotalAmount	*String*	*No longer needed*
Track	String	The track II data captured from the magnetic strip of the credit card. The track II data is required to ensure the lowest per-transaction rate from the processing company when performing swiped transactions (Retail and Restaurant). Sending the track II data is not allowed if the merchant's industry type is MOTO or eCommerce. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in. Alternatively, the GetParseData method can be used to parse the track data and set the Card, ExpDate, and Track properties automatically.
TroutD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
User° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).
VehicleID	String	Only required for Wright Express cards (5 digits) and Voyager cards (8 digits). **Note:** Required for both manual and swiped transactions.
XMLtrans	*Boolean*	*Set to True to activate the XML message format. It is recommended that the "3" parameter be passed to the Send method to activate the XML message format instead of using the XMLtrans property. See the description for the Send method for more information.*
Zip	String	The cardholder's zip code. The Zip property is used for address verification. **Max Length:** 9 digits. Address verification can only be performed on non-swiped transactions. **Note:** For manually keyed transactions, the Zip is required to qualify for the lowest transaction rates. **Note:** If submitting the 9-digit zip, do not include the dash. **Citi©** - When used with Citi Plan the zip is limited to 5-digits.
CfgType	Long	The database archive configuration type setup in PCCharge. Valid value: Currently, only 0 is supported. 0 = CFG_TXN_ARCHIVE = Configure Transaction Archive. Use action code ZC.
CfgEnabled	Boolean	Enable or disable current database archive configuration (1 = Enable, 0 = Disable).

Property	Data Type	Description - Charge Class Properties
CfgPath	String	Specify path for saved output files (Example: backed up transaction database). Must end with a backslash "\".
CfgSizeLimit	String	Transaction archive size limit for GUI archive prompting and validation. Specified in megabytes.
CfgKeepDays	String	Transaction archive preservation range. All transactions within the past number of "keep days" will remain in the pccw.mdb database following a transaction archive command.
CashierName	String	Submit Cashier's login name.
CashierPassword	String	Submit Cashier's password.
NewCashierPassword	String	Submit a new password for the Target Cashier. **Note:** case-sensitive Must be complex: • minimum 7 characters • must have at least one upper case character • one numeric character one special character (e.g., @, $, %, etc.) **NOTE:** See **Cashier Permissions** on page 75 for an example.
NewCashierConfirmPassword	String	Resubmit the password for verification. **Note**: Must match the original NewCashierPassword **NOTE:** See **Cashier Permissions** on page 75 for an example.
AddCashier1	String	Allows the integrator to submit new cashiers up to 5 at a time. Requires 3 Name Value pairs to be entered for each AddCashier tag submitted. **CASHIER_NAME <Name>** **PASSWORD <Password>** **PERMISSIONS <Permissions>** Example: <ADD_CASHIER_1>CASHIER_NAME CreditR PASSWORD CreditR123% PERMISSIONS 10000|000|000|000|000000000000|0000|000000 </ADD_CASHIER_1> **NOTE:** See **Cashier Permissions** on page 75 for an example.
AddCashier2	String	
AddCashier3	String	
AddCashier4	String	
AddCashier5	String	
TargetCashierName	String	Submitted when performing a function to edit a cashier's settings. This specifies which cashier's information to alter. **NOTE:** See **Cashier Permissions** on page 75 for an example.
CustomerFirstName [1]	String	Applicant's first name.
MiddleInitial [1]	String	Applicant's middle initial.
CustomerLastName [1]	String	Applicant's last name.
Suffix [1]	String	Applicant's suffix. (Ex. M.D., Ph.D, Jr.)
AptSuite [1]	String	Applicant's apartment or suite number.
CustomerCity [1]	String	Applicant's city.
State [1]	String	Applicant data. (Ex: GA, FL, MA…)
Email [1]	String	Applicant data. (Ex: XXX@XX.XXX)
Phone_Number [1]	String	Applicant data. Format xxxxxxxxx 10-digits No Dashes
SSNum [1]	String	Applicant data. Format xxxxxxxxx 9-digits No Dashes
Birth_Date [1]	String	Applicant data. Format is MMDDYYYY.
EmpName [1]	String	Applicant data. Employer name.
WorkPhoneNumber [1]	String	Applicant data. Employer phone number.
HomeOwner [1]	String	Applicant data. Format: '**O**' = Own '**R**' = Rent '**B**' = Board '**P**' = Live w/ parents '**M**' = Military
AnnualIncome [1]	String	Applicant data. Format: Up to 6 digits. Whole dollar amount with no decimals.
PhotoIDState [1]	String	Applicant data. (Ex: GA, FL, MA…)

Property	Data Type	Description - Charge Class Properties
CorrelationUID [1]	String	Received as a response to a Credit App (P3)
PendingNumber [1]	String	Received as a response to a Credit App (P3)
ReplyFlag [1]	Boolean	Reply notification. Format: **Y** = Accept **N** = Decline
SourceCode [1]	String	**Citi Plan** Format: Opt In – **"AP"** Opt Out – **"AO"**
IDType [1]	String	Applicant data. Type of identification being submitted. Format: **'D'** – Driver's License **'O'** - Other
FraudFlag [1]	Boolean	Code 10. Values are **1** or **0: 1** = True, **0** = False
HealthCareAmount [#]	String	Total Healthcare Amount. This amount must be greater than or equal to the sum of the other amount categories. Max: **12 - digits** Format: **DDD.CC**
PrescriptionAmount [#]	String	(Optional) Total amount of the prescription-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
VisionAmount [#]	String	(Optional) Total amount of the vision-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
ClinicAmount [#]	String	(Optional) Total amount of the clinic-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
DentalAmount [#]	String	(Optional) Total amount of the dental-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
TransitAmount [#]	String	(Optional) Reserved for future use. Max: **12 - digits** Format: **DDD.CC**
CopayAmount [#]	String	(Optional) Reserved for future use. Max: **12 - digits** Format: **DDD.CC**
FSA [#]	Boolean	Indicates the transaction is FSA. This will bypass PCCharge BIN checking as PCCharge will assume that the POS has verified that this card has met the BIN requirements for an FSA transaction. **1** = True, **0** = False **Note:** If this is not passed it defaults to False.

° These properties are the minimum required to process a Sale or Pre-Authorization transaction.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

*** If the "Use Default Processor" option is enabled in the **PC**Charge preferences, and the `Processor` and `MerchantNumber` properties are omitted from the transaction request, **PC**Charge will process all transactions using the "Default Processor". The "Default Processor" is defined as the first merchant number that is set up **PC**Charge. Consult the **Multi-Merchant Support** section (see page 68) for more information on the "Use Default Processor" option. In addition, `Processor` and `MerchantNumber` should not be set when doing follow-on transactions. Refer to the section **Follow On Transactions** (see page 70) for more information.

[1] These properties are required to process Citi applications. For more information and examples see Citi Credit Plan on page 116.

[#] Additional tags used to process FSA/HRA transactions. For more information please see the section about **FSA on page 99**.

Charge Class Methods

Method	Returned Value	Description - Charge Class Methods
AddMatch	String	The AddMatch method returns a string representation of the address verification response received from **PC**Charge. The address response code will be used to determine what string should go into AddMatch.
Cancel	None	The Cancel method attempts to cancel the transaction in progress. Calling the Cancel method does not guarantee that the transaction will be canceled; it simply attempts to cancel the transaction.
Clear	None	The Clear method will clear the values in all properties and methods. It is recommended that this method be called: a. after the transaction results have been retrieved by using the various .get methods b. after the DeleteUserFiles method has been called c. prior to running the next transaction
CommercialCardType	String	The CommercialCardType method is used to determine whether or not a credit card is a commercial card. CommercialCardType requires that a string parameter, the credit card number, is passed when calling the method, that the Path property is set to a valid **PC**Charge directory, and that a valid Bin.mdb database resides in that directory. CommercialCardType returns TRUE if the BIN range of the card appears in the Bin.mdb database, FALSE if it does not.
CVV2Match	String	The CVV2Match method returns a string representation of the card verification response that is received from **PC**Charge. The card verification response code will be used to determine what string should go into CVV2Match.
DeleteUserFiles	None	**For use with File Transfer CommMethod only.** The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
GetAcctDataSrc	String	Returns the entry method of the transaction.
GetACI	String	Returns the VPS indicator to indicate wherever the card is a VISA, MC or AMEX card PS2000 data. This value is not returned by all processing companies.
GetAddText1	String	Only supported on Fleet One, this field contains miscellaneous additional text returned from host. Currently PCCharge will support GetAddText1-GetAddText4.
GetAmountDue	String	Returns the amount due, only for Elavon (NOVA) pre-paid and FSA functionality. **Format is DDD.CC** **Note:** "Enable Pre-Paid Cards" option within PCCharge MUST be checked for this feature to work. The option is located under Setup>Credit Card Company>Extended Data and only with the Credit Card Company set to Elavon (NOVA). This option is only configurable in *PCCharge version 5.7.1 release I sp9a* and above. **Note:** For this to work with FSA transactions, one must enable FSA and Partial Auth within the Extended Data Screen for supported processors.
GetAuth	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
GetAuthAmount	String	Used in partial redemption transactions where only part of the amount was authorized. Returns the actual authorized amount.

Method	Returned Value	Description - Charge Class Methods
GetAVS	String	Returns the AVS response code from the issuing bank. If performing Address Verification on card-not-present transactions, this code indicates how well the AVS information passed in matches what the issuing bank has on file for the cardholder. Consult the section **DevKit Constants** for a description of values that may be returned (see page 143)
GetCCAvailBalance	String	**Pre-paid credit cards with Elavon (NOVA) -** Returns the PrePaid card balance. **FSA** – Returns the available balance on the FSA/HRA card. **Format: DDD.CC**
GetCaptured	Boolean	The GetCaptured method returns TRUE if **PC**Charge returns "CAPTURED" as the result of the transaction. Otherwise, FALSE will be returned. The GetCaptured method is used to determine if a transaction that will result in a monetary transfer (Sale, Credit, Post-Authorization, etc.) is approved or declined. A "CAPTURED" response indicates that the transaction has been approved, and that the transaction has been placed in the open batch.
GetCardIDCode	String	Returns a code that is used to verify the identity of the cardholder.
getCommercialCardType	String	The getCommercialCardType method requires a string parameter, the credit card number, and will determine the credit card number's commercial card type. This method requires that the Path variable be set to a valid **PC**Charge directory and it uses the Bin.mdb database in the **PC**Charge directory to determine the commercial card type. **Valid return values:** B – Business P,L,G -- Purchase C – Corporate F – Fleet **Example:** getCommercialCardType("4055011111111111") will return "P".
GetCompanyCity	String	The GetCompanyCity method returns the city of the merchant that is registered in **PC**Charge. The Index and Path properties must be set correctly for this method to work.
GetCompanyName	String	The GetCompanyName method returns the company name of the merchant that is registered in **PC**Charge. The Index and Path properties must be set correctly for this method to work.
GetCompanyState	String	The GetCompanyState method returns the state of the merchant that is registered in **PC**Charge. The Index and Path properties must be set correctly for this method to work.
GetCompanyStreet	String	The GetCompanyStreet method returns the street address of the merchant that is registered in **PC**Charge. The Index and Path properties must be set correctly for this method to work.
GetCompanyZip	String	The GetCompanyZip method returns the zip code of the merchant that is registered in **PC**Charge. The Index and Path properties must be set correctly for this method to work.
GetCreditCardIssuer	String	The GetCreditCardIssuer method returns the abbreviation of the credit card issuer's name for the card number that is present in the Card property. Consult the section **DevKit Constants** for a description of values (see page 143).
GetCreditCardType	String	The GetCreditCardType method returns either the abbreviation of the credit card issuer of the card that is present in the Card property, or the optional card parameter that is passed to the GetCreditCardType method. Consult the section **DevKit Constants** for descriptions of values (see page 143). (GetCreditCardType is the same as GetCardIssuer).
GetDCAvailBalance	String	Returns the available balance on pre-paid debit cards. Only for pre-paid debit cards with Elavon (NOVA).
GetCVV2	String	Returns the CVV2/CVC2/CID response code from the issuing bank. If performing CVV2/CVC2/CID validation on card-not-present transactions, this code indicates if the CVV2/CVC2/CID code passed in matches what the issuing bank has on file for the cardholder. Consult the section **DevKit Constants** for a description of values that may be returned (see page 143)

Method	Returned Value	Description - Charge Class Methods
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetEstGratuityAmount	String	The GetEstGratuityAmount returns the estimated gratuity amount of the original transaction.
GetGratuityAmount	String	The GetGratuityAmount returns the gratuity amount of the original transaction.
GetHostType	Integer	The GetHostType method returns an integer that indicates if a processor / merchant number is Host based or Terminal based. GetHostType requires three parameters: 1) **Processor code** - Consult the section **DevKit Constants** (see page 143) for a list of valid processor codes 2) **Merchant account** - Must be a valid merchant account set up in **PC**Charge 3) **TID type** - Valid Values for TID type: 0 – Credit; 1 – Check; 2 – Debit; 3 – EBT; 4 – GiftCard GetHostType will return one the following values based on the parameters passed in: 0 – The processor is Host based 1 – The processor is Terminal based -1 – The processor is a Hybrid (supports both Host and Terminal processing) or invalid processor / merchant number. **Example:** .GetHostType("VISA", "999999999911", 0) will return 0 **Note:** Chase Paymentech (GSAR), Elavon (NOVA), and FDMS South / NaBanco (NB) are considered hybrid processors. GetHostType will return a -1 for these processors.
GetIND	String	Returns the IND code. The IND code is a transaction description code and an Interchange compliance field. This value is not returned by all processing companies.
GetIndex	Long	The GetIndex returns the index of the Processor and MerchantNumber combination that is stored in **PC**Charge file tid.pcc. The GetIndex function will not work property if valid Path, Processor, and MerchantNumber properties are not provided.
GetItemID	String	The GetItemID echoes the item ID from the original transaction.
GetMSI	String	Returns the Market Specific Indicator. This value indicates the transaction's market segment. This value is assigned by the card associations and is not returned with all transactions.
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetParseData	String	The GetParseData method will parse a string (containing credit card track data) passed to it and populate the Card, ExpDate, and Track properties with the appropriate data. GetParseData will return an integer indicating its success. **Valid return values:** 0 (error parsing data), 1 (track I successful), or 2 (track I & II successful).
GetPLProcessor	String	Retrieves the private label Processor ID currently setup in PCCharge. The .path property must be specified.
GetPLMerchantNumber	String	Retrieves the private label Merchant Number currently setup in PCCharge. The .path property must be specified.
GetPCard	String	The GetPCard method returns the PurchaseCard field from the .oux file. Not all processors support this field. **Valid Values:** 1 = purchasing card, 0 = otherwise
GetPEM	String	Returns the Point of Entry Mode that is associated with the transaction. This value is not returned by all processing companies.

Method	Returned Value	Description - Charge Class Methods
GetPS2000	String	PS2000 Data. This data will be as received during the original authorization processing. It will not be present for offline transactions. PS2000 Data is a variable; it will either be one character or up to 20. It is data concerning the card type and transaction that the processor will send back during the authorization process. This value is not returned by all processing companies.
GetRecordCount	String	The number of records matching the inquiry
GetReceipt	String	Only used for CITI private label cards. Returns the disclosure agreement.
GetRefNumber	String	Returns the reference number associated with the transaction. The reference number is assigned by the card associations. The reference number is used to help identify the transaction and is useful for the cardholder and merchant when doing research. This value is not returned with all transactions.
GetRespCode	String	Returns the response code that is provided by the processor. This value is not returned by all processing companies.
GetResponseCommercialType	String	Returns the type of commercial card that was used for the transaction. This value is not returned by all processing companies.
GetResponsePurchaseCardType	String	Returns a flag indicating whether the processor indicated whether the card was a Purchasing Card or not. This value is not returned by all processing companies. **Valid values:** 1 = Purchasing Card, 0 = Otherwise
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
GetRestrictCode	String	**Note:** Only supported on Fleet One. The product restriction code.
GetRet	String	Returns the Retrieval reference number. This value is not returned by all processing companies
GetTBatch	String	Returns the active batch number for the transaction. This value is not returned by all processing companies.
GetTDate	String	Returns the date that the transaction was processed. This value is not returned by all processing companies.
GetTerminationStatus	String	Retrieves Termination Status which returns a "6" upon successful transaction. If anything other than a "6" is returned there may be an issue with the settlement file and should be investigated. This applies to : **GSAR** and **NBS** **Response:** **6 = Successful Transaction**
GetTI	String	This will indicate the transaction Identifier for VISA or AMEX, it will also return the MC Bank Net reference if the card is a MasterCard. This value is not returned by all processing companies.
GetTicket	String	Returns the ticket number or invoice of the transaction. This value is echoed back from the original transaction or is generated by **PC**Charge if one is required to complete the transaction.
GetTICode	String	Returns the validation code for VISA or the Bank Net Date if the card is a MasterCard**.**This value is not returned by all processing companies.
GetTIM	String	Returns the Time of the transaction. This value is not returned by all processing companies.
GetTraceNumber	String	Returns the trace number from the processor. Only for pre-paid credit cards with Elavon (NOVA).
GetTitem	String	Returns the Transaction Item number or the number that is associated with the transaction in the settlement file. This value is not returned by all processing companies.
GetTransRecord	String	Contains nested XML tags providing information on transaction(s) pulled from Trans table in the PCCharge database (pccw.mdb)
GetTransactionReferenceNumber	String	Returns the transaction reference number from the processor. Only for pre-paid credit cards with Elavon (NOVA).
GetTransNum	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.

Method	Returned Value	Description - Charge Class Methods
GetTroutD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved from the .oux file can be used to view the results of a Transaction Inquiry (ZI) transaction. Refer to the section **Transaction Inquiry** (see page 128) for more information. The text can also be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.
GetXMLRequest	String	This method is used to echo the text that is sent in the request file associated with the transaction. The request (.inx) file contains XML string data. The text that is sent in the .inx file can be used to view the message of any transaction sent to the server. Note: This method must be called after the calling send and before DeleteUserFiles method.
PccSysExists	Boolean	**For use with File Transfer CommMethod only.** The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions.
Send	Boolean	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the class will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has an optional parameter that indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send 3 **Note:** The other values that appear in the enumerated list are for internal use only—do not attempt to use any values other than the ones listed above.
ValidCardLength	Boolean	The ValidCardLength method returns true if the card is of the correct length. Otherwise, false will be returned. ValidCardLength has an optional string parameter in which a card number can be passed. If the parameter is blank, it will use the Card property.
ValidDate	Boolean	The ValidDate method returns TRUE if the expiration date provided in the ExpDate property is valid, or FALSE if it is not.

Method	Returned Value	Description - Charge Class Methods
VerifyAmount	Boolean	The VerifyAmount method returns TRUE if the amount provided in the Amount property is in a valid format (DDDDDD.CC), or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
VerifyCreditCard	Boolean	The VerifyCreditCard method returns TRUE if the credit card number's format is valid and meets the requirements set forth by the credit card companies, FALSE if it does not. If FALSE is returned, use the GetErrorCode and GetErrorDesc methods to determine the reason for failure. VerifyCreditCard has an optional string parameter in which a credit card number can be passed. If the parameter is left blank, VerifyCreditCard will analyze the value set in the Card property.
VerifyExpDate	Boolean	The VerifyExpDate method returns TRUE if the expiration date provided in the ExpDate property is correct and in the right format, or FALSE if it is not. VerifyExpDate calls the ValidDate function to validate the expiration date. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
VerifyMerchantNumber	Boolean	The VerifyMerchantNumber method returns TRUE if the merchant number that is passed to it is set up in **PCC**harge, otherwise, FALSE is returned. Specifically, this method checks for the merchant number in the file TID.PCC, which is located in the **PCC**harge directory. The Path property must be set before calling this Method.
GetCurrentDBSize	String	Current transaction database size in bytes.
GetConfigDBSize	String	Current configured size limit for transaction archive in bytes.
SetSSLCertificate	None	User can set the local certificate to authenticate the SSL TCP/IP request. **Note:** This routine has four parameters. CertificateName : this will store the Store name of the store CertificateLocation : this will assign the store location CertificateIssuesby : This will assign Certificate issued by CertificateSerialNumber : This will assign certificate serial number **Note**: See SSL Information on page 70 for more information.
GetCorrelationUID [1]	String	Returned from Citi, needed for Counter Offer Submission.
GetPendingNumber [1]	String	Returned from Citi, needed for Counter Offer Submission.
GetReplyFlag [1]	String	**'Y'** or **'N'** sent to Citi to notify message is a reply to an offer.
GetOpenToBuy [1]	String	Only present in the response if the "Display OTB" is turned ON. Decimal is included.
GetCreditLimit [1]	String	Only present the in the response. Retrieves credit limit.

[1] These properties are required to process **Citi applications**. For more information and examples see **Citi Credit Plan** on page 116.

Debit Class

The Debit class provides integrators with properties and methods used to submit debit card and EBT transactions to **PC**Charge. To use the Debit class to integrate transaction processing, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to process transactions by using the PccSysExists method.

2. Retrieve pertinent data, such as the PIN, the Key Serial Number (if DUKPT), etc., from the PINpad.

3. Assign the appropriate values to the properties required for the transaction to be performed and validate the values using the various .Verify methods. (The properties marked with a ° in the **Debit Class properties** table are the minimum required to process a Debit Sale transaction.)

4. Call the Send method. (**Note:** When calling the Send method, it is recommended that "3" is passed as a parameter to activate the XML message format)

5. Wait for the transaction to process and then call the various .Get methods to determine the outcome of the transaction (code using the .Get methods may be placed immediately after the Send method). The most important information can be acquired by calling the GetResult and GetAuth methods. If an error occurs, call the GetErrorCode and GetErrorDesc methods to determine the nature of the error.

6. Call the DeleteUserFiles method to delete all files related to the transaction.

7. Call the ClearVariables method to reset all the properties and methods related to the transaction or destroy the object.

When processing debit cards, a PINpad is required to allow the customer to enter their PIN. In addition, debit card information is always collected via a card swipe device, never via keyboard entry. Because of this, a card reader is also required. (Some EBT transactions can be manually entered).

When processing U.S. debit card transactions, merchants have the option of allowing the customer to receive cash back on a transaction. For instance, the customer purchases $50 of products and wants $25 cash back, set the Amount to 50.00 and CashBack to 25.00. This will withdraw a total of $75 from the debit card account, $50 for the products and $25 for cash to give to the customer.

Consult the section **Pseudo-code** (see page 158) for various examples that may be followed when using the Debit class to perform transaction processing.

For information on integrating Canadian Debit, see the section **Canadian (Interac) Debit Transactions** (see page 113).

Note: **PC**Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

Debit Class Properties

Property Name	Data Type	Description – Debit Class Properties
Action° °°	Long	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
Amount° °°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00". **Note:** If performing an EBT Balance Inquiry transaction and providing an amount, set this property to "0.00".
AuthCode	String	**EBT Only**: For an EBT Post (Prior Auth Sale) or Force transaction: The Authorization code from the original voice authorization.
Billpay	String	Only valid for Visa debit and credit transactions. It is used to indicate the transaction is being ran for payment of a bill (ultilty, monthly gym dues, etc.) **Valid values:** 0 – Non-Bill payment transaction 1 – Bill payment transaction
Card° °°	String	The Debit/EBT card number that will be used when processing the transaction. **Max Length:** 20 characters. **Example:** 5424180279791765
CashBack	String	The amount of cash back that the customer will receive. This amount is in addition to value entered in Amount property. For example, if the total amount of the sale is $10 and the customer has requested $5 cash back, Amount should be set to $10 and CashBack should be set to $5. The CashBack property should be formatted the same the Amount property. **Max Length:** 9 characters. **Note:** Some debit processors do not support the cash back feature.
Command	String	Because the Action property is defined as "long", this property was added to allow action codes that contain strings. This property is only used for action code M1 (Key Change Request).
CommMethod	Enum	Specifies which communication method will be used. 0 – File_Transfer 1 – TCP/IP Please refer to page 20 for a description of these methods. If TCP/IP is selected, the IPAddress, Port and EnableSSL properties must also be set. If File_Transfer is set then the Path property must be set.
DebitType°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Set this to the bank account type that the customer specified when entering transaction data into the PINpad. **Valid Values:** "Chequing" or "Savings"
EnableSSL	Boolean	**For use with TCP/IP CommMethod only**. SSL is available starting with PCCharge version 5.8. Set to True to enable SSL Socket integration. **Note**: See SSL Information on page 70 for more information.
ExpDate° °°	String	The expiration date associated with the Debit/EBT card number that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208 Set this property if there is an expiration date associated with the Debit/EBT card.
FoodStamp	Boolean	**EBT Only:** Indicates what type of EBT transaction will be performed. **Valid Values:** 1 – Food stamp transaction; 0 – Cash benefits transaction
Gratuity°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. This is the Gratuity Amount of the transaction.

Property Name	Data Type	Description - Debit Class Properties
IPAddress	String	**For use with TCP/IP CommMethod only.** IPAddress of machine where PCCharge is running. Defaults to 127.0.0.1
KeySerialNumber°°	String	If a Key Serial Number is returned from the PINpad, this property should be populated with that number. **If processing transactions with a PINpad using DUKPT encryption**, this value is sixteen or twenty characters long (depending on the processor's encryption). The **PC**Charge DevKit provides several tools for retrieving data from PINpads. If the **PC**Charge integration method chosen doesn't support these tools or the tools do not support the PINpad being used, a direct interface to the PINpad must be written by the integrator. **If processing transactions with a Verifone SC5000 PINpad**, set this property to the Chip Serial Number of the PINpad.
LanguageCode°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Set this to the language that is indicated by the Language Code that is encoded in the track II data on the customer's card. **Valid Values:** "English" or "French" (pass in the literal string)
MACData°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Set this to the MAC Block value returned by the PINpad.
Manual° °°	Long	Flag that indicates whether the transaction was swiped or manually entered. This property must be set to 1 (swiped) for Debit transactions or swiped EBT transactions. If the transaction was swiped, the Track property must also be set. If performing a manually keyed EBT transaction, such as a Force or Voucher, set this property to 0 (manually entered).
Member	String	The cardholder's name. **Max Length:** 20 characters.
MerchantNumber° °° ***	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Debit Card Setup** window or **EBT Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
OrigPurchData	String	The Original Purchase Data. Used when performing a Debit Return with the processors TSYS, Heartland, RBS WorldPay, and NPC. This is the original transaction date. **Format:** DDMMhhmm
OutDelay	Single	The delay time before the **PC**Charge directory is polled for a transaction response file (.oux file). The default is 0.25 seconds. This value should only be modified if the integration is not performing properly. This could be caused if the client machine is slow or there is network lag that causes the server to spend more time checking for .oux files than processing transactions.
Path° °°	String	**For use with File Transfer CommMethod only.** The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ (**default**) **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Pin° °°	String	The encrypted PIN block that is retrieved from the PINpad. The PIN is provided to the processor for verification. **Length:** 16 characters. The **PC**Charge DevKit provides several tools for retrieving data from PINpads. If the **PC**Charge integration method chosen doesn't support these tools or the tools do not support the PINpad being used, a direct interface to the PINpad must be written by the integrator.
Port	String	**For use with TCP/IP CommMethod only.** Open port of PCCharge.
ProductDetailAmount_XX	String	**Note:** Only required for the processor NBS. This is the total dollar amount for PRODUCT_DETAIL_PRODUCT_CODE_XX being authorized. For example, PRODUCT_DETAIL_PRODUCT_CODE_1 has a PRODUCT_DETAIL_QUANTITY_1 = 2 and a PRODUCT_DETAIL_UNIT_PRICE_1 = $2.00, therefore the PRODUCT_DETAIL_AMOUNT_1 = $4.00

Property Name	Data Type	Description - Debit Class Properties
ProductDetailCount	String	**Note:** Only required for the processor NBS. All card types are configurable except for Fleet One which is limited to 7 records. Only 1 - 10 records are currently supported through PCCharge for all card types.
ProductDetailCode_XX	String	**Note:** This is the number of items for RODUCT_DETAIL_PRODUCT_CODE_XX. PCCharge will support 1 - 10. • NBS supports using 1 - 10. • BAMS Tandem (NPC) supports using 1 - 3.
ProductDetailQuantity_XX	String	**Note:** Only used for the processor NBS. This is the unit price for PRODUCT_DETAIL_PRODUCT_CODE_XX. This is only used for Fleet One and Fuelman. Currently, PCCharge will support 1 - 10.
Processor° °° ***	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
Reference	String	NBS/ Fleet One cards require a Reference Number to be sent with each transaction. This is a minimum of 2 digits and a max of 15. This must be all numeric.
RFID	String	Set to 1 if card information was read from RFID (Radio Frequency Identification) device. If card was read from from RFID, track data must be populated and manual flag must be set to 1. Set to 0 otherwise.
ShiftID		Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. The Shift ID. This value is optional. **Format:** Alphanumeric **Max Length:** 1 character.
Ticket	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.
TimeOut	Long	The number of seconds after which a timeout error will be returned from the class. The count will start when the Send method is called. The default timeout value is 90 seconds. It is highly recommended that integrators review the section **Timeouts** (see page 59). Setting the TimeOut value improperly could cause reconciliation issues and problems such as double-charging a customer's account.
Track° °°	String	The track II data captured from the magnetic strip of the card. The track II data is required for Debit transactions and for swiped EBT transactions. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in.
TransNum	*String*	*No longer needed*
TroutD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
User° °° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).
Voucher	String	**EBT Only:** The voucher number for an EBT force transaction. The voucher is provided by the processor at the time of authorization and must be supplied to clear the voucher.
XMLtran	*Boolean*	*Set to True to activate the XML message format. It is recommended that the "3" parameter be passed to the Send method to activate the XML message format instead of using the XMLtran property. See the description for the Send method for more information.*
CashierName	String	Submit Cashier's login name.

Property Name	Data Type	Description - Debit Class Properties
CashierPassword	String	Submit Cashier's password.

° These properties are required to process a Debit Sale transaction.

°° These properties are required to process a Canadian Debit Sale transaction using Global Payments East (NDC) and the SC5000 PINpad.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

*** If the "Use Default Processor" option is enabled in the **PC**Charge preferences, and the Processor and MerchantNumber properties are omitted from the transaction request, **PC**Charge will process all transactions using the "Default Processor". The "Default Processor" is defined as the first merchant number that is set up **PC**Charge. Consult the **Multi-Merchant Support** section (see page 68) for more information on the "Use Default Processor" option. In addition, Processor and MerchantNumber should not be set when doing follow-on transactions. Refer to the section **Follow On Transactions** (see page 70) for more information.

Debit Class Methods

Method Name	Returned Value	Description - Debit Class Methods
ClearVariables	None	The ClearVariables method will clear the values in all properties and methods. It is recommended that this method be called: a. after the transaction results have been retrieved by using the various .get methods b. after the DeleteUserFiles method has been called c. prior to running the next transaction
DeleteUserFiles	None	**For use with File Transfer CommMethod only.** The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
GetApproved	Boolean	The GetApproved method returns TRUE if **PC**Charge returns "APPROVED" as the result of the transaction. Otherwise, FALSE will be returned.
GetAuth	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
GetAuxRespCode	String	When using the SC5000 PINpad, returns the ISO response code
GetAvlBalance	String	**EBT Only:** The GetAvlBalance method returns the available balance on the EBT card. This value is not returned by all processing companies.
GetCaptured	Boolean	The GetCaptured method returns TRUE if **PC**Charge returns "CAPTURED" as the result of the transaction. Otherwise, FALSE will be returned. The GetCaptured method is used to determine if a transaction that will result in a monetary transfer (Sale, Credit, Post-Authorization, etc.) is approved or declined. A "CAPTURED" response indicates that the transaction has been approved, and that the transaction has been placed in the open batch.
GetEBTCashBalance	String	**EBT Only:** Returns the remaining balance on a Cash Benefits card. This value is not returned by all processing companies.
GetEBTFoodBalance	String	**EBT Only:** Returns the remaining balance on a Food Stamp card. This value is not returned by all processing companies.

Method Name	Returned Value	Description - Debit Class Methods
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetMerchantInfo	String	The GetMerchantInfo method returns a string containing all of the debit merchant numbers and processors set up in **PC**Charge. The string will begin with STX and will end with ETX. GS will separate each record, and FS will separate fields within a record. **Example:** <STX>CES <FS>0000000927996296767<GS>GSAR<FS> 999999999999519<GS>VISA<FS>999999999911<ETX> Refer to the section **Multi-Merchant Support** (see page 68) for more information on the GetMerchantInfo method.
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetMSI	String	**For Debit Master Session:** Returns the new master key (if one exists) sent by the processor that should be passed to the PINpad.
GetPOSSequenceNumber	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Returns the current POS Sequence Number for the PINpad. The Path property must be set to the **PC**Charge directory and the PINpad's Chip Serial Number must be passed as a parameter when calling the GetPOSSequenceNumber method.
GetRefNumber	String	Returns the reference number associated with the transaction. The reference number is used to help identify the transaction and is useful for the cardholder and merchant when doing research. This value is not returned with all transactions.
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
GetSurchargeAmount	String	**For GSAR Debit US only:** Returns the surcharge amount that was charged by the bank when using debit with GSAR. This value has to be called in order for the developer to know how much the card was actually charged in addition to the transaction amount and cash back.
GetTermFee	String	The GetTermFee method returns the terminal fee that applies to the transaction. The terminal fee is designated by the processor. Not all processors return the terminal fee.
GetTI	String	**For Debit Master Session:** Returns the new working key (if one exists) sent by the processor that should be passed to the PINpad for the next transaction. **For GSAR EBT:** Returns the ledger balance.
GetTraceNum	String	The GetTraceNum method returns the trace number for the transaction that was returned by the processor. Not all processors support this field.
GetTransNum	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
GetTroutD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved can be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.

Method Name	Returned Value	Description - Debit Class Methods
GetXMLRequest	String	This method is used to echo the text that is sent in the request file associated with the transaction. The request (.inx) file contains XML string data. The text that is sent in the .inx file can be used to view the message of any transaction sent to the server. Note: This method must be called after the calling send and before DeleteUserFiles method.
PccSysExists	Boolean	**For use with File Transfer CommMethod only.** The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions.
Send	None	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the class will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has an optional parameter that indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send 3 **Note:** The other values that appear in the enumerated list are for internal use only-- do not attempt to use any values other than the ones listed above.
VerifyAmount	Boolean	The VerifyAmount method returns true if the amount provided in the Amount property is in a valid format (DDDDDD.CC). Otherwise, VerifyAmount will return false. If false is returned, check the error code to determine the reason for failure Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
SetSSLCertificate	None	User can set the local certificate to authenticate the SSL TCP/IP request. **Note:** This routine has four parameters. CertificateName : this will store the Store name of the store CertificateLocation : this will assign the store location CertificateIssuesby : This will assign Certificate issued by CertificateSerialNumber : This will assign certificate serial number **Note**: See SSL Information on page 70 for more information.

Check Class

The Check class provides integrators with properties and methods used to submit check verification, guarantee, and conversion transactions to **PC**Charge. To use the Check class to integrate transaction processing, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to process transactions by using the PccSysExists method.

2. Assign the appropriate values to the properties required for the transaction to be performed. (The properties marked with a ° in the **Check Class properties** table are the minimum required to process a check verification transaction.)

3. Call the Send method. (**Note:** When calling the Send method, it is recommended that "3" is passed as a parameter to activate the XML message format)

4. Wait for the transaction to process and then call the various .Get methods to determine the outcome of the transaction (code using the .Get methods may be placed immediately after the Send method). The most important information can be acquired by calling the GetResult and GetAuth methods. If an error occurs, call the GetErrorCode and GetErrorDesc methods to determine the nature of the error.

5. Call the DeleteUserFiles method to delete all files related to the transaction.

6. Call the Clear method to reset all the properties and methods related to the transaction or destroy the object.

Note: **PC**Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

Check Class Properties

Property Name	Data Type	Description - Check Class Properties
Account_Number°°	String	For Check, MICR, or Double ID: The account number that will be used when processing the transaction. **Max Length:** 20 characters.
Action°	String	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
Amount°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
AdjustmentAmount	String	Total amount of the transaction after adjustment (i.e. if the original transaction was $5.00 and it should have been $50.00, the adjustment transaction request should have the .Amount property set equal to 50.00).
Birth_Date°°	String	The date of birth of the check writer. **Length:** Exactly six characters. **Format:** MMDDYY. The birth date is required for DL (Driver's License) check transactions.
Cash_Back	String	The amount of cash back that the customer will receive. This amount is in addition to value entered in Amount property. For example, if the total amount of the sale is $10 and the customer has requested $5 cash back, Amount should be set to $10 and CashBack should be set to $5. The CashBack property should be formatted the same the Amount property. **Max Length:** 9 characters. **Note:** Some processors do not support the cash back feature.
CashierNum	String	The Cashier Number
CheckType	String	**Valid Values:** 0 = Personal check, 1 = Business check **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
Check_Number°	String	The check number of the check that will be used when processing the transaction. **Max Length:** 10 characters.
CommMethod	Enum	Specifies which communication method will be used. 0 – File_Transfer 1 – TCP/IP Please refer to page 20 for a description of these methods. If TCP/IP is selected, the IPAddress, Port and EnableSSL properties must also be set. If File_Transfer is set then the Path property must be set.
CustomerCity	String	The customer's city. **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
CustomerName	String	The first and last name of the customer. **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.

Property Name	Data Type	Description - Check Class Properties
CheckReaderCode	Enum	Passes the type of Check Reader that is being used. Currently only used by Telecheck and will only be set if TECK is the set processor. Cannot be configured in the PCCharge GUI. **Valid Values:** 1 - Magtek_mini_micr 2 - EnCheck_3000 3 - IVI_2500 4 - IVI_430 5 - IVI_431 6 - ICE_5700 7 - MagtekImager 8 - VeriFone_CR1000i 9 - Epson_TMH6000 10 - Epson_TMH6000Imager 11 - WelchAllyn_ScanTeam 8300 12 - VeriFone_CR600 13 - Magtek_Imager_with_Modem 14 - IBM_4610_reader_printer 15 - Ingenico_EC2600 16 - RDM_EC5000 17 - RDM_EC6000 18 - NCR_7158_and_7167 19 - LS_100 20 - Magtek_Excella 21 - Magtek_Excella_DLCapture_FBChkImg 22 - Verifone_Model_Quartet
CustomerStreet	String	The street address of the customer. **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
Drivers_License°°	String	The driver's license number of the individual writing the check. **Max Length:** 20 characters. The driver's license is required for DL (Driver's License) transactions and when performing Double ID transactions.
DLTrackII	String	The parsed TrackII data from the driver's license. **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
EnableSSL	Boolean	**For use with TCP/IP CommMethod only**. SSL is available starting with PCCharge version 5.8. Set to `True` to enable SSL Socket integration. **Note**: See SSL Information on page 70 for more information.
IPAddress	String	**For use with TCP/IP CommMethod only.** IPAddress of machine where PCCharge is running. Defaults to 127.0.0.1
ManagerNum	String	Used for BPS Double ID transactions. Optional Manager Number for manager override.
Manual°	String	Flag that indicates whether the transaction was manually entered or swiped. **Valid values:** `0` = manual transaction, `1` = swiped transaction
MerchantNumber°	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Check Services Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
MICR_DATA	String	The raw MICR data from the bottom of the check. Used for conversion transactions.
MICRStatus	String	**Valid Values:** 15 = Valid read by MICR reader, 15I = Valid read by MICR reader with imaging capability, 9 = Manual only **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
Multi	String	Flag that indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PC**Charge GUI. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid values:** `1 = TRUE, 0 = FALSE` **Default value:** `0`. See the section **Multi-trans Wait** for more information (see page 67). This Flag has no effect if processing will occur over IP or leased line.
OutDelay	Single	The delay time before the **PC**Charge directory is polled for a transaction response file (.oux file). The default is 0.25 seconds. This value should only be modified if the integration is not performing properly. This could be caused if the client machine is slow or there is network lag that causes the server to spend more time checking for `.oux` files than processing transactions.

Property Name	Data Type	Description - Check Class Properties
Path°	String	**For use with File Transfer CommMethod only.** The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ (**default**) **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Phone_Number°°	String	The phone number of the individual writing the check. **Max Length:** 7 digits. **Format:** digits only. The phone number is required for COD (Checks On Delivery).
Port	String	**For use with TCP/IP CommMethod only.** Open port of PCCharge.
Processor°	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
Services	ServicesType	The type of check verification to be performed. This property may be specified by using a numerical value or an enumerated value if the programming language being used supports enumerated values. **Valid values:** 0 (MICR) – MICR 1 (COD) – Checks-On-Delivery 2 (DL) – Driver's License 3 (DI) – Double ID 4 (SPS) – Use if Check processor is SPS **Note:** The value set in the Services property overrides the value set in the Action property.
PrintReceipts	String	The number of receipts that **PC**Charge should print for the transaction. This value will override the corresponding value in the **PC**Charge GUI. **PC**Charge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.
State°°	String	The state code of the state that issued the check writer's driver's license. The state code is required for DL (Driver's License). **Format:** 2 characters.
Ticket	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.
TimeOut	Long	The number of seconds after which a timeout error will be returned from the class. The count will start when the Send method is called. The default timeout value is 90 seconds. It is highly recommended that integrators review the section **Timeouts** (see page 59). Setting the TimeOut value improperly could cause reconciliation issues and problems such as double-charging a customer's account.
Transit_Number°°	String	The Transit Routing Number / ABA number that will be used when processing the transaction. This value indicates which bank issued the check. **Max Length:** 9 characters. This value is required for MICR transactions and when performing Double ID transactions.
TroutD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
User° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).

Property Name	Data Type	Description - Check Class Properties
XMLtrans	*Boolean*	*Set to True to activate the XML message format. It is recommended that the "3" parameter be passed to the Send method to activate the XML message format instead of using the XMLtrans property. See the description for the Send method for more information.*
Zip_Code°°	String	The check writer's ZIP code. **Max Length:** 9 characters. **Format:** digits only. This value is required for COD transactions. **Note:** If submitting the 9-digit zip, do not include the dash.
CashierName	String	Submit Cashier's login name.
CashierPassword	String	Submit Cashier's password.

Note: To perform Double ID, both the MICR_DATA and Drivers_License fields must be populated.

° These properties are required, regardless of service type.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

°° COD -- required for Checks-On-Delivery
DL -- required for Driver's License
MICR -- required for MICR

Check Class Methods

Method Name	Returned Value	Description - Check Class Methods
Clear	None	The Clear method will clear the values in all properties and methods. It is recommended that this method be called: d. after the transaction results have been retrieved by using the various .get methods e. after the DeleteUserFiles method has been called f. prior to running the next transaction
DeleteUserFiles	None	**For use with File Transfer CommMethod only.** The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
GetApproved	Boolean	The GetApproved method returns TRUE if **PC**Charge returns "APPROVED" as the result of the transaction. Otherwise, FALSE will be returned. An "APPROVED" response indicates that a Verification has been approved.
GetAuth	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
GetCaptured	Boolean	The GetCaptured method returns TRUE if **PC**Charge returns "CAPTURED" as the result of the transaction. Otherwise, FALSE will be returned. The GetCaptured method is used to determine if a conversion transaction that will result in a monetary transfer is approved or declined. A "CAPTURED" response indicates that the transaction has been approved, and that the transaction has been placed in the open batch.
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).

Method Name	Returned Value	Description - Check Class Methods
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetRespCode	String	Returns the response code that is provided by the processor. This value is not returned by all processing companies.
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
GetResultCode	String	Returns the result code that is provided by the processor. This value is not returned by all processing companies.
GetReturnCheckFee	String	Returns the response from the processor which indicates the fee for returned checks. **Note**: Only used for the processor TECK
GetReturnCheckNote	String	Returns the response from the processor which displays a note for returned checks. **Note**: Only used for the processor TECK
GetReference	String	Returns the reference number that is provided by the processor. This value is not returned by all processing companies.
GetTransNum	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
GetTroutD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
GetTraceID	String	**Only for TECK.** Returns the Trace ID associated with the transaction.
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved can be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.
GetXMLRequest	String	This method is used to echo the text that is sent in the request file associated with the transaction. The request (.inx) file contains XML string data. The text that is sent in the .inx file can be used to view the message of any transaction sent to the server. Note: This method must be called after the calling send and before DeleteUserFiles method.
PccSysExists	Boolean	**For use with File Transfer CommMethod only.** The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions.

Method Name	Returned Value	Description - Check Class Methods
Send	None	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the class will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has an optional parameter that indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send 3 **Note:** The other values that appear in the enumerated list are for internal use only—do not attempt to use any values other than the ones listed above.
VerifyAmount	Boolean	The VerifyAmount method returns TRUE if the amount provided in the Amount property is in a valid format (DDDDDD.CC), or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
SetSSLCertificate	None	User can set the local certificate to authenticate the SSL TCP/IP request. **Note:** This routine has four parameters. CertificateName : this will store the Store name of the store CertificateLocation : this will assign the store location CertificateIssuesby : This will assign Certificate issued by CertificateSerialNumber : This will assign certificate serial number **Note**: See SSL Information on page 70 for more information.

EBT

Because Debit and EBT transaction are similar, the Debit Class should be used to perform EBT transactions. Consult the section **Debit Class** for more information (See page 280)

Gift Class

The `Gift` class provides integrators with properties and methods used to submit gift card transactions to **PC**Charge. To use the `Gift` class to integrate transaction processing, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to process transactions by using the `PccSysExists` method.

2. Assign the appropriate values to the properties required for the transaction to be performed and validate the values using the various `.Verify` methods. (The properties marked with a ° in the **Gift Class properties** table are the minimum required to process a Gift Card Redemption / Sale transaction.)

3. Call the `Send` method. (**Note:** When calling the `Send` method, it is recommended that "3" is passed as a parameter to activate the XML message format)

4. Wait for the transaction to process and then call the various `.Get` methods to determine the outcome of the transaction (code using the `.Get` methods may be placed immediately after the `Send` method). The most important information can be acquired by calling the `GetResult` and `GetAuth` methods. If an error occurs, call the `GetErrorCode` and `GetErrorDesc` methods to determine the nature of the error.

5. Call the `DeleteUserFiles` method to delete all files related to the transaction.

6. Call the `Clear` method to reset all the properties and methods related to the transaction or destroy the object.

Note: **PC**Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

VeriFone Stored Value API (GAPI)

The VeriFone Stored Value API (GAPI) is a proprietary specification that allows for stored value card processors to add themselves to **PC**Charge. Applications using GAPI can also integrate with **PC**Charge using the various integration methods. For more information on adding a stored value card processor to **PC**Charge, and how to obtain the VeriFone Stored Value API, please contact VeriFone sales at 1-800-725-9264.

Gift Class Properties

Property Name	Data Type	Description - Gift Class Properties
Action°	String	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
Amount°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00". For Valuelink (VLNK) Balance Adjustment: **Format:** +/-DDDDD.CC.
Authcode	String	The auth code property that will be used for processing Voids for VTEC, VLNK, MELL, and GSAR. For VTEC and VLNK, set this property to the auth code of the original transaction to be voided. For GSAR and MELL, set this property to the reference number of the original transaction to be voided. For BPS, set to retrieval reference number of original transaction (the one to be voided).
Card°	String	The gift card number that will be used when processing the transaction. **Max Length:** 20 characters.
CardSeqNum	String	The card sequence number for the transaction. Currently, GSAR is the only processor that uses this property. If sending multiple transactions to GSAR, this should be the sequence number of the transaction. For example, if ten cards are being issued and this is the fifth in the sequence, set CardSeqNum to 5. CardSeqNum should be no more than two characters long.
CashierID	String	The numeric cashier ID for the gift card transaction. The only gift card processors that currently support the CashierID are VTEC and VLNK.
CheckCard	Boolean	Flag that indicates whether to activate gift card validity testing. **Valid Values:** TRUE; FALSE. **Default value:** TRUE. This value must be set to FALSE when performing Follow on transactions because the card number is omitted from these transaction requests.
CommMethod	Enum	Specifies which communication method will be used. 0 - File_Transfer 1 - TCP/IP Please refer to page 20 for a description of these methods. If TCP/IP is selected, the IPAddress, Port and EnableSSL properties must also be set. If File_Transfer is set then the Path property must be set.
Demo	Boolean	The demo mode flag. In demo mode, a simulated response is returned in which even amounts return approved, and odd amounts return declined. **Valid Values:** TRUE - Activates demo mode FALSE - Deactivates demo mode (default)
EnableSSL	Boolean	**For use with TCP/IP CommMethod only**. SSL is available starting with PCCharge version 5.8. Set to True to enable SSL Socket integration. **Note**: See SSL Information on page 70 for more information.
ExpDate	String	The expiration date associated with the gift card that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208 **Note:** Most gift cards do not have an expiration date.
Force	Boolean	The Force flag. The Force flag indicates whether or not an approval code has already been issued. The Force flag is used only by GSAR Redemption or a single GSAR Issuance/Add Value transaction.

Property Name	Data Type	Description - Gift Class Properties
GiftPin	String	Only used for the processor **SVS**. To retrieve pin, call GetGfitPin upon activation. Used for only for virtual gift card transactions.
GratuityAmount	String	The gratuity amount for the transaction. Tip should be no more than 9 characters long (including the decimal). **Format:** DDDDDD.CC.
Industry°°	String	The industry type for the transaction. **Valid Values:** 1 = retail, 2 = restaurant. **For VLNK:** 0 = retail, 1 = restaurant, 2 = e-Commerce.
IPAddress	String	**For use with TCP/IP CommMethod only.** IPAddress of machine where PCCharge is running. Defaults to 127.0.0.1
LastValidDate	String	The last year that will be considered a valid expiration date. Currently, the default value in the class is "09". It is recommended that a setting is provided by which the end-user can change this property; otherwise, in the future, end users will require a new class to be distributed to resolve expiration date issues. **Length:** 2 digits. **Format:** YY **Example:** If LastValidDate is set to 05, then cards between 06 and 99 are considered to be 1906 to 1999, and cards between 00 and 05 are 2000 to 2005.
Loyalty	Boolean	The Loyalty flag indicates whether or not the transaction is a loyalty transaction. Currently, only VTEC supports the Loyalty flag. The default value of the Loyalty flag is false.
Manual°	Long	Flag that indicates whether the transaction was manually entered or swiped. If the transaction was swiped, the Track property must also be set. **Valid values:** 0 = manual transaction, 1 = swiped transaction
MerchantNumber°	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Gift Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
Multi	String	Flag that indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PC**Charge GUI. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid values:** 1 = TRUE, 0 = FALSE **Default value:** 0. See the section **Multi-trans Wait** for more information (see page 58). This Flag has no effect if processing will occur over IP or leased line.
OldCard	String	The old card property. OldCard should be no more than twenty characters long. **For VTEC:** OldCard will be used for the Replace transaction. **For VLNK:** OldCard will be used for the Balance Merge and Balance Transfer transactions.
OutDelay	Single	The delay time before the **PC**Charge directory is polled for a transaction response file (.oux file). The default is 0.25 seconds. This value should only be modified if the integration is not performing properly. This could be caused if the client machine is slow or there is network lag that causes the server to spend more time checking for .oux files than processing transactions.
Partial	Boolean	For GSAR: Flag indicating whether the transaction is a partial redemption transaction.
Path°	String	**For use with File Transfer CommMethod only.** The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ (**default**) **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Points	String	The number of points that will be redeemed in a Loyalty Points transaction. Points should be no more than nine characters long.
Port	String	**For use with TCP/IP CommMethod only.** Open port of PCCharge.
PrintReceipts	String	The number of receipts that **PC**Charge should print for the transaction. This value will override the corresponding value in the **PC**Charge GUI. **PC**Charge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.

Property Name	Data Type	Description - Gift Class Properties
Processor°	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
PromoCode	String	Used for GVEX: A code defined by the merchant that affects the calculation from amount and units to points.
Refund	String	Flag that indicates whether to provide the customer a refund when performing a VTEC Deactivate transaction. **Valid Values:** 1 – Provide refund 0 – Do not provide refund
RFID	String	Set to 1 if card information was read from RFID (Radio Frequency Identification) device. If card was read from from RFID, track data must be populated and manual flag must be set to 1. Set to 0 otherwise.
Ticket	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all gift processors support ticket numbers.
TimeOut	Long	The number of seconds after which a timeout error will be returned from the class. The count will start when the Send method is called. The default timeout value is 90 seconds. It is highly recommended that integrators review the section **Timeouts** (see page 59). Setting the TimeOut value improperly could cause reconciliation issues and problems such as double-charging a customer's account.
TIP	String	The tip amount for the transaction. TIP should be no more than 9 characters long (including the decimal). **Format:** DDDDDD.CC. Currently, tips are supported via the TIP property only for VTEC and VLNK restaurant transactions.
TotalCardNum	String	The total number of cards that will be sent to **PC**Charge. GSAR is currently the only processor that uses this property. If sending multiple transactions to **PC**Charge, this should be the total number of the transactions that will be sent. **Example:** If ten cards are being issued, set TotalCardNum to 10. TotalCardNum should be no more than two characters long.
Track	String	The track II data captured from the magnetic strip of the card.. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in. Alternatively, the GetParseData method can be used to parse the track data and set the Card, ExpDate, and Track properties automatically.
TroutD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
User° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).
VirtualGiftCardFlag	Boolean	Only used for the processor **SVS**. 0 - False, 1 - True – Only sent on an activation to determine if a pin should be returned.
XMLtran	*Boolean*	*Set to True to activate the XML message format. It is recommended that the "3" parameter be passed to the Send method to activate the XML message format instead of using the XMLtran property. See the description for the Send method for more information.*
TableNumber	String	Only used for GAPI in restaurant mode. This is the table number of the gift card holder
TrackI	String	Only used for GAPI. The Track I information associated with the card
CashierName	String	Submit Cashier's login name.

Property Name	Data Type	Description - Gift Class Properties
CashierPassword	String	Submit Cashier's password.
SkuLoyalty	String	**GIVEX ONLY** – **Integration Only** Pass this tag in order to send product codes, quantities and amounts for loyalty transactions. **Format:** **Single Product Code:** <SKU_LOYALTY>Product Code,Amount,Quantity</SKU_LOYALTY> **Multi-Product Codes (Separated with a semi-colon):** <SKU_LOYALTY>Product Code,Amount,Quantity; Product Code,Amount,Quantity</SKU_LOYALTY>

° These properties are required to process a gift card redemption or sale transaction.

°° Required for VTEC gift card transactions

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

GiftCard Class Methods

Method Name	Returned Value	Description - GiftCard Class Methods
Abort	Boolean	The Abort method attempts to cancel the transaction in progress and will return a Boolean value that indicates whether or not the transaction was canceled. **Note:** This method is not available when integrating using FoxPro. Use the Cancel method instead.
Cancel	None	The Cancel method attempts to cancel the transaction in progress. Calling the Cancel method does not guarantee that the transaction will be canceled; it simply attempts to cancel the transaction.
Clear	None	The Clear method will clear the values in all properties and methods. It is recommended that this method be called: g. after the transaction results have been retrieved by using the various .get methods h. after the DeleteUserFiles method has been called i. prior to running the next transaction
DeleteUserFiles	None	**For use with File Transfer CommMethod only.** The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
GetActivationCount	String	Returns the number of activations in the current batch
GetActivationTotalAmount	String	Returns the total dollar amount of activations in the current batch
GetAddPointsCount	String	Returns the number of AddPoints Transactions in the current batch
GetAddPointsTotalAmount	String	Returns the total dollar amount of AddPoints transactions in the current batch
GetAddValueCount	String	Returns the number of AddValue transactions in the current batch
GetAddValueTotalAmount	String	Returns the total dollar amount of AddValue transactions in the current batch
GetAmountDue	String	Used in partial redemption transactions where only part of the amount was authorized. Returns the remainder amount that is owed to the merchant.

Method Name	Returned Value	Description - GiftCard Class Methods
GetAuth	String	The GetAuth method returns the authorization number for approved transactions or the reason the transaction was declined (if the processor provides one). **For GVEX Balance transaction:** GetAuth will return the balance remaining on an account. **For all other GVEX transactions:** GetAuth will return the transaction's reference/error message. For VTEC, returns the Auth Code. **For a VTEC Batch function:** use this method to retrieve the number of sales done that day and the total amounts of sales in the following format <# of transaction>, <amount>.
GetAuthAmount	String	Used in partial redemption transactions where only part of the amount was authorized. Returns the actual authorized amount.
GetBalanceTransferCount	String	Returns the number of Balance Transfers in the current batch
GetBalanceTransferTotalAmount	String	Returns the total dollar amount of Balance Transfers in the current batch
GetCaptured	Boolean	The GetCaptured method returns TRUE if **PC**Charge returns "CAPTURED" as the result of the transaction. Otherwise, FALSE will be returned. The GetCaptured method is used to determine if a transaction that will result in a monetary transfer (Sale, Credit, Post-Authorization, etc.) is approved or declined. A "CAPTURED" response indicates that the transaction has been approved.
GetCashBack	String	Used in redemption for remaining balance transactions where the transaction amount is so close to the balance of the card that the entire balance is authorized. Returns the remainder that is owed to the customer.
GetCreditCount	String	Returns the number of credits in the current batch
GetCreditTotalAmount	String	Returns the total dollar amount of credits in the current batch
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetExp	String	Returns the expiration date for processors who issue expiration dates in the response.
GetGiftCardBalance	String	Returns the gift card balance.
GetGiftCardIssuer	String	The GetGiftCardIssuer method returns the gift card issuer of the card that is present in the Card property. Currently, there are no standards for indicating what type of gift card is present. Therefore, whatever value is in the Processor property is what will be returned.
GetGiftCardType	String	The GetGiftCardType method returns the gift card issuer of the card that is present in the Card property or the optional card parameter that is passed to the GetGiftCardType method (the GetGiftCardType is the same as GetGiftCardIssuer). Currently, there are no standards for indicating what type of gift card is present. Therefore, whatever value is in the Processor property is what will be returned.
GetGiftPin	String	Only used for the processor **SVS**. Returned on activation if the virtual gift card tag is set to "1".
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetParseData	String	The GetParseData method will parse a string (containing credit card track data) passed to it and populate the Card, ExpDate, and Track properties with the appropriate data. GetParseData will return an integer indicating its success. **Valid return values:** 0 (error parsing data), 1 (track I successful), or 2 (track I & II successful).
GetPointsCount	String	Returns the number of points transactions in the current batch
GetPointsTotalAmount	String	Returns the total dollar amount of points transactions in the current batch
GetProcRespCode	String	The processor response code. Only returned by the processor **SVS**.

Method Name	Returned Value	Description - GiftCard Class Methods
GetRefNumber	String	The GetRefNumber returns the Reference field from the .oux file. The Reference field is used for different purposes (depending on the gift card processor). **For GVEX Register transaction:** The first eleven digits of an account number will be returned. **For all VTEC transactions:** The account's remaining balance will be returned. **For a VTEC batch function:** use this method to retrieve the number of activations done that day and the total amounts of activations in the following format <# of transaction>, <amount>.>. For a BPS **Redemption** transaction, returns the retrieval reference number.
GetRespCode	String	Returns the response code that is provided by the processor. This value is not returned by all processing companies.
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
GetRet	String	**For GVEX:** Returns the loyalty balance. **For VLNK:** Returns the trace number. **For a VTEC batch function: :** use this method to retrieve the number of Gift Transactions Voids performed that day. You can call GetVoidBalance to determine the total amount of the voids.
GetSaleCount	String	Returns the number of redemptions in the current batch
GetSaleTotalAmount	String	Returns the total dollar amount of redemptions in the current batch
GetTI	String	The GetTicket method returns the Ticket field from the .oux file. The Ticket field will return the ticket for all transactions except for a VTEC batch function. **For a VTEC batch function:** use this method to retrieve the number of gift card that has been de-activated that day and the total amounts of de-activations in the following format <# of transaction>, <amount>.>.
GetTicket	String	The GetTicket method returns the Ticket field from the .oux file. The Ticket field will return the ticket for all transactions except for a VTEC batch function. **For a VTEC batch function:** use this method to retrieve the number of gift card that has been de-activated that day and the total amounts of de-activations in the following format <# of transaction>, <amount>.>.
GetTIM	String	Returns the Time of the transaction. This value is not returned by all processing companies. For VTEC, returns the Amount Due.
GetTipCount	String	Returns the number of Tip transactions in the current batch
GetTipTotalAmount	String	Returns the total dollar amount of Tip transactions in the current batch
GetTransDateTime	String	Returns the transaction date and time when passed back by a processor.
GetTransNum	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
GetTroutD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
GetUpdateData	*String*	*Used internally*
GetVoidBalance	String	Returns the Void Balance
GetVoidCount	String	Returns the number of voids in the current batch
GetVoidTotalAmount	String	Returns the total dollar amount of Voids in the current batch
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved can be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.

Method Name	Returned Value	Description - GiftCard Class Methods
GetXMLRequest	String	This method is used to echo the text that is sent in the request file associated with the transaction. The request (.inx) file contains XML string data. The text that is sent in the .inx file can be used to view the message of any transaction sent to the server. Note: This method must be called after the calling send and before DeleteUserFiles method.
PccSysExists	Boolean	**For use with File Transfer CommMethod only.** The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions.
Send	None	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the class will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has an optional parameter that indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send 3 **Note:** The other values that appear in the enumerated list are for internal use only—do not attempt to use any values other than the ones listed above.
ValidCardLength	Boolean	The ValidCardLength method returns true if the card is of the correct length. Otherwise, ValidCardLength will return false. ValidCreditCard has an optional string parameter in which a card number can be passed. If the parameter is blank, it will use the Card property. If false is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
ValidDate	Boolean	The ValidDate method returns TRUE if the expiration date provided in the ExpDate property is valid, or FALSE if it is not.
ValidIssuer	Boolean	Returns TRUE for valid card issuer
VerifyAmount	Boolean	The VerifyAmount method returns TRUE if the amount provided in the Amount property is in a valid format (DDDDDD.CC), or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
VerifyAmount2	Boolean	The VerifyAmount2 method returns TRUE if the amount provided in the Amount property is in a valid format (+/-DDDDD.CC). or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150). The difference between VerifyAmount and VerifyAmount2 is that VerifyAmount2 allows a + or – to be in the first position of the Amount property. This is needed for Balance Adjustment transactions.

Method Name	Returned Value	Description - GiftCard Class Methods
VerifyExpDate	Boolean	The VerifyExpDate method returns TRUE if the expiration date provided in the ExpDate property is correct and in the right format, or FALSE if it is not. VerifyExpDate calls the ValidDate function to validate the expiration date. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
VerifyGiftCard	Boolean	The VerifyGiftCard method verifies that a card is provided and that the card is the expected length. VerifyGiftCard will return true if these conditions are met. Otherwise, VerifyGiftCard will return false.
VerifyMerchantNumber	*Boolean*	*No Longer Supported*
VerifyProcessor	*Boolean*	*No Longer Supported*
GetPreAuthCount	String	Only for GAPI, this returns the total number of gift card pre-auth transactions processed that day.
GetPreAuthAmount	String	Only for GAPI , this returns the total amount of gift card pre-auth transaction processed that day.
GetPostAuthCount	String	Only for GAPI, this returns the total number of the gift card post-auth transactions processed that day.
GetPostAuthAmount	String	Only for GAPI, this returns the total amount of the post-auth transactions processed that day.
GetIssuanceCount	String	Only for GAPI, this returns the total number of gift cards issued that day.
GetIssuanceTotalAmount	String	Only for GAPI, returns the total amount of the gift cards issued that day.
GetDeactivateCount	String	Only for GAPI, this returns how many gift cards where deactivated that day.
GetDeactivateTotalAmount	String	Only for GAPI, this returns the total amount of gift card deactivations that day.
GetBalanceAdjustCount	String	Only for GAPI, this returns the number of gift cards that were balance adjusted that day.
GetBalanceAdjustTotalAmount	String	Only for GAPI, this returns the total amount of balance adjustment on gift cards that day.
GetBalanceMergeCount	String	Only for GAPI, this returns the total number of the gift cards that were balance merged that day.
GetBalanceMergeTotalAmount	String	Only for GAPI, this returns the total amount of gift card balance merges that day.
GetReportLostStolenCount	String	Only for GAPI, returns the total reported stolen or lost gift cards that day.
GetReportLostStolenTotalAmount	String	Only for GAPI, returns the total amount of all stolen or reported lost gift cards that day.
GetCashoutTotalAmount	String	Only for GAPI, returns the total amount of all cashout transactions processed that day.
GetCashoutCount	String	Only for GAPI, returns the total number of the cashout transactions processed that day.
GetReactivateCount	String	Only for GAPI, returns the total number of gift cards that have been reactivated that day.
GetReactivateTotalAmount	String	Only for GAPI, the total amount of all gift cards that have been reactivated that day.
SetSSLCertificate	None	User can set the local certificate to authenticate the SSL TCP/IP request. **Note:** This routine has four parameters. CertificateName : this will store the Store name of the store CertificateLocation : this will assign the store location CertificateIssuesby : This will assign Certificate issued by CertificateSerialNumber : This will assign certificate serial number **Note**: See SSL Information on page 70 for more information.

Batch Class

Batch Class Properties

Property Name	Data Type	Description - Batch Class Properties
Action	Single	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
BatchCloseType	String	Flag that determines what type of batch close will occur. This flag only supported by FDMS Atlanta and Fifth-Third when using action code 30 or 31 **Valid values:** 1 – Standard End of Day Batch Close (Default) 2 – Shift Close 3 – Fifth-Third Terminal Based Batch Close of Debit, EBT, or Gift
Cancel	Boolean	Set the Cancel property to TRUE to attempt to cancel the settle/close function. Check the GetResult method to see if the function was Canceled.
CommMethod	Enum	Specifies which communication method will be used. 0 – File_Transfer 1 – TCP/IP Please refer to page 20 for a description of these methods. If TCP/IP is selected, the IPAddress, Port and EnableSSL properties must also be set. If File_Transfer is set then the Path property must be set.
Demo	String	The demo mode flag. In demo mode, a simulated response is returned. **Valid Values:** 1 – Activates demo mode 0 – Deactivates demo mode (default)
EnableSSL	String	**For use with TCP/IP CommMethod only**. SSL is available starting with PCCharge version 5.8. Set to True to enable SSL Socket integration. **Note**: See SSL Information on page 70 for more information.
IPAddress	String	**For use with TCP/IP CommMethod only.** IPAddress of machine where PCCharge is running. Defaults to 127.0.0.1
MerchantNumber	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the various setup windows of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
OutDelay	Single	The delay time before the **PC**Charge directory is polled for a transaction response file (.oux file). The default is 0.25 seconds. This value should only be modified if the integration is not performing properly. This could be caused if the client machine is slow or there is network lag that causes the server to spend more time checking for .oux files than processing transactions.
Path	String	**For use with File Transfer CommMethod only.** The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ (**default**) **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Port	String	**For use with TCP/IP CommMethod only.** Open port of PCCharge.
Processor	String	The code for the processing company that will be used when performing batch operations. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
SplitProcessor	String	Only used when settling the processor CITI for private label transactions. Set this property to the main credit card processor ID code being used.

Property Name	Data Type	Description - Batch Class Properties
TimeOut	Long	The number of seconds after which a timeout error will be returned from the class. The count will start when the Send method is called. The default timeout value is 90 seconds. It is highly recommended that integrators review the section **Timeouts** (see page 59).
User	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).
XMLtrans	*Boolean*	*Set to True to activate the XML message format. It is recommended that the "3" parameter be passed to the Send method to activate the XML message format instead of using the XMLtrans property. See the description for the Send method for more information.*
Command	String	
CashierName	String	Submit Cashier's login name.
CashierPassword	String	Submit Cashier's password.
BatchNumber	String	Submit batch number that was returned on the AMEX split dial settlement, needs to be passed when performing the following Action Codes: 35 – Reverse Batch 36 – Resubmit Batch 37 – Get Results 38 – Finalize Batch
CardType	String	Submit with a Batch Inquiry to receive summary results based on Card Type. Valid Values: MC – MasterCard VISA – Visa AMEX – American Express DISC – Discover Card DEBIT – Debit Card EBT – Electronic Benefits Card PL – Private Label JCB – JCB Card

The following properties are no longer available in the Batch class and should be ignored:

AmexAmount	PurchaseCount
AmexCount	Response
Balance	ReturnAmount
BatchDate	ReturnCount
BatchNumber	Store
CIC	Terminal
ItemCount	VisaMCAmount
MTS	VisaMCCount
PurchaseAmount	

Batch Class Methods

Method Name	Returned Value	Description - Batch Class Methods
Clear	None	The Clear method will clear the values in all properties and methods. It is recommended that this method be called: j. after the transaction results have been retrieved by using the various .get methods k. after the DeleteUserFiles method has been called l. prior to running the next transaction

Method Name	Returned Value	Description - Batch Class Methods
DeleteUserFiles	None	**For use with File Transfer CommMethod only.** The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
GetAccepted	Boolean	The GetAccepted method returns a value of true if the batch was accepted (for Terminal based processors) or processed (for Host based processors). If the batch was not accepted or processed, GetAccepted will return a value of false. Index is an optional parameter. If multiple batches have been settled, there will be several entries in the .oux file. GetNumber(Index) will return the number of merchant accounts that will be in the .oux file.
GetBalance	String	The GetBalance method returns the dollar amount of the last settled batch or the amount waiting to be settled in the open batch.
GetBatches	String	The GetBatches method can only be used with Batch Inquiry transactions. This method returns the number of batches that will be settled for a particular merchant number. **Example:** If a merchant account is set up as TSYS using TCP/IP, there is a limitation on how many transactions can be sent across on a single batch. Therefore, **PC**Charge breaks the batch up into smaller batches. GetBatches returns the number of smaller batches that would be created for that merchant account.
GetBatchNumber	String	After a terminal-based batch settles, returns the batch number.
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetFileExt	String	Returns the file extension for the files the batch class will be accessing.
GetItemCount	String	The GetItemCount method returns the number of transactions that are in the batch for which the function was performed.
GetMerchantInfo	String	The GetMerchantInfo method returns a string containing all of the merchant numbers and processors set up in **PC**Charge. The string will also indicate whether the processor is Host based (H), Terminal based (T), or a hybrid (Y). The string will begin with STX and will end with ETX. GS will separate each record, and FS will separate fields within a record. **Example:** <STX>CES <FS>0000009279962967<FS>T<GS>GSAR<FS> 999999999999519<FS>T<GS>VISA<FS>999999999911<FS>T<ETX> Refer to the section **Multi-Merchant Support** (see page 68) for more information on the GetMerchantInfo method.
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetNumberIndexs	Integer	The GetNumberIndexs method returns the number of merchant numbers that are stored in **PC**Charge. This number indicates how many batches will be settled or closed if an action code of 39 is submitted.
GetProcessor	String	Returns the processor ID specified in the Processor property. Only returned with CITI settlement.
GetProcessed	Boolean	The GetProcessed method returns true if the result of the action performed was "Processed". "Processed" is a response that **PC**Charge returns for an inquiry transaction and a Close on a Host based processor.

Method Name	Returned Value	Description - Batch Class Methods
GetRespCode	String	Returns the response code for the batch if the close batch command was given. The response code indicates whether or not the transaction was successfully closed. If the batch is declined, the GetResult method will provide more information indicating why the transaction was not approved. **Valid Values:** 2 = Settled, 6 = Declined, or 8 = Deferred.
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions. The GetAccepted or GetProcessed methods will indicate whether or not the batch operation was successful. However, if both GetAccepted and GetProcessed are FALSE, the GetResult method will provide more information about why the batch operation was not successful.
GetSettleAmount	*None*	*No Longer Supported*
GetSettleNumber	String	The GetSettleNumber method returns the settlement number that is stored in association with the transaction in **PC**Charge's database. This number is a sequencing number **PC**Charge generates internally; the processing company does not generate it.
GetStatus	String	Returns the status of the batch when performing an inquiry or a batch close/settle operation. If performing a batch close/settlement operation, GetStatus will return a response from the processor that indicates whether or not the batch was successfully closed or settled. **Example:** If TSYS rejects the batch, GetStatus will return the RB (rejected batch) number from TSYS . If TSYS accepts the batch, GetStatus will return the batch number and an "ACCEPTED" response will be returned.
GetSystemInfo	None	The GetSystemInfo method is used to set the MerchantNumber and Processor properties of the Batch class. To use GetSystemInfo, pass the index number of a merchant number that is registered in **PC**Charge as a parameter (for example, the first Merchant number that is set up in **PC**Charge is assigned the index of "1"). Once the index number has been passed to **PC**Charge via GetSystemInfo, the merchant number and processor can be retrieved using the MerchantNumber and Processor properties.
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved can be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.
GetXMLRequest	String	This method is used to echo the text that is sent in the request file associated with the transaction. The request (.inx) file contains XML string data. The text that is sent in the .inx file can be used to view the message of any transaction sent to the server. Note: This method must be called after the calling send and before DeleteUserFiles method.
PccSysExists	Boolean	**For use with File Transfer CommMethod only.** The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions.

Method Name	Returned Value	Description - Batch Class Methods
Send	None	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the class will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has an optional parameter that indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send 3 **Note:** The other values that appear in the enumerated list are for internal use only-- do not attempt to use any values other than the ones listed above.
SetSSLCertificate	None	User can set the local certificate to authenticate the SSL TCP/IP request. **Note:** This routine has four parameters. CertificateName : this will store the Store name of the store CertificateLocation : this will assign the store location CertificateIssuesby : This will assign Certificate issued by CertificateSerialNumber : This will assign certificate serial number **Note**: See SSL Information on page 70 for more information.
GetRecordCount	String	The number of records matching the inquiry
GetTerminationStatus	String	Retrieves Termination Status which returns a "6" upon successful settlement. If anything other than a "6" is returned there may be an issue with the settlement file and should be investigated. This applies to : **GSAR**, **Elavon (NOVA)**, and **NBS** **Response:** **6 = Batch Settled and file has been deleted.** **8 = Batch Settled, but the file is locked and cannot be deleted.**
GetEBTCount	String	Number of EBT Transactions
GetEBTAmount	String	Total Dollar Amount of EBT Transactions
GetAMEXCount	String	Number of AMEX Transactions
GetAMEXAmount	String	Total Dollar Amount of AMEX Transactions
GetTotalCount	String	Number of Transactions
GetTotalAmount	String	Total Dollar Amount of Transactions in the batch
GetVISACount	String	Number of VISA Transactions
GetVISAAmount	String	Total Dollar Amount of VISA Transactions
GetMCCount	String	Number of MC Transactions
GetMCAmount	String	Total Dollar Amount of MC Transactions
GetDISCCount	String	Number of DISC Transactions
GetDISCAmount	String	Total Dollar Amount of DISC Transactions
GetPLCount	String	Number of PL Transactions
GetPLAmount	String	Total Dollar Amount of PL Transactions
GetJCBCount	String	Number of JCB Transactions
GetJCBAmount	String	Total Dollar Amount of JCB Transactions

Note: In the event there are multiple batches waiting to be settled in one settlement, the integrated application will need to be designed to loop through the settlement response to retrieve the response for each batch.

Offline Class

Offline Class Properties

Property Name	Data Type	Description - Offline Class Properties
GetInProcessRecord	Long	Returns the index of the record in the .bch file that is currently being processed. This property will be updated during the ProcessFile function to reflect the stage of the process.
GetVoid	Boolean	Returns true if the transaction requested is marked as void. GetVoid will take an integer and a string as arguments. The integer is the Index of the record to be checked, and the string is the name of the .bch file. If an error is encountered, GetVoid will set the error code and description and exit the function.
MerchantNumber	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Credit Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric. MerchantNumber will be used to process the transactions when the .bch file is processed.
Path	String	The path of the directory in which the .bch file resides. **Example:** C:\Program Files\PCCW\MyBatchFiles.
PccwPath	String	The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ (**default**) **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Processor	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152). Processor will also be used to determine the index that will be used for processing the .bch file.
User	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).
Void	Boolean	Sets the flag that indicates whether or not that transaction should be marked as voided. **Usage:** Load the transaction. Set the Void property to true. Use the UpdateRecord method to save the data.

Offline Class Methods

Method Name	Returned Value	Description - Offline Class Properties
Compact	Boolean	The Compact method takes all transactions marked as voided out of the .bch file. If no .bch file is passed in when Compact is called, it will use a file name that consists of the processor and index. **Example:** Visa1.bch. If an error is encountered during processing, Compact will set the error code and description and will exit the function.

Method Name	Returned Value	Description - Offline Class Properties
Connect	Boolean	The Connect method sets an internal class to the object that is passed in to the Connect method. During the processing of the ProcessFile method, the object will call the OnUpdate function of that object and pass to that method an integer that identifies the record that is being processed. Connect will return true if it successfully sets the class to the object passed. Otherwise, Connect will be return false.
Disconnect	Boolean	The Disconnect method sets the internal class to Nothing. The internal class is the same class that is modified when Connect is called.
EraseFile	Boolean	The EraseFile method sends the .bch file to the Recycle Bin. If no .bch file is passed in when EraseFile is called, it will use a file name that consists of the processor and index. **Example:** Visa1.bch. If EraseFile encounters an error during processing, the error code and description will be set and will exit the function.
GetAmount	String	The GetAmount method returns the amount of the transaction in the current record. The GetRecord method must be called first.
GetAppCode	String	The GetAppCode method returns the approval code of the transaction in the current record. The GetRecord method must be called first.
GetBalance	Currency	The GetBalance method returns the total balance of the current batch file. This balance will be set after calling GetTotals or ProcessFile.
GetBchFile	String	The GetBchFile method returns a file name if one is not provided. The file name will consist of the processor code and index. **Example:** Visa1.bch.
GetCard	String	The GetCard method returns the card of the transaction in the current record. The GetRecord method must be called first.
GetCount	Long	The GetCount method returns the number of transactions that are stored in the .bch file. This variable will be updated after the Compact, GetTotals, and ProcessFile.
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during use of various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetExpDate	String	The GetExpDate method returns the expiration date of the transaction in the current record. The GetRecord method must be called first.
GetIndex	Long	The GetIndex method returns the index of the process and merchant number combinations in the file tid.pcc. The MerchantNumber and Processor properties will be used to determine the index.
GetItem	String	The GetItem method returns the item or record number of the transaction in the current file. The GetRecord method must be called first.
GetRecord	Boolean	The GetRecord method updates the current transaction data with the data at the index that is provided when calling GetRecord.
GetRecords	Long	The GetRecords method returns the number of records that are in a particular .bch file. This data will be updated after calling the GetVoid, GetTotals, and ProcessFile method.
GetTicket	String	The GetTicket method returns the ticket number of the transaction in the current record. The GetRecord method must be called first.
GetTotals	Boolean	The GetTotals method updates the balance, counts the transactions in the .bch file, and puts them in local variables. These figures can be retrieved with the GetCount and GetBalance.
GetTransType	String	The GetTransType method returns a string representation of the type of transaction in the current record. **Example:** Sale, Void, Credit, etc. The GetRecord method must be called first.

Method Name	Returned Value	Description - Offline Class Properties
ProcessFile	Boolean	The ProcessFile method accesses the .bch file provided (or calls GetBCHFile if not provided), and processes the transactions in the file. If an error occurs while processing the file, ProcessFile will update the error code and description and processing will be terminated.
UpdateRecord	Boolean	The UpdateRecord method accesses the .bch file provided (or calls GetBCHFile if not provided), and marks the records provided as voided if the Void flag is set to true. If an error occurs while processing the file, UpdateRecord will update the error code and description and processing will be terminated.

Reporting

The Charge class may be used by integrators to submit report requests. A report request can have **PC**Charge print a report to it's default report printer or have **PC**Charge generate a file containing the report output. If generating a file, the **PC**Charge reporting interface supports three different file types:

1. Portable Document Format (.pdf)
2. Rich Text Files (.rtf)
3. Standard Text files (.txt)

Note: The reporting interface cannot be configured to send reports directly to the screen.

The following outlines the properties used for submitting report requests to PCCharge with the Charge class. The properties in the Charge class that are not documented below should be left blank when submitting report requests.

Property	Data Type	Description - Charge Class Reporting Properties
Action°	Long	The action code that identifies what type of report will be requested. **Valid Values:** 81-84. **Example:** If running a credit card detail report, set the action code to "81". Consult the section **DevKit Constants** for a list of valid values (see page 143).
Card	String	**User name filter**. If a valid user name is set in the Card property, the report will be filtered by that user name. The report returned will consist of only those transactions processed by the user name specified. **Example:** "User1". If this property is left blank, the report will show transactions processed by all users.
CheckCard°	Boolean	Flag that indicates whether to activate credit card validity testing. **Valid Values:** TRUE; FALSE. **Default value:** TRUE. **This value must be set to FALSE when submitting a report request.**
CommMethod	Enum	Specifies which communication method will be used. 0 – File_Transfer 1 – TCP/IP Please refer to page 20 for a description of these methods. If TCP/IP is selected, the IPAddress, Port and EnableSSL properties must also be set. If File_Transfer is set then the Path property must be set.
EnableSSL	Boolean	**For use with TCP/IP CommMethod only**. SSL is available starting with PCCharge version 5.8. Set to True to enable SSL Socket integration. **Note**: See SSL Information on page 70 for more information.
IPAddress	String	**For use with TCP/IP CommMethod only.** IPAddress of machine where PCCharge is running. Defaults to 127.0.0.1
Manual	Long	**Result filter**. Use this filter to create a report consisting of only those transactions with the result specified. **Valid Values:** 0 = all (default), 1 = approved, 2 = declined **Example:** 1

Property	Data Type	Description - Charge Class Reporting Properties
Member	String	**Ending Date/Time filter**. Specifies the end date and end time of the report. **Format**: Date: MM/DD/YY Time: HH:MM:SS PM. When used in conjunction with Street; will create a report consisting of only those transactions processed between the start and end date/time specified (inclusive). If an end date is not specified, today's date is assumed. If an end time is not specified, 11:59:59 PM is assumed. The end date can be passed without the end time. However, the end time cannot be passed without the end date. **Examples:** "07/06/05 06:00:00 PM" or "07/06/05"
MerchantNumber	String	**Merchant Number filter**. Set this property to filter the report by the merchant number specified. Setting this property will generate a report consisting of only those transactions processed via the merchant number specified. To generate a report that includes all merchant numbers in **PC**Charge, set this property to "ALL" or leave blank. **Example:** "99999999911"
Path°	String	**For use with File Transfer CommMethod only.** The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ (**default**) **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
PeriodicPayment	String	**Report Output setting**. Determines if the report will be printed by **PC**Charge or written to a file. **Valid Values:** "0" = print to default printer specified in **PC**Charge (default). "1" = print to file using filename specified in TransID and path specified in TRACK.
Port	String	**For use with TCP/IP CommMethod only.** Open port of PCCharge.
Street	String	**Starting Date/Time Filter** (Optional) Specifies the start date and start time of the report. **Format**: Date: MM/DD/YY Time: HH:MM:SS PM. Use to create a report consisting of only those transactions processed on or after the date specified. If a start date is not specified, today's date is assumed. If a start time is not specified, 12:00:00 AM is assumed. The start date can be passed without the start time. However, the start time cannot be passed without the start date. **Examples:** "03/04/05 09:00:00 AM" or "03/04/05"
TimeOut	Long	The number of seconds after which a timeout error will be returned from the class. The count will start when the Send method is called. The default timeout value is 90 seconds. It is highly recommended that integrators review the section **Timeouts** (see page 59).
Track	String	**Destination Directory for Report File**. Specifies the destination directory where the report file will be generated by **PC**Charge (if PeriodicPayment is set to "1"). **Example:** "C:\My Documents\PCCReports\" **Path Formats:** UNC, MS-DOS(8 Characters) and Long. **Max Length:** 40 characters (if the Destination Directory is longer than 40 characters, use CustCode for the additional characters. Must end with a "\" unless the directory name will be continued in the CustCode property. **Note:** If running in a Client/Server environment, this property is the path from the server running **PC**Charge, not the client. For example, if a client submitted a report request that specified "C:\" as the destination directory, the report would be written to the local hard drive of the server running **PC**Charge, not to the client's hard drive.
CustCode	String	**Destination Directory for Report File (continued)**. Continuation of the destination directory (if the directory name is greater than 40 characters). **Max Length:** 25 characters. Must end with a "\"

Property	Data Type	Description - Charge Class Reporting Properties
TRANSID	String	**Report File name/Report File Type**. Specifies the filename and extension of the report file generated by **PC**Charge (if PeriodicPayment is set to "1"). Also determines what file type will be used when **PC**Charge generates the report. To specify the file type, set the extension to one of the following: **.pdf** – Create the report file in the Portable Document Format. **Ex.** Report.pdf **.rtf** – Create the report file in Rich Text. **Ex.** Report.rtf **.txt** – Create a report file in flat text. **Ex.** Report.txt **Default: .txt** (If an extension other than the ones listed is passed, the report will be returned as flat text and a .txt extension will be added to the filename)
User°	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).

° These properties are required to submit a report request.

The following outlines the methods used to process report requests. The methods in the Charge class that are not documented below will not be used when processing report requests.

Method	Returned Value	Description - Charge Class Reporting Methods
Cancel	None	The Cancel method attempts to cancel the transaction in progress. Calling the Cancel method does not guarantee that the transaction will be canceled; it simply attempts to cancel the transaction.
Clear	None	The Clear method will clear the values in all properties and methods. It is recommended that this method be called: a. after the transaction results have been retrieved by using the various .get methods b. after the DeleteUserFiles method has been called c. prior to running the next transaction
DeleteUserFiles	None	**For use with File Transfer CommMethod only.** The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
GetAuth	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.

Method	Returned Value	Description - Charge Class Reporting Methods
PccSysExists	Boolean	**For use with File Transfer CommMethod only.** The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions.
Send	Integer	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the class will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has an optional parameter that indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send 3 **Note:** The other values that appear in the enumerated list are for internal use only-- do not attempt to use any values other than the ones listed above.
VerifyMerchantNumber	Boolean	The VerifyMerchantNumber method returns TRUE if the merchant number that is passed to it is set up in **PC**Charge, otherwise, FALSE is returned. Specifically, this method checks for the merchant number in the file TID.PCC, which is located in the **PC**Charge directory. The Path property must be set before calling this Method.

OLE/COM Method

Several exposed OLE classes in **PC**Charge allow integrators to perform various processing functions. Before these classes can be used, a reference will need to be made to the **PC**Charge executable. For example, in Microsoft Visual Basic 6, follow the procedure below.

1. Choose **Project** | **References** from the Visual Basic menu bar.

2. After the References window opens, from the list, scroll to and check the box next to either:

   - **PCCharge Pro** (to reference **PC**Charge Pro)
   - **Active-Charge Payment Server** (to reference **PC**Charge Payment Server)

   and click **OK**. (The DevKit installation attempts to install both **PC**Charge Pro and **PC**Charge Payment Server by default. If the programs have not yet been installed on the system, install them from the DevKit CD and refer to the section **Getting Started** (see page 24) to set up both products.)

PCCharge Pro and **PC**Charge Payment Server provide the developer with several OLE classes to allow the integration of payment processing:

- **PccCharge** (for Credit card integration)
- **PCCDebit** (for Debit card integration)
- **PccCheck** (for Check integration)
- **PCCEBT** (for EBT card integration)
- **PCCGiftCard** (for Gift/Loyalty integration)
- **PccBatch** & **PccSettle** (for Batch/Settlement integration)
- **PccPinPad** (for PINpad integration)
- **PccBin** (for utilities used to determine Commercial Card information)

The properties and methods of the various classes can be viewed through the object browser. If MS VB6 is not being used, refer to the language documentation for instructions on adding OLE references.

Note: The additional classes that are exposed that do not appear in the list above do not provide transaction processing functionality. These classes provide various utilities or provide setup functionality. See the sections **Utility Related Classes** (see page 378) and **Setup Related Classes** (see page 382) for more information on these additional classes.

Note: The OLE method of integration was primarily designed to be used for integrations in which the **PC**Charge engine and the integrated application both reside on the same machine.

WARNING: If integrating via OLE, the integrated application must use the same version of **PC**Charge as the version of the **PC**Charge DevKit used to create the integration.

Using the OLE classes to integrate synchronously

To program in a synchronous manner, create an instance of any of the OLE classes by using the following line of code:

Set <instance name> = New <object name>

For example, to create an instance based on the PccCharge class, the following line of code would be used in MS VB6:

Set Charge = New ActiveCharge.PccCharge

Using the OLE classes to integrate asynchronously

To program in an asynchronous manner using the OLE classes, the object definition for the OLE class must indicate to use events. Also, a parameter must be passed with the Send method that indicates asynchronous communication.

To define an object that indicates to use events based on the PccCharge class, the following line of code would be used in MS VB6:

Dim WithEvents Charge As ActiveCharge.PccCharge

To create an instance based on the PccCharge class, the following line of code would be used in MS VB6:

Set Charge = New ActiveCharge.PccCharge

To pass the parameter that indicates asynchronous communication to the Send method, use the following format:

.Send [**TRUE** (for asynchronous) or **FALSE** (for synchronous)], **3** (XML message format)]

For example, the following line of code would indicate asynchronous communication using the XML message format:

.Send True, 3

PccCharge Class

The PccCharge class provides integrators with properties and methods used to submit credit card transactions to **PC**Charge. To use the PccCharge class to integrate transaction processing, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to process transactions by using the PccSysExists method.

2. Assign the appropriate values to the properties required for the transaction to be performed and validate the values using the various .Verify methods. (The properties marked with a ° in the **PccCharge Properties** table are the minimum required to process a Sale or Pre-Authorization transaction.)

3. Call the Send method. (**Note:** When calling the Send method, it is recommended that "3" is passed as a parameter to activate the XML message format)

4. If programming asynchronously, wait for the Error or Finish event to occur.

5. If programming synchronously, code using the .Get methods may be placed immediately after the Send method

6. Call the various .Get methods to determine the outcome of the transaction. The most important information can be acquired by calling the GetResult and GetAuth methods. If an error occurs, call the GetErrorCode and GetErrorDesc methods to determine the nature of the error.

7. Call the DeleteUserFiles method to delete all files related to the transaction.

8. Call the Clear method to reset all the properties and methods related to the transaction or destroy the object.

Consult the **Pseudo-code** section (see page 158) for various examples that may be followed when using the Charge.OCX to perform transaction processing.

Note: **PC**Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

PccCharge Properties

Property Name	Data Type	Description - PccCharge Properties
Action°	Long	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
Amount°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
AmxChargeDescription	String	The American Express Charge Description. This is a general description describing merchandise: the AMEX representative and the merchant will decide on an appropriate description. **Note:** Only Required for Retail, MOTO and Restaurant transactions when using AMEX direct settlement or TSYS **Max Length:** 23 bytes
AmxDescription_1	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AmxDescription_2	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AmxDescription_3	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AmxDescription_4	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AuthCode	String	The Authorization code. This value is returned by the issuing bank and should only be set in a transaction request if processing a Post-Authorization and the Post-Authorization is being used to add a Voice-Authorization to the batch or to "store" a Voice-Authorization. (For information on stored Voice-Authorizations, see page 91). The AuthCode property does not need to be set if the Post-Authorization completes a standard Pre-Authorization using the TroutD value of the Pre-Authorization. See the section **Follow On Transactions** for more information (see page 70).
BDate	String	The Business / Batch Date. If populated, this value will be placed in the Business Date column of the transaction record in the **PC**Charge database (pccw.mdb). **Format:** MMDDYY

Property Name	Data Type	Description - PccCharge Properties
Billpay	String	Only valid for Visa debit and credit transactions. It is used to indicate the transaction is being ran for payment of a bill (ultilty, monthly gym dues, etc.) **Valid values:** 0 – Non-Bill payment transaction 1 – Bill payment transaction
Card°	String	The credit card number that will be used when processing the transaction. **Max Length:** 20 characters. **Example:** 5424180279791765 **NOTE:** Is it the integrator's responsibility to remove spaces in the card number if there are spaces in the Track II card number data.
CardPresent	String	**For Retail or Restaurant transactions**: Flag that indicates whether the card was present. **For eCommerce transactions**: Flag that indicates what type of transaction occurred. **Valid values:** 0 = Card not present, 1 = Card present (for Retail, MOTO, or Restaurant); D = Digital goods, P = Physical goods (for eCommerce)
Command	String	The action code that identifies what type of transaction will be performed. **Valid Values:** 1-10, 13-15, ZI, ZH. **Example:** If running a credit card sale, set the action code to "1". Consult the section **DevKit Constants** for a list of valid values (see page 93). **Note:** Because the Action property is defined as "long", this property was added to allow action codes that contain strings (such as Transaction Inquiry - ZI). If the Command property is set, its value will override the value set in Action.
CmrclCardFlag	String	The type of commercial card being submitted. The CommercialCardType function in the PccBin class should be used to retrieve the 1 character value from **PC**Charge that indicates what type of commercial card will be submitted. See the section **Commercial Card Transactions** (see page 93) for more information. **Max Length**: 1 character **Valid values:** B – Business P,L,G -- Purchase C – Corporate F – Fleet
CustCode	String	Customer code for purchasing/commercial cards. This property must be set for commercial card transactions in order to get the best discount rate. Additionally, the transaction's action code must indicate that the transaction is a commercial card transaction. **Note:** Global East (NDC), terminal based, requires the customer code be all upper case. **Max Length:** 25 characters, alphanumeric only.
CVV2	String	The CVV2 value for the transaction. The card verification value (CVV2 for Visa, CVC2 for MasterCard, and CID for AMEX and Discover) is a 3 or 4 digit number that is embossed in the signature panel for Visa, MasterCard, and Discover and on the front of the card for AMEX. All AMEX cards utilize a 4 digit CID. **Max Length:** 4 characters. CVV2 should only be passed on non-swiped transactions.
CreditPlanNumber	String	The credit plan numbers are established by the processor **CITI** for each merchant, they define the type of Disclaimer to print on receipts. This information will vary from merchant to merchant.
DriverID	String	Driver identification field. Only required for Wright Express, Voyager and Fleet One cards.
DriverPIN	String	Driver personal identification number. Only required for Fuelman cards.
DEST_ZIP_CODE	String	Destination Zip Code for American Express purchasing/commercial cards. This property must be set for American Express commercial card transactions when using American Express as the processor (or via split dial) in order to get the best discount rate. Additionally, the transaction's action code must indicate that the transaction is a commercial card transaction.
EnhancedTransFlag	*Boolean*	*Used internally*
ExpDate°	String	The expiration date associated with the credit card number that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208
ExtMsg	*String*	*Used internally*

Property Name	Data Type	Description - PccCharge Properties
GratuityAmnt	String	**For use with Restaurant transactions only.** The actual gratuity amount for a Sale with Gratuity (action code 14) , Gratuity (action code 13) , or Post-Authorization (action code 5) transaction. See the section **Restaurant Transactions** (see page 105) for more information.
GratuityAmntEst	String	**For use with Restaurant transactions only.** The estimated gratuity amount for a Sale (action code 1) or Pre-Authorization (action code 4) transaction. If the GratuityAmntEst is populated, **PC**Charge will submit the sum of the values in the Amount and GratuityAmntEst fields for authorization. If the transaction is authorized, only the value in the Amount field will be placed in the **PC**Charge settlement file (if running a Sale). By using the GratuityAmntEst, the merchant can help ensure that the customer has enough available credit on their card to leave a tip. Once the customer indicates the amount of the tip that will be left, a gratuity transaction (action code 13) must be performed on the sale prior to settlement in order to add the actual gratuity to the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. See the section **Restaurant Transactions** (see page 105) for more information. **Note:** It is recommended to check with the processor or merchant service provider for guidance on what amount to set this value to. Incorrectly setting this value can result in downgrades.
IDNumber	String	Only required for Voyager cards, dependant on Restriction Code. Four to six digits. Note: Only used for Pre-Authorization transactions **Update:** Beginning with PCCharge version 5.8, this is also used for Citi© Private Label.
ImpTransFlag	*Boolean*	*Used internally*
IsPurchaseCard	*Boolean*	*Used internally*
ItemCodes	String	The Item ID for the transaction. This field is only used for Chase Paymentech (GSAR) and can store five (5) four-digit codes that are defined by Chase Paymentech. **Example:** If ItemCodes is set to 0001000200030004 0005, it stores 5 item IDs (0001, 0002, 0003, 0004, and 0005). These numbers must be obtained from Chase Paymentech.
LanguageCode	*String*	*Used internally*
LastValidDate	String	The last year that will be considered a valid expiration date. This value can be set programmatically or through the **PC**Charge GUI. **Length:** 2 digits. **Format:** YY **Example:** If LastValidDate is set to 05, then cards between 06 and 99 are considered to be 1906 to 1999, and cards between 00 and 05 are 2000 to 2005.
MACData	*String*	*Used internally*
MACState	*MACState*	*Used internally*
Manual°	Long	Flag that indicates whether the transaction was manually entered or swiped. If the transaction was swiped, the Track property must also be set. **Valid values:** 0 = manual transaction, 1 = swiped transaction
MCSC	String	The Multiple Count Sequence Count. This is the total number of installments that will be charged in a non-restaurant recurring billing scenario. **Max Length**: 2 characters. **Example:** If there are 5 payments to be made, set this property to "5".

Property Name	Data Type	Description - PccCharge Properties
MCSN	String	**In a restaurant environment:** The server or cashier id. **Max Length:** 2. This field should be passed for reporting and reconciliation purposes. See the section **Restaurant Transactions** (see page 105) for more information. **Processor specific note:** The Server ID is required for AMEX card transactions. Also required when using the processor NB and GSAR in restaurant business type. **In a non-restaurant environment**, this field is the Multiple Count Sequence Number. This is the transaction number within the total number of payment installments in a recurring billing scenario. **Max Length**: 2 characters. **Example:** If there are 5 payments to be made and this transaction is the first transaction, set this property to "1". The first transaction should also include the CVV property, but this value should **not** be stored or sent for subsequent transactions.
member	String	The cardholder's name. **Max Length:** 20 characters.
MerchantNumber° ***	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Credit Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
Method	*TxnMethodType*	*Used internally*
Multi	String	Flag that indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PC**Charge GUI. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid values:** 1 = TRUE, 0 = FALSE **Default value:** 0. See the section **Multi-trans Wait** for more information (see page 67). This Flag has no effect if processing will occur over IP or leased line.
Odometer	String	The odometer reading. Only required for Fleet One (7 digits), Voyager (7 digits), and Fuelman (6 digits) cards.
Offline	String	Flag that indicates whether **PC**Charge should process the transaction offline. If the offline flag is set, **PC**Charge will put the transaction into a .BCH file that resides in the **PC**Charge directory for importing at a later time. The file can only be imported from the **PC**Charge GUI. **Valid values:** 1 = TRUE, 0 = FALSE
Path°	String	The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
PeriodicPayment	String	Flag that indicates whether the transaction is a recurring transaction. **Valid values:** 1 = TRUE, 0 = FALSE **Note:** If periodic payment is set to true, the recurring billing properties must also be set to achieve the best processing rates.
PrintReceipts	String	The number of receipts that **PC**Charge should print for the transaction. This value will override the corresponding value in the **PC**Charge GUI. **PC**Charge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.
Processor° ***	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
ProductDetailAmount_XX	String	**Note:** Only required for the processor NBS. This is the total dollar amount for PRODUCT_DETAIL_PRODUCT_CODE_XX being authorized. For example, PRODUCT_DETAIL_PRODUCT_CODE_1 has a PRODUCT_DETAIL_QUANTITY_1 = 2 and a PRODUCT_DETAIL_UNIT_PRICE_1 = $2.00, therefore the PRODUCT_DETAIL_AMOUNT_1 = $4.00

Property Name	Data Type	Description - PccCharge Properties
ProductDetailCount	String	**Note:** Only required for the processor NBS. All card types are configurable except for Fleet One which is limited to 7 records. Only 1 – 10 records are currently supported through PCCharge for all card types.
ProductDetailCode_XX	String	**Note:** This is the number of items for RODUCT_DETAIL_PRODUCT_CODE_XX. PCCharge will support 1 – 10. • NBS supports using 1 – 10. • BAMS Tandem (NPC) supports using 1 – 3.
ProductDetailQuantity_XX	String	**Note:** Only used for the processor NBS. This is the unit price for PRODUCT_DETAIL_PRODUCT_CODE_XX. This is only used for Fleet One and Fuelman. Currently, PCCharge will support 1 – 10.
Reference	String	The reference number from the original transaction (returned by the processor). Set this property only if processing a Post-Authorization and the Post-Authorization is being used to add a Voice-Authorization to the batch or to "store" a Voice-Authorization. (For information on stored Voice-Authorizations, see page 91). The Reference property does not need to be set if the Post-Authorization completes a standard Pre-Authorization using the TroutD value of the Pre-Authorization. See the section **Follow On Transactions** for more information (see page 70). **Max Length:** 8 characters. **Note:** NBS/ Fleet One cards require a Reference Number to be sent with each transaction. This is a minimum of 2 digits and a max of 15. This must be all numeric.
RestrictionCode	String	Only required for Voyager cards. This is used to determine the level of identification and which fields are required. Two digits. Valid Values: 00 - No ID Number or Odometer required. Fuel and Other allowed. 01 - No ID Number or Odometer required. Fuel only allowed. 10 - ID Number only required. Fuel and Other allowed. 11 - ID Number only required. Fuel only allowed. 20 - Odometer only required. Fuel and Other allowed. 21 - Odometer only required. Fuel only allowed. 30 - ID Number and Odometer required. Fuel and Other allowed. 31 - ID Number and Odometer required. Fuel only allowed. **Note:** Required for both manual and swiped transactions.
RFID	String	Set to 1 if card information was read from RFID (Radio Frequency Identification) device. If card was read from from RFID, track data must be populated and manual flag must be set to 1. Set to 0 otherwise.
ShiftID	*String*	*Used internally*
SmartTermMsg	*String*	*Used internally*
SmartTermRequest	*Boolean*	*Used internally*
Store	String	Flag indicating whether a Voice Authorization transaction should be stored. This flag should only be submitted when performing a Post-Authorization transaction (action code 5) that includes an authorization code from the voice operator. For more information on stored Voice-Authorizations, see page 91. **Valid Value:** 1 - Store the Voice Authorization transaction.
Street	String	The cardholder's billing street address. The Street property is used for address verification. Address verification can only be performed on non-swiped transactions. **For FDC:** Use first 5 digits only. **Note:** For manually keyed transactions, Street is required to qualify for the lowest transaction rates. **Max Length:** 20 characters **Citi**© - When used with Citi Plan PO Box's are not allowed.
TaxAmt	String	The tax amount. This is the portion of the amount that is tax. Providing the tax amount is required to obtain the best rate on commercial card transactions. **Max Length**: 9 characters (including the decimal). **Format:** DDDDDD.CC. The transaction's action code must indicate that it is a commercial transaction. Tax amount should be included in the amount field.
TaxExempt	Boolean	Tax Exempt Flag. This flag is used to indicate if the purchase is tax exempt. Used only for Commercial Card Transactions. **Valid Values:** 1 – Purchase is tax exempt; 0 – Purchase is not tax exempt.

Property Name	Data Type	Description - PccCharge Properties
Ticket	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** For manually keyed transactions, Ticket is required to qualify for the lowest transaction rates. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.
TimeOut	Long	The number of seconds after which a timeout error will be returned from **PC**Charge. The count will start when the Send method is called. It is highly recommended that integrators review the section **Timeouts** (see page 59). Setting the TimeOut value improperly could cause reconciliation issues and problems such as double-charging a customer's account. **Note:** The TimeOut properly is only applicable when programming in an asynchronous manner.
TotalAmount	*String*	*No longer needed*
Track	String	The track II data captured from the magnetic strip of the credit card. The track II data is required to ensure the lowest per-transaction rate from the processing company when performing swiped transactions (Retail and Restaurant). Sending the track II data is not allowed if the merchant's industry type is MOTO or eCommerce. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in.
TroutD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
User° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).
VehicleID	String	Only required for Wright Express cards (5 digits) and Voyager cards (8 digits). Note: Required for both manual and swiped transactions.
Zip	String	The cardholder's zip code. The Zip property is used for address verification. **Max Length:** 9 digits. Address verification can only be performed on non-swiped transactions. **Note:** For manually keyed transactions, the Zip is required to qualify for the lowest transaction rates. **Note:** If submitting the 9-digit zip, do not include the dash. **Citi**© - When used with Citi Plan the zip is limited to 5-digits.
CashierName	String	Submit Cashier's login name.
Password	String	Submit Cashier's password.
NewCashierPassword	String	Submit a new password for the Target Cashier. **Note:** case-sensitive Must be complex: • minimum 7 characters • must have at least one upper case character • one numeric character one special character (e.g., @, $, %, etc.) **NOTE:** See **Cashier Permissions** on page 75 for an example.
NewCashierConfirmPassword	String	Resubmit the password for verification. **Note**: Must match the original NewCashierPassword **NOTE:** See **Cashier Permissions** on page 75 for an example.
AddCashier1	String	Allows the integrator to submit new cashiers up to 5 at a time. Requires 3 Name Value pairs to be entered for each AddCashier tag submitted. **CASHIER_NAME <Name>** **PASSWORD <Password>** **PERMISSIONS <Permissions>** Example: <ADD_CASHIER_1>CASHIER_NAME CreditR PASSWORD CreditR123%
AddCashier2	String	
AddCashier3	String	
AddCashier4	String	

Property Name	Data Type	Description - PccCharge Properties
AddCashier5	String	PERMISSIONS 10000|000|000|000|000000000000|0000|000000 </ADD_CASHIER_1> **NOTE:** See **Cashier Permissions** on page 75 for an example.
TargetCashierName	String	Submitted when performing a function to edit a cashier's settings. This specifies which cashier's information to alter. **NOTE:** See **Cashier Permissions** on page 75 for an example.
CustomerFirstName [1]	String	Applicant's first name.
MiddleInt [1]	String	Applicant's middle initial.
CustomerLastName [1]	String	Applicant's last name.
Suffix [1]	String	Applicant's suffix. (Ex. M.D., Ph.D, Jr.)
AptSuite [1]	String	Applicant's apartment or suite number.
CustomerCity [1]	String	Applicant's city.
State [1]	String	Applicant data. (Ex: GA, FL, MA…)
EMail [1]	String	Applicant data. (Ex: XXX@XX.XXX)
Phone_Number [1]	String	Applicant data. Format xxxxxxxxxx 10-digits No Dashes
SSNum [1]	String	Applicant data. Format xxxxxxxxx 9-digits No Dashes
Birth_Date [1]	String	Applicant data. Format is MMDDYYYY.
EmpName [1]	String	Applicant data. Employer name.
WorkPhoneNumber [1]	String	Applicant data. Employer phone number.
HomeOwner [1]	String	Applicant data. Format: '**O**' = Own '**R**' = Rent '**B**' = Board '**P**' = Live w/ parents '**M**' = Military
AnnualIncome [1]	String	Applicant data. Format: Up to 6 digits. Whole dollar amount with no decimals.
PhotoIDState [1]	String	Applicant data. (Ex: GA, FL, MA…)
CorrelationUID [1]	String	Received as a response to a Credit App (P3)
PendingNumber [1]	String	Received as a response to a Credit App (P3)
ReplyStatusFlag [1]	Boolean	Reply notification. Format: **Y** = Accept **N** = Decline
SourceCode [1]	String	**Citi Plan** Format: Opt In - **"AP"** Opt Out - **"AO"**
IDType [1]	String	Applicant data. Type of identification being submitted. Format: **'D'** - Driver's License **'O'** - Other
FraudFlag [1]	Boolean	Code 10. Values are **1** or **0: 1** = True, **0** = False
AmountHealthCare [#]	String	Total Healthcare Amount. This amount must be greater than or equal to the sum of the other amount categories. Max: **12 - digits** Format: **DDD.CC**
AmountPrescription [#]	String	(Optional) Total amount of the prescription-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
AmountVision [#]	String	(Optional) Total amount of the vision-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**

Property Name	Data Type	Description - PccCharge Properties
AmountClinic #	String	(Optional) Total amount of the clinic-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
AmountDental #	String	(Optional) Total amount of the dental-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
AmountTransit #	String	(Optional) Reserved for future use. Max: **12 - digits** Format: **DDD.CC**
AmountCopay #	String	(Optional) Reserved for future use. Max: **12 - digits** Format: **DDD.CC**
FSA #	Boolean	Indicates the transaction is FSA. This will bypass PCCharge BIN checking as PCCharge will assume that the POS has verified that this card has met the BIN requirements for an FSA transaction. **1** = True, **0** = False **Note:** If this is not passed it defaults to False.

° These properties are the minimum required to process a Sale or Pre-Authorization transaction.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

*** If the "Use Default Processor" option is enabled in the **PC**Charge preferences, and the `Processor` and `MerchantNumber` properties are omitted from the transaction request, **PC**Charge will process all transactions using the "Default Processor". The "Default Processor" is defined as the first merchant number that is set up **PC**Charge. Consult the **Multi-Merchant Support** section (see page 68) for more information on the "Use Default Processor" option. In addition, `Processor` and `MerchantNumber` should not be set when doing follow-on transactions. Refer to the section **Follow On Transactions** (see page 70) for more information.

[1] These properties are required to process **Citi applications**. For more information and examples see **Citi Credit Plan** on page 116.

Additional tags used to process FSA/HRA transactions. For more information please see the section about **FSA on page 99**.

PccCharge Methods

Method Name	Returned Value	Description - PccCharge Methods
Cancel	None	Cancels transaction in progress
CheckPCard	Boolean	Used to determine whether a credit card is a commercial card or not. This method requires that a credit card number be passed as a parameter. Returns TRUE if a commercial card, FALSE otherwise.
DeleteUserFiles	None	The `DeleteUserFiles` method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the `User` property. The `DeleteUserFiles` method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files, the `Error` event will be triggered (if asynchronous) and the `GetErrorDesc` method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
GetACI	String	Returns the Authorization Characteristics Indicator is that is provided by the card associations. This value is stored for settlement.
GetAddText1	String	Only supported on Fleet One, this field contains miscellaneous additional text returned from host. Currently PCCharge will support GetAddText1-GetAddText4.

Method Name	Returned Value	Description - PccCharge Methods
GetAmountDue	String	Returns the amount due, only for Elavon (NOVA) pre-paid and FSA functionality. **Format is DDD.CC** **Note:** "Enable Pre-Paid Cards" option within PCCharge MUST be checked for this feature to work. The option is located under Setup>Credit Card Company>Extended Data and only with the Credit Card Company set to Elavon (NOVA). This option is only configurable in *PCCharge version 5.7.1 release I sp9a* and above. **Note:** For this to work with FSA transactions, one must enable FSA and Partial Auth within the Extended Data Screen for supported processors.
GetAuth	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
GetAuthAmount	String	Used in partial redemption transactions where only part of the amount was authorized. Returns the actual authorized amount.
GetAVS	String	Returns the AVS response code from the issuing bank. If performing Address Verification on card-not-present transactions, this code indicates how well the AVS information passed in matches what the issuing bank has on file for the cardholder. Consult the section **DevKit Constants** for a description of values that may be returned (see page 143)
GetCaptured	Boolean	The GetCaptured method returns TRUE if **PC**Charge returns "CAPTURED" as the result of the transaction. Otherwise, FALSE will be returned. The GetCaptured method is used to determine if a transaction that will result in a monetary transfer (Sale, Credit, Post-Authorization, etc.) is approved or declined. A "CAPTURED" response indicates that the transaction has been approved, and that the transaction has been placed in the open batch.
GetCreditCardType	String	The GetCreditCardType method returns the abbreviation of the credit card issuer. This method requires that a credit card number be passed as a parameter. Consult the section **DevKit Constants** for descriptions of values (see page 143). (GetCreditCardType is the same as GetCardIssuer).
GetCCAvailBalance	String	**Pre-paid credit cards with Elavon (NOVA) -** Returns the PrePaid card balance. **FSA -** Returns the available balance on the FSA/HRA card. **Format: DDD.CC**
GetCVV2	String	Returns the CVV2/CVC2/CID response code from the issuing bank. If performing CVV2/CVC2/CID validation on card-not-present transactions, this code indicates if the CVV2/CVC2/CID code passed in matches what the issuing bank has on file for the cardholder. Consult the section **DevKit Constants** for a description of values that may be returned (see page 143)
GetDCAvailBalance	String	Returns the available balance on pre-paid debit cards. Only for pre-paid debit cards with Elavon (NOVA).
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).

Method Name	Returned Value	Description - PccCharge Methods
GetHostType	Integer	The GetHostType method returns an integer that indicates if a processor / merchant number is Host based or Terminal based. GetHostType requires three parameters: 1) **Processor code** - Consult the section **DevKit Constants** (see page 143) for a list of valid processor codes 2) **Merchant account** - Must be a valid merchant account set up in **PC**Charge 3) **TID type** - Valid Values for TID type: 0 – Credit; 1 – Check; 2 – Debit; 3 – EBT; 4 – GiftCard GetHostType will return one the following values based on the parameters passed in: 0 – The processor is Terminal based 1 – The processor is Host based -1 – The processor is a Hybrid (supports both Host and Terminal processing) or invalid processor / merchant number. **Example:** .GetHostType("VISA", "999999999911", 0) will return 0 **Note:** Chase Paymentech (GSAR), Elavon (NOVA), and FDMS South / NaBanco (NB) are considered hybrid processors. GetHostType will return a -1 for these processors.
GetIND	String	Returns the IND code. The IND code is a transaction description code and an Interchange compliance field. This value is not returned by all processing companies.
GetMSI	String	Returns the Market Specific Indicator. This value indicates the transaction's market segment. This value is assigned by the card associations and is not returned with all transactions.
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetPCard	String	Returns 1 if **PC**Charge recognizes the card as a purchasing/corporate card. Otherwise, **PC**Charge returns 0.
GetPEM	String	Returns the Point of Entry Mode that is associated with the transaction. This value is not returned by all processing companies.
GetPS2000	String	PS2000 Data. This data will be as received during the original authorization processing. It will not be present for offline transactions. PS2000 Data is a variable; it will either be one character or up to 20. It is data concerning the card type and transaction that the processor will send back during the authorization process. This value is not returned by all processing companies.
GetRecordCount	String	The number of records matching the inquiry (ZI command).
GetRefNumber	String	Returns the reference number associated with the transaction. The reference number is assigned by the card associations. The reference number is used to help identify the transaction and is useful for the cardholder and merchant when doing research. This value is not returned with all transactions.
GetRespCode	String	Returns the response code that is provided by the processor. This value is not returned by all processing companies.
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
GetRestrictCode	String	**Note:** Only supported on Fleet One. The product restriction code.
GetRET	String	Returns the Retrieval reference number. This value is not returned by all processing companies
GetTBatch	String	Returns the active batch number for the transaction. This value is not returned by all processing companies.
GetTDate	String	Returns the date that the transaction was processed. This value is not returned by all processing companies.

Method Name	Returned Value	Description - PccCharge Methods
GetTerminationStatus	String	Retrieves Termination Status which returns a "6" upon successful transaction. If anything other than a "6" is returned there may be an issue with the settlement file and should be investigated. This applies to : **GSAR** and **NBS** **Response:** **6 = Successful Transaction**
GetTI	String	Returns the Transaction Identifier that is returned from the processor. This value is not returned by all processing companies.
GetTicket	String	Returns the ticket number or invoice of the transaction. This value is echoed back from the original transaction or is generated by **PC**Charge if one is required to complete the transaction.
GetTICode	String	Returns the Transaction Indicator Code that is returned from the processor. The Transaction Indicator Code is a Validation code for VISA / MasterCard. This value is not returned by all processing companies.
GetTIM	String	Returns the Time of the transaction. This value is not returned by all processing companies.
GetTitem	String	Returns the Transaction Item number or the number that is associated with the transaction in the settlement file. This value is not returned by all processing companies.
GetTraceNumber	String	Returns the trace number from the processor. Only for pre-paid credit cards with Elavon (NOVA).
GetTransNum	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
GetTransRecord	String	Contains nested XML tags providing information on transaction(s) pulled from Trans table in the PCCharge database (pccw.mdb) (ZI command).
GetTransactionReferenceNumber	String	Returns the transaction reference number from the processor. Only for pre-paid credit cards with Elavon (NOVA).
GetTroutD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
GetUpdateData	*String*	*Used internally*
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved from the .oux file can be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.
PccSysExists	Boolean	The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions.

Method Name	Returned Value	Description - PccCharge Methods
Send	None	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the class will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has two optional parameters. The first parameter indicates whether the Send method will process transactions synchronously or asynchronously. **Note:** The object must defined to use events in order to allow asynchronous communication. **Valid Values:** True – process asynchronously **(Default)** False – process synchronously The second parameter indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send True, 3 **Note:** The other values that appear in the enumerated list are for internal use only-- do not attempt to use any values other than the ones listed above.
ValidCardLength	Boolean	Returns TRUE for card of correct length
ValidCardLengthII	Boolean	Returns TRUE for card of correct length
ValidDate	Boolean	The ValidDate method returns TRUE if the expiration date provided in the ExpDate property is valid, or FALSE if it is not.
ValidIssuer	Boolean	Returns TRUE for valid card issuer
VerifyAmount	Boolean	The VerifyAmount method returns TRUE if the amount provided in the Amount property is in a valid format (DDDDDD.CC), or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
VerifyCreditCard	Boolean	The VerifyCreditCard method returns TRUE if the credit card number's format is valid and meets the requirements set forth by the credit card companies, FALSE if it does not. If FALSE is returned, use the GetErrorCode and GetErrorDesc methods to determine the reason for failure. VerifyCreditCard has a required string parameter in which the credit card number to be checked must be passed.
VerifyExpDate	Boolean	The VerifyExpDate method returns TRUE if the expiration date provided in the ExpDate property is correct and in the right format, or FALSE if it is not. VerifyExpDate calls the ValidDate function to validate the expiration date. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
VerifyMerchantNumber	Boolean	The VerifyMerchantNumber method returns TRUE if the merchant number that is passed to it is set up in **PC**Charge, otherwise, FALSE is returned. Specifically, this method checks for the merchant number in the file TID.PCC, which is located in the **PC**Charge directory. The Path property must be set before calling this Method.
VerifyProcessor	Boolean	Returns TRUE if processor is valid
GetCorrelationUID [1]	String	Returned from Citi, needed for Counter Offer Submission.

Method Name	Returned Value	Description - PccCharge Methods
GetPendingNumber [1]	String	Returned from Citi, needed for Counter Offer Submission.
GetReplyFlag [1]	String	**'Y'** or **'N'** sent to Citi to notify message is a reply to an offer.
GetOpentoBuy [1]	String	Only present in the response if the "Display OTB" is turned ON. Decimal is included.
GetCreditLimit [1]	String	Only present the in the response. Retrieves credit limit.

[1] These properties are required to process **Citi applications**. For more information and examples see **Citi Credit Plan** on page 116.

PccCharge Events

Event Name	Description - PccCharge Events
Error	The Error event is fired any time an error occurs in the class. Once an Error event has fired, call GetErrorCode and GetErrorDesc to determine what kind of error has occurred. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
Finish	The Finish event will fire when the transaction has been completed. This means that **PC**Charge has processed the transaction successfully and has placed a file with the extension of .oux in the **PC**Charge directory. The name of the .oux file will be what was set in the User property of the transaction request. Call the GetResult method to determine whether or not the transaction was approved. A list of valid results can be found in the **Transaction Result Constants** section (see page 156).

Note: When doing asynchronous transactions in an event-driven programming model, it is important to place all result or error routines in either the Finish or Error events. Do not place any code that uses the .get methods after invoking the Send method.

PCCDebit Class

The PCCDebit class provides integrators with properties and methods used to submit credit card transactions to **PC**Charge. To use the PCCDebit class to integrate transaction processing, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to process transactions by using the PccSysExists method.

2. Assign the appropriate values to the properties required for the transaction to be performed and validate the values using the various .Verify methods. (The properties marked with a ° in the **PCCDebit Properties** table are the minimum required to process a Debit Sale transaction.)

3. Call the Send method. (**Note:** When calling the Send method, it is recommended that "3" is passed as a parameter to activate the XML message format)

4. If programming asynchronously, wait for the Error or Finish event to occur.

5. If programming synchronously, code using the .Get methods may be placed immediately after the Send method

6. Call the various .Get methods to determine the outcome of the transaction. The most important information can be acquired by calling the GetResult and GetAuth methods. If an error occurs, call the GetErrorCode and GetErrorDesc methods to determine the nature of the error.

7. Call the DeleteUserFiles method to delete all files related to the transaction.

8. Destroy the object.

When processing debit cards, a PINpad is required to allow the customer to enter their PIN. In addition, debit card information is always collected via a card swipe device, never via keyboard entry. Because of this, a card reader is also required.

When processing U.S. debit card transactions, merchants have the option of allowing the customer to receive cash back on a transaction. For instance, the customer purchases $50 of products and wants $25 cash back, set the Amount to 50.00 and CashBack to 25.00. This will withdraw a total of $75 from the debit card account, $50 for the products and $25 for cash to give to the customer.

Consult the section **Pseudo-code** (see page 158) for various examples that may be followed when using the Debit class to perform transaction processing.

For information on integrating Canadian Debit, see the section **Canadian (Interac) Debit Transactions** (see page 113).

This is a Multi Use class.

Note: **PC**Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

PCCDebit Properties

Property Name	Data Type	Description - PCCDebit Properties
Action° °°	Long	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
Amount° °°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
AuthCode	*String*	*The AuthCode field is only applicable to EBT transactions.*
BDate	String	The Business / Batch Date. If populated, this value will be placed in the Business Date column of the transaction record in the **PC**Charge database (pccw.mdb). **Format:** MMDDYY
Billpay	String	Only valid for Visa debit and credit transactions. It is used to indicate the transaction is being ran for payment of a bill (ultilty, monthly gym dues, etc.) **Valid values:** 0 – Non-Bill payment transaction 1 – Bill payment transaction
Card° °°	String	The Debit card number that will be used when processing the transaction. **Max Length:** 20 characters. **Example:** 5424180279791765
CashBack	String	The amount of cash back that the customer will receive. This amount is in addition to value entered in Amount property. For example, if the total amount of the sale is $10 and the customer has requested $5 cash back, Amount should be set to $10 and CashBack should be set to $5. The CashBack property should be formatted the same the Amount property. **Max Length:** 9 characters. **Note:** Some debit processors do not support the cash back feature.
DebitType°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Set this to the bank account type that the customer specified when entering transaction data into the PINpad. **Valid Values:** "Chequing" or "Savings"
DriverID	String	Driver identification field. Only required for Wright Express, Voyager and Fleet One cards.
DriverPIN	String	Driver personal identification number. Only required for Fuelman cards.
EBTType	*String*	*Used internally*
EnhancedTransFlag	*Boolean*	*Used internally*
ExpDate° °°	String	The expiration date associated with the Debit card number that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208 Set this property if there is an expiration date associated with the Debit card.
ExtFile	*File*	*Used internally*
Gratuity°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. This is the Gratuity Amount of the transaction.
IDNumber	String	Only required for Voyager cards, dependant on Restriction Code. Four to six digits. **Note:** Only used for Pre-Authorization transactions **Update:** Beginning with PCCharge version 5.8, this is also used for Citi© Private Label.

Property Name	Data Type	Description - PCCDebit Properties
KeySerialNumber°°	String	If a Key Serial Number is returned from the PINpad, this property should be populated with that number. **If processing transactions with a PINpad using DUKPT encryption**, this value is sixteen or twenty characters long (depending on the processor's encryption). The **PC**Charge DevKit provides several tools for retrieving data from PINpads. If the **PC**Charge integration method chosen doesn't support these tools or the tools do not support the PINpad being used, a direct interface to the PINpad must be written by the integrator. **If processing transactions with a Verifone SC5000 PINpad**, set this property to the Chip Serial Number of the PINpad.
LanguageCode°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Set this to the language that is indicated by the Language Code that is encoded in the track II data on the customer's card. **Valid Values:** "English" or "French" (pass in the literal string)
MACData°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Set this to the MAC Block value returned by the PINpad.
MACState	*MACState*	*Used internally*
Manual° °°	Integer	Flag that indicates whether the transaction was swiped or manually entered. This property must be set to 1 (swiped) and the Track property must also be set.
member	String	The cardholder's name. **Max Length:** 20 characters.
MerchantNumber° °° ***	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Debit Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
Method	*TxnMethodType*	*Used internally*
Multi	Boolean	Flag that indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PC**Charge GUI. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid values:** 1 = TRUE, 0 = FALSE **Default value:** 0. See the section **Multi-trans Wait** for more information (see page 67). This Flag has no effect if processing will occur over IP or leased line.
Odometer	String	The odometer reading. Only required for Fleet One (7 digits), Voyager (7 digits), and Fuelman (6 digits) cards.
OrigPurchData	String	The Original Purchase Data. Used when performing a Debit Return with the processors TSYS, Heartland, RBS WorldPay, and NPC. This is the original transaction date. **Format:** DDMMhhmm
Path° °°	String	The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Pin° °°	String	The encrypted PIN block that is retrieved from the PINpad. The PIN is provided to the processor for verification. **Length:** 16 characters. The **PC**Charge DevKit provides several tools for retrieving data from PINpads. If the **PC**Charge integration method chosen doesn't support these tools or the tools do not support the PINpad being used, a direct interface to the PINpad must be written by the integrator.
Processor° °° ***	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).

Property Name	Data Type	Description - PCCDebit Properties
ProductDetailAmount_XX	String	**Note:** Only required for the processor NBS. This is the total dollar amount for PRODUCT_DETAIL_PRODUCT_CODE_XX being authorized. For example, PRODUCT_DETAIL_PRODUCT_CODE_1 has a PRODUCT_DETAIL_QUANTITY_1 = 2 and a PRODUCT_DETAIL_UNIT_PRICE_1 = $2.00, therefore the PRODUCT_DETAIL_AMOUNT_1 = $4.00
ProductDetailCount	String	**Note:** Only required for the processor NBS. All card types are configurable except for Fleet One which is limited to 7 records. Only 1 – 10 records are currently supported through PCCharge for all card types.
ProductDetailCode_XX	String	**Note:** This is the number of items for RODUCT_DETAIL_PRODUCT_CODE_XX. PCCharge will support 1 – 10. • NBS supports using 1 – 10. • BAMS Tandem (NPC) supports using 1 – 3.
ProductDetailQuantity_XX	String	**Note:** Only used for the processor NBS. This is the unit price for PRODUCT_DETAIL_PRODUCT_CODE_XX. This is only used for Fleet One and Fuelman. Currently, PCCharge will support 1 – 10.
Reference	String	NBS/ Fleet One cards require a Reference Number to be sent with each transaction. This is a minimum of 2 digits and a max of 15. This must be all numeric.
RestrictionCode	String	Only required for Voyager cards. This is used to determine the level of identification and which fields are required. Two digits. Valid Values: 00 - No ID Number or Odometer required. Fuel and Other allowed. 01 - No ID Number or Odometer required. Fuel only allowed. 10 - ID Number only required. Fuel and Other allowed. 11 - ID Number only required. Fuel only allowed. 20 - Odometer only required. Fuel and Other allowed. 21 - Odometer only required. Fuel only allowed. 30 - ID Number and Odometer required. Fuel and Other allowed. 31 - ID Number and Odometer required. Fuel only allowed. **Note:** Required for both manual and swiped transactions.
ShiftID	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. The Shift ID. This value is optional. **Format:** Alphanumeric **Max Length:** 1 character.
SmartTermMsg	*String*	*Used internally*
SmartTermRequest	*Boolean*	*Used internally*
Ticket	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.
TimeOut	Long	The number of seconds after which a timeout error will be returned from **PC**Charge. The count will start when the Send method is called. It is highly recommended that integrators review the section **Timeouts** (see page 59). Setting the TimeOut value improperly could cause reconciliation issues and problems such as double-charging a customer's account. **Note:** The TimeOut properly is only applicable when programming in an asynchronous manner.
Track° °°	String	The track II data captured from the magnetic strip of the card. The track II data is required. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in.
TroutD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
User° °° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).

Property Name	Data Type	Description - PCCDebit Properties
VehicleID	String	Only required for Wright Express cards (5 digits) and Voyager cards (8 digits). **Note:** Required for both manual and swiped transactions.
CashierName	String	Submit Cashier's login name.
Password	String	Submit Cashier's password.

° These properties are required to process a Debit Sale transaction.

°° These properties are required to process a Canadian Debit Sale transaction using Global Payments East (NDC) and the SC5000 PINpad.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

*** If the "Use Default Processor" option is enabled in the **PC**Charge preferences, and the Processor and MerchantNumber properties are omitted from the transaction request, **PC**Charge will process all transactions using the "Default Processor". The "Default Processor" is defined as the first merchant number that is set up **PC**Charge. Consult the **Multi-Merchant Support** section (see page 68) for more information on the "Use Default Processor" option. In addition, Processor and MerchantNumber should not be set when doing follow-on transactions. Refer to the section **Follow On Transactions** (see page 70) for more information.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

PCCDebit Methods

Method Name	Returned Value	Description - PCCDebit Methods
Cancel	None	Cancels current transaction
DeleteUserFiles	None	The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files, the Error event will be triggered (if asynchronous) and the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
GetAddText1	String	Only supported on Fleet One, this field contains miscellaneous additional text returned from host. Currently PCCharge will support GetAddText1-GetAddText4.
GetApproved	Boolean	The GetApproved method returns TRUE if **PC**Charge returns "APPROVED" as the result of the transaction. Otherwise, FALSE will be returned.
GetAuth	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
GetAuxRespCode	String	When using the SC5000 PINpad, returns the ISO response code
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetMSI	String	**For Debit Master Session:** Returns the new master key (if one exists) sent by the processor that should be passed to the PINpad.

Method Name	Returned Value	Description - PCCDebit Methods
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetPOSSequenceNumber	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Returns the current POS Sequence Number for the PINpad. The Path property must be set to the **PC**Charge directory and the PINpad's Chip Serial Number must be passed as a parameter when calling the GetPOSSequenceNumber method.
GetRefNumber	String	Returns the reference number associated with the transaction. The reference number is used to help identify the transaction and is useful for the cardholder and merchant when doing research. This value is not returned with all transactions.
GetRespCode	String	Returns the response code that is provided by the processor. This value is not returned by all processing companies.
GetRestrictCode	String	**Note:** Only supported on Fleet One. The product restriction code.
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
GetTI	String	Returns the Transaction Identifier that is returned from the processor. This value is not returned by all processing companies.
GetTICode	String	Returns the Transaction Indicator Code that is returned from the processor. This value is not returned by all processing companies.
GetTransNum	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
GetTroutD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
GetUpdateData	*String*	*Used internally*
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved from the .oux file can be used to view the results of a Transaction Inquiry (ZI) transaction. Refer to the section **Transaction Inquiry** (see page 128) for more information. The text can also be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.
PccSysExists	Boolean	The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions.

Method Name	Returned Value	Description - PCCDebit Methods
Send	None	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the class will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has two optional parameters. The first parameter indicates whether the Send method will process transactions synchronously or asynchronously. **Note:** The object must defined to use events in order to allow asynchronous communication. **Valid Values:** True – process asynchronously **(Default)** False – process synchronously The second parameter indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send True, 3 **Note:** The other values that appear in the enumerated list are for internal use only-- do not attempt to use any values other than the ones listed above.
VerifyAmount	Boolean	The VerifyAmount method returns TRUE if the amount provided in the Amount property is in a valid format (DDDDDD.CC), or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).

PCCDebit Events

Event Name	Description - PCCDebit Events
Error	The Error event is fired any time an error occurs in the class. Once an Error event has fired, call GetErrorCode and GetErrorDesc to determine what kind of error has occurred. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
Finish	The Finish event will fire when the transaction has been completed. This means that **PC**Charge has processed the transaction successfully and has placed a file with the extension of .oux in the **PC**Charge directory. The name of the .oux file will be what was set in the User property of the transaction request. Call the GetResult method to determine whether or not the transaction was approved. A list of valid results can be found in the **Transaction Result Constants** section (see page 156).

Note: When doing asynchronous transactions in an event-driven programming model, it is important to place all result or error routines in either the Finish or Error events. Do not place any code that uses the .get methods after invoking the Send method.

PccCheck Class

The PccCheck class provides integrators with properties and methods used to submit check verification, guarantee, and conversion transactions to **PC**Charge. To use the PccCheck class to integrate transaction processing, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to process transactions by using the PccSysExists method.

2. Assign the appropriate values to the properties required for the transaction to be performed and validate the values using the various .Verify methods. (The properties marked with a ° in the **PccCheck Properties** table are the minimum required to process a check verification transaction.)

3. Call the Send method. (**Note:** When calling the Send method, it is recommended that "3" is passed as a parameter to activate the XML message format)

4. If programming asynchronously, wait for the Error or Finish event to occur.

5. If programming synchronously, code using the .Get methods may be placed immediately after the Send method

6. Call the various .Get methods to determine the outcome of the transaction. The most important information can be acquired by calling the GetResult and GetAuth methods. If an error occurs, call the GetErrorCode and GetErrorDesc methods to determine the nature of the error.

7. Call the DeleteUserFiles method to delete all files related to the transaction.

8. Destroy the object.

Note: PC Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

PccCheck Properties

Property Name	Data Type	Description - PccCheck Properties
Account_Number°°	String	For Check, MICR, or Double ID: The account number that will be used when processing the transaction. **Max Length:** 20 characters.
Action°	Long	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
Amount°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
Birth_Date°°	String	The date of birth of the check writer. **Length:** Exactly six characters. **Format:** MMDDYY. The birth date is required for DL (Driver's License) check transactions.
Cash_Back	String	The amount of cash back that the customer will receive. This amount is in addition to value entered in Amount property. For example, if the total amount of the sale is $10 and the customer has requested $5 cash back, Amount should be set to $10 and CashBack should be set to $5. The CashBack property should be formatted the same the Amount property. **Max Length:** 9 characters. **Note:** Some processors do not support the cash back feature.
CashierNum	String	The Cashier Number
Check_Number°	String	The check number of the check that will be used when processing the transaction. **Max Length:** 10 characters.
Drivers_License°°	String	The driver's license number of the individual writing the check. **Max Length:** 20 characters. The driver's license is required for DL (Driver's License) transactions and when performing Double ID transactions.
EnhancedTransFlag	*Boolean*	*Used internally*
LanguageCode	*String*	*Used internally*
ManagerNum	String	Used for BPS Double ID transactions. Optional Manager Number for manager override.
Manual°	Long	Flag that indicates whether the transaction was manually entered or swiped. **Valid values:** 0 = manual transaction, 1 = swiped transaction
MerchantNumber°	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Check Services Setup** window of **PC**Charge. **Max Length:** 32 characters. This value can be alphanumeric.
Method	*TxnMethodType*	*Used internally*
MICR	String	The raw MICR data from the bottom of the check. Used for conversion transactions.
Multi	Boolean	Flag that indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PC**Charge GUI. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid values:** 1 = TRUE, 0 = FALSE **Default value:** 0. See the section **Multi-trans Wait** for more information (see page 67). This Flag has no effect if processing will occur over IP or leased line.

Property Name	Data Type	Description - PccCheck Properties
Path°	String	The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Phone_Number°°	String	The phone number of the individual writing the check. **Max Length:** 7 digits. **Format:** digits only. The phone number is required for COD (Checks On Delivery).
Processor°	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
Services	Integer	The type of check verification to be performed. **Valid values:** 0 – MICR 1 – Checks-On-Delivery 2 – Driver's License 3 – Double ID **Note:** The value set in the Services property overrides the value set in the Action property.
ShiftID	*String*	*Used internally*
State°°	String	The state code of the state that issued the check writer's driver's license. The state code is required for DL (Driver's License). **Format:** 2 characters.
Ticket	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.
TimeOut	Long	The number of seconds after which a timeout error will be returned from **PC**Charge. The count will start when the Send method is called. It is highly recommended that integrators review the section **Timeouts** (see page 59). Setting the TimeOut value improperly could cause reconciliation issues and problems such as double-charging a customer's account. **Note:** The TimeOut properly is only applicable when programming in an asynchronous manner.
Transit_Number°°	String	The Transit Routing Number / ABA number that will be used when processing the transaction. This value indicates which bank issued the check. **Max Length:** 9 characters. This value is required for MICR transactions and when performing Double ID transactions.
TroutD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
User° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).
Zip_Code°°	String	The check writer's ZIP code. **Max Length:** 9 characters. **Format:** digits only. This value is required for COD transactions. **Note:** If submitting the 9-digit zip, do not include the dash.

Note: To perform Double ID, both the MICR and Drivers_License fields must be populated.

° These properties are required, regardless of service type.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

°° COD -- required for Checks-On-Delivery
DL -- required for Driver's License
MICR -- required for MICR

PccCheck Methods

Method Name	Returned Value	Description - PccCheck Methods
`Cancel`	None	Cancels transaction in progress
`DeleteUserFiles`	None	The `DeleteUserFiles` method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the `User` property. The `DeleteUserFiles` method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files, the `Error` event will be triggered (if asynchronous) and the `GetErrorDesc` method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
`GetApproved`	Boolean	The `GetApproved` method returns `TRUE` if **PC**Charge returns "APPROVED" as the result of the transaction. Otherwise, `FALSE` will be returned. An "APPROVED" response indicates that a Verification has been approved.
`GetAuth`	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
`GetErrorCode`	Long	The `GetErrorCode` method returns an error code if an error was encountered during the use of various methods such as the `Send`, `Cancel`, `DeleteUserFiles`, and `PccSysExists`. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
`GetErrorDesc`	String	The `GetErrorDesc` method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
`GetMerchantNumber`	String	Returns the merchant number that was specified in the MerchantNumber property.
`GetRefNumber`	String	Returns the reference number associated with the transaction. The reference number is used to help identify the transaction and is useful for the check writer and merchant when doing research. This value is not returned with all transactions.
`GetResult`	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
`GetTroutD`	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the `TroutD` field in the **PC**Charge database (`PCCW.MDB`) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
`GetUpdateData`	*String*	*Used internally*

Method Name	Returned Value	Description - PccCheck Methods
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved from the .oux file can be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.
PccSysExists	Boolean	The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions.
Send	None	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the class will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has two optional parameters. The first parameter indicates whether the Send method will process transactions synchronously or asynchronously. **Note:** The object must defined to use events in order to allow asynchronous communication. **Valid Values:** True – process asynchronously **(Default)** False – process synchronously The second parameter indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send True, 3 **Note:** The other values that appear in the enumerated list are for internal use only-- do not attempt to use any values other than the ones listed above.
VerifyAmount	Boolean	The VerifyAmount method returns TRUE if the amount provided in the Amount property is in a valid format (DDDDDD.CC), or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
CashierName	String	Submit Cashier's login name.
Password	String	Submit Cashier's password.

PccCheck Events

Event Name	Description - PccCheck Events
Error	The Error event is fired any time an error occurs in the class. Once an Error event has fired, call GetErrorCode and GetErrorDesc to determine what kind of error has occurred. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
Finish	The Finish event will fire when the transaction has been completed. This means that **PC**Charge has processed the transaction successfully and has placed a file with the extension of .oux in the **PC**Charge directory. The name of the .oux file will be what was set in the User property of the transaction request. Call the GetResult method to determine whether or not the transaction was approved. A list of valid results can be found in the **Transaction Result Constants** section (see page 156).

Note: When doing asynchronous transactions in an event-driven programming model, it is important to place all result or error routines in either the Finish or Error events. Do not place any code that uses the .get methods after invoking the Send method.

PCCEBT Class

The PCCEBT class provides integrators with properties and methods used to submit EBT transactions to **PC**Charge. To use the PCCEBT class to integrate transaction processing, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to process transactions by using the PccSysExists method.

2. Assign the appropriate values to the properties required for the transaction to be performed and validate the values using the various .Verify methods. (The properties marked with a ° in the **PCCEBT Properties** table are the minimum required to process an EBT transaction.)

3. Call the Send method. (**Note:** When calling the Send method, it is recommended that "3" is passed as a parameter to activate the XML message format)

4. If programming asynchronously, wait for the Error or Finish event to occur.

5. If programming synchronously, code using the .Get methods may be placed immediately after the Send method

6. Call the various .Get methods to determine the outcome of the transaction. The most important information can be acquired by calling the GetResult and GetAuth methods. If an error occurs, call the GetErrorCode and GetErrorDesc methods to determine the nature of the error.

7. Call the DeleteUserFiles method to delete all files related to the transaction.

8. Destroy the object.

When processing EBT cards, a PINpad is required to allow the customer to enter their PIN. In addition, debit card information is always collected via a card swipe device, never via keyboard entry. Because of this, a card reader is also required. (Some EBT transactions can be manually entered).

When processing EBT card transactions, merchants have the option of allowing the customer to receive cash back on a transaction. For instance, the customer purchases $50 of products and wants $25 cash back, set the Amount to 50.00 and CashBack to 25.00. This will withdraw a total of $75 from the EBT card account, $50 for the products and $25 for cash to give to the customer.

This is a Multi Use Class.

Note: **PC**Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

PCCEBT Properties

Property Name	Data Type	Description - PCCEBT Properties
Action°	Long	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
Amount°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
AuthCode	String	For an EBT Post (Prior Auth Sale) or Force transaction: The Authorization code from the original voice authorization.
BDate	String	The Business / Batch Date. If populated, this value will be placed in the Business Date column of the transaction record in the **PC**Charge database (pccw.mdb). **Format:** MMDDYY
Billpay	String	Only valid for Visa debit and credit transactions. It is used to indicate the transaction is being ran for payment of a bill (ultilty, monthly gym dues, etc.) **Valid values:** 0 – Non-Bill payment transaction 1 – Bill payment transaction
Card°	String	The EBT card number that will be used when processing the transaction. **Max Length:** 20 characters. **Example:** 5424180279791765
CashBack	String	The amount of cash back that the customer will receive. This amount is in addition to value entered in Amount property. For example, if the total amount of the sale is $10 and the customer has requested $5 cash back, Amount should be set to $10 and CashBack should be set to $5. The CashBack property should be formatted the same the Amount property. **Max Length:** 9 characters. **Note:** Some debit processors do not support the cash back feature.
EBTType	String	Indicates what type of EBT transaction will be performed. **Valid Values:** F – Food stamp transaction; C – Cash benefits transaction
EnhancedTransFlag	*Boolean*	*Used internally*
ExpDate	String	The expiration date associated with the EBT card number that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208 Set this property if there is an expiration date associated with the EBT card.
ExtFile	*String*	*Used internally*
KeySerialNumber	String	If a Key Serial Number is returned from the PINpad, this property should be populated with that number. This value is only applicable for PINpads using DUKPT encryption. This value is sixteen or twenty characters long (depending on the processor's encryption). The **PC**Charge DevKit provides several tools for retrieving data from PINpads. If the **PC**Charge integration method chosen doesn't support these tools or the tools do not support the PINpad being used, a direct interface to the PINpad must be written by the integrator.
MACData	*String*	*Used internally*
MACState	*MACState*	*Used internally*
Manual°	Integer	Flag that indicates whether the transaction was swiped or manually entered. This property must be set to 1 (swiped) for swiped EBT transactions. If the transaction was swiped, the Track property must also be set. If performing a manually keyed EBT transaction, such as a Force or Voucher, set this property to 0 (manually entered).

Property Name	Data Type	Description - PCCEBT Properties
member	String	The cardholder's name. **Max Length:** 20 characters.
MerchantNumber°	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **EBT Card Setup** window of **PC**Charge. **Max Length:** 32 characters. This value can be alphanumeric.
Method	*TxnMethodType*	*Used internally*
Multi	Boolean	Flag that indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PC**Charge GUI. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid values:** 1 = TRUE, 0 = FALSE **Default value:** 0. See the section **Multi-trans Wait** for more information (see page 67). This Flag has no effect if processing will occur over IP or leased line.
OrigPurchData	*String*	*Not needed for EBT Transactions.*
Path°	String	The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Pin°	String	The encrypted PIN block that is retrieved from the PINpad. The PIN is provided to the processor for verification. **Length:** 16 characters. The **PC**Charge DevKit provides several tools for retrieving data from PINpads. If the **PC**Charge integration method chosen doesn't support these tools or the tools do not support the PINpad being used, a direct interface to the PINpad must be written by the integrator.
Processor°	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
Reference	*String*	*No longer needed*
SmartTermMsg	*String*	*Used internally*
SmartTermRequest	*Boolean*	*Used internally*
Ticket	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.
TimeOut	Long	The number of seconds after which a timeout error will be returned from **PC**Charge. The count will start when the Send method is called. It is highly recommended that integrators review the section **Timeouts** (see page 59). Setting the TimeOut value improperly could cause reconciliation issues and problems such as double-charging a customer's account. **Note:** The TimeOut properly is only applicable when programming in an asynchronous manner.
Track°	String	The track II data captured from the magnetic strip of the card. The track II data is required for swiped EBT transactions. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in.
TroutD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.

Property Name	Data Type	Description - PCCEBT Properties
User°	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).
VoucherNum	String	The voucher number for an EBT force transaction. The voucher is provided by the processor at the time of authorization and must be supplied to clear the voucher.
CashierName	String	Submit Cashier's login name.
Password	String	Submit Cashier's password.

° These fields are required to process a transaction.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

PCCEBT Methods

Method Name	Returned Value	Description - PCCEBT Methods
BalanceTotals	String	Current amount of EBT transactions
Cancel	None	Cancels current transaction
DeleteUserFiles	None	The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files, the Error event will be triggered (if asynchronous) and the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
GetApproved	Boolean	The GetApproved method returns TRUE if **PC**Charge returns "APPROVED" as the result of the transaction. Otherwise, FALSE will be returned.
GetAuth	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
GetAuxRespCode	String	When using the SC5000 PINpad, returns the ISO response code
GetEBTCashBalance	String	Returns the remaining balance on a Cash Benefits card. This value is not returned by all processing companies.
GetEBTFoodBalance	String	Returns the remaining balance on a Food Stamp card. This value is not returned by all processing companies.
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetRefNumber	String	Returns the reference number associated with the transaction. The reference number is used to help identify the transaction and is useful for the cardholder and merchant when doing research. This value is not returned with all transactions.
GetRespCode	String	Returns the response code that is provided by the processor. This value is not returned by all processing companies.

Method Name	Returned Value	Description - PCCEBT Methods
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
GetTI	String	Returns the Transaction Identifier that is returned from the processor. This value is not returned by all processing companies.
GetTICode	String	Returns the Transaction Indicator Code that is returned from the processor. This value is not returned by all processing companies.
GetTransNum	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
GetTroutD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
GetUpdateData	*String*	*Used internally*
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved from the .oux file can be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.
PccSysExists	Boolean	The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions.

Method Name	Returned Value	Description - PCCEBT Methods
Send	None	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the class will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has two optional parameters. The first parameter indicates whether the Send method will process transactions synchronously or asynchronously. **Note:** The object must defined to use events in order to allow asynchronous communication. **Valid Values:** True – process asynchronously **(Default)** False – process synchronously The second parameter indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send True, 3 **Note:** The other values that appear in the enumerated list are for internal use only-- do not attempt to use any values other than the ones listed above.
VerifyAmount	Boolean	The VerifyAmount method returns TRUE if the amount provided in the Amount property is in a valid format (DDDDDD.CC), or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).

PCCEBT Events

Event Name	Description - PCCEBT Events
Error	The Error event is fired any time an error occurs in the class. Once an Error event has fired, call GetErrorCode and GetErrorDesc to determine what kind of error has occurred. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
Finish	The Finish event will fire when the transaction has been completed. This means that **PC**Charge has processed the transaction successfully and has placed a file with the extension of .oux in the **PC**Charge directory. The name of the .oux file will be what was set in the User property of the transaction request. Call the GetResult method to determine whether or not the transaction was approved. A list of valid results can be found in the **Transaction Result Constants** section (see page 156).

Note: When doing asynchronous transactions in an event-driven programming model, it is important to place all result or error routines in either the Finish or Error events. Do not place any code that uses the .get methods after invoking the Send method.

PCCGiftCard Class

The PCCGiftCard class provides integrators with properties and methods used to submit gift card transactions to **PC**Charge. To use the PCCGiftCard class to integrate transaction processing, follow the procedure below:

1. Set the path to the **PC**Charge directory and check to see if **PC**Charge is running and available to process transactions by using the PccSysExists method.

2. Assign the appropriate values to the properties required for the transaction to be performed and validate the values using the various .Verify methods. (The properties marked with a ° in the **PCCGiftCard Properties** table are the minimum required to process a Gift Card Redemption / Sale transaction.)

3. Call the Send method. (**Note:** When calling the Send method, it is recommended that "3" is passed as a parameter to activate the XML message format)

4. If programming asynchronously, wait for the Error or Finish event to occur.

5. If programming synchronously, code using the .Get methods may be placed immediately after the Send method

6. Call the various .Get methods to determine the outcome of the transaction. The most important information can be acquired by calling the GetResult and GetAuth methods. If an error occurs, call the GetErrorCode and GetErrorDesc methods to determine the nature of the error.

7. Call the DeleteUserFiles method to delete all files related to the transaction.

8. Destroy the object.

Note: PC Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

VeriFone Stored Value API (GAPI)

The VeriFone Stored Value API (GAPI) is a proprietary specification that allows for stored value card processors to add themselves to **PC**Charge. Applications using GAPI can also integrate with **PC**Charge using the various integration methods. For more information on adding a stored value card processor to **PC**Charge, and how to obtain the VeriFone Stored Value API, please contact VeriFone sales at 1-800-725-9264.

PCCGiftCard Properties

Property Name	Data Type	Description - PCCGiftCard Properties
Action°	Long	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
Amount°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00". For Valuelink (VLNK) Balance Adjustment: **Format:** +/-DDDDD.CC.
Authcode	String	For **Void** transactions. For VTEC and VLNK, set to auth code of original transaction (the one to be voided). For GSAR and MELL, set to ref num of original transaction (the one to be voided). For BPS, set to retrieval reference number of original transaction (the one to be voided).
BDate	String	The Business / Batch Date. If populated, this value will be placed in the Business Date column of the transaction record in the **PC**Charge database (pccw.mdb). **Format:** MMDDYY
Card°	String	The gift card number that will be used when processing the transaction. **Max Length:** 20 characters.
CardSeqNum	String	For GSAR multi **Issuance**, sequence number of cards issued at time of transaction. **Example:** Ten cards are being issued. To send the fifth, set CardSeqNum to 5.
CashierID	String	VTEC and VLNK -- (optional) –numeric value that identifies the cashier performing the transaction.
DeactivateRefund	Boolean	Flag that indicates whether to provide the customer a refund when performing a VTEC Deactivate transaction. **Valid Values:** 1 – Provide refund 0 – Do not provide refund
EnhancedTransFlag	*Boolean*	*Used internally*
ExpDate	String	The expiration date associated with the gift card that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208 **Note:** Most gift cards do not have an expiration date.
ExtFile	*String*	*Used internally*
FORCE	Boolean	Set to true (1 = true, 0 = false) to process a transaction for which an approval code has already been issued -- only valid for a GSAR **Redemption** transaction or a single GSAR **Issuance/Add Value** transaction.
GiftPin	String	Only used for the processor **SVS**. To retrieve pin, call GetGfitPin upon activation. Used for only for virtual gift card transactions.
IndType°°	String	Indicates industry type (1 = retail, 2 = restaurant). VLNK -- (0 = retail, 1 = restaurant, 2 = e-commerce).
LanguageCode	*String*	*Used internally*
LastValidDate	String	The last year that will be considered a valid expiration date. **Length:** 2 digits. **Format:** YY **Example:** If LastValidDate is set to 05, then cards between 06 and 99 are considered to be 1906 to 1999, and cards between 00 and 05 are 2000 to 2005.
Loyalty	Boolean	VTEC loyalty transaction flag (0 = non-loyalty, 1 = loyalty).
MerchantNumber°	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Gift Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.

Property Name	Data Type	Description - PCCGiftCard Properties
Method	*TxnMethodType*	*Used internally*
Multi	Boolean	Flag that indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PC**Charge GUI. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid values:** 1 = TRUE, 0 = FALSE **Default value:** 0. See the section **Multi-trans Wait** for more information (see page 67). This Flag has no effect if processing will occur over IP or leased line.
OldCard	String	VTEC -- **Replace** transaction. Set to account number of old card. VLNK -- **Balance Merge** and **Balance Transfer** transactions. Set to account number of old card.
Partial	Boolean	For GSAR: Flag indicating whether the transaction is a partial redemption transaction.
Path°	String	The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ **Path Formats:** UNC, MS-DOS (8 Characters) and Long. 100 characters maximum. Must end with a "\".
Pin	*String*	*Used internally for Givex.*
Points	String	For GVEX **Points** transactions. Set to number of loyalty points for account.
PrintReceipts	String	The number of receipts that **PC**Charge should print for the transaction. This value will override the corresponding value in the **PC**Charge GUI. **PC**Charge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.
Processor°	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
PromoCode	String	Used for GVEX: A code defined by the merchant that affects the calculation from amount and units to points.
Refund	String	Flag that indicates whether to provide the customer a refund when performing a VTEC Deactivate transaction. **Valid Values:** 1 – Provide refund 0 – Do not provide refund
ShiftID	*String*	*Used internally*
TI	*String*	*No longer needed*
Ticket	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all gift processors support ticket numbers.
TimeOut	Long	The number of seconds after which a timeout error will be returned from **PC**Charge. The count will start when the Send method is called. It is highly recommended that integrators review the section **Timeouts** (see page 59). Setting the TimeOut value improperly could cause reconciliation issues and problems such as double-charging a customer's account. **Note:** The TimeOut properly is only applicable when programming in an asynchronous manner.
TIP	String	Used for VTEC and VLNK restaurant transactions.
TotalCardNum	String	For GSAR multi **Issuance**, total number of cards being issued at time of transaction.
Track	String	The track II data captured from the magnetic strip of the card. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in.

Property Name	Data Type	Description - PCCGiftCard Properties
TroutD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
VirtualGiftCardFlag	Boolean	Only used for the processor **SVS**. 0 - False, 1 - True – Only sent on an activation to determine if a pin should be returned.
User° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).
CashierName	String	Submit Cashier's login name.
Password	String	Submit Cashier's password.
SkuLoyalty	String	**GIVEX ONLY** – **Integration Only** Pass this tag in order to send product codes, quantities and amounts for loyalty transactions. **Format:** **Single Product Code:** <SKU_LOYALTY>Product Code,Amount,Quantity</SKU_LOYALTY> **Multi-Product Codes (Separated with a semi-colon):** <SKU_LOYALTY>Product Code,Amount,Quantity; Product Code,Amount,Quantity</SKU_LOYALTY>

° These properties are required to process a gift card redemption or sale transaction.

°° Required for VTEC gift card transactions

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

PCCGiftCard Methods

Method Name	Returned Value	Description - PCCGiftCard Methods
Cancel	None	Cancels transaction in progress
DeleteUserFiles	None	The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files, the Error event will be triggered (if asynchronous) and the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
GetActivationCount	String	Returns the number of activations in the current batch
GetActivationTotalAmount	String	Returns the total dollar amount of activations in the current batch
GetAddPointsCount	String	Returns the number of AddPoints Transactions in the current batch
GetAddPointsTotalAmount	String	Returns the total dollar amount of AddPoints transactions in the current batch
GetAddValueCount	String	Returns the number of AddValue transactions in the current batch
GetAddValueTotalAmount	String	Returns the total dollar amount of AddValue transactions in the current batch
GetAmountDue	String	Used in partial redemption transactions where only part of the amount was authorized. Returns the remainder amount that is owed to the merchant.

Method Name	Returned Value	Description - PCCGiftCard Methods
GetAuth	String	The GetAuth method returns the authorization number for approved transactions or the reason the transaction was declined (if the processor provides one). **For GVEX Balance transaction:** GetAuth will return the balance remaining on an account. **For all other GVEX transactions:** GetAuth will return the transaction's reference/error message. For VTEC, returns the Auth Code. **For a VTEC Batch function:** use this method to retrieve the number of sales done that day and the total amounts of sales in the following format <# of transaction>, <amount>.
GetAuthAmount	String	Used in partial redemption transactions where only part of the amount was authorized. Returns the actual authorized amount.
GetBalanceTransferCount	String	Returns the number of Balance Transfers in the current batch
GetBalanceTransferTotalAmount	String	Returns the total dollar amount of Balance Transfers in the current batch
GetCaptured	String	The GetCaptured method returns TRUE if **PC**Charge returns "CAPTURED" as the result of the transaction. Otherwise, FALSE will be returned. The GetCaptured method is used to determine if a transaction that will result in a monetary transfer (Sale, Credit, Post-Authorization, etc.) is approved or declined. A "CAPTURED" response indicates that the transaction has been approved.
GetCashBack	String	Used in redemption for remaining balance transactions where the transaction amount is so close to the balance of the card that the entire balance is authorized. Returns the remainder that is owed to the customer.
GetCreditCount	String	Returns the number of credits in the current batch
GetCreditTotalAmount	String	Returns the total dollar amount of credits in the current batch
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetExp	String	Returns the expiration date for processors who issue expiration dates in the response.
GetGiftCardBalance	String	Returns the gift card balance.
GetGiftCardType	String	Returns type of gift card represented by card property. Consult the section **DevKit Constants** for descriptions of values (see page 143).
GetGfitPin	String	Only used for the processor **SVS**. Returned on activation if the virtual gift card tag is set to "1".
GetMiscMessage	String	Returns the MiscMessage
GetMerchantNumber	String	Returns the merchant number that was specified in the MerchantNumber property.
GetPointsCount	String	Returns the number of points transactions in the current batch
GetPointsTotalAmount	String	Returns the total dollar amount of points transactions in the current batch
GetProcRespCode	String	The processor response code. Only returned by the processor **SVS**.
GetRefNumber	String	The GetRefNumber returns the Reference field from the .oux file. The Reference field is used for different purposes (depending on the gift card processor). **For GVEX Register transaction:** The first eleven digits of an account number will be returned. **For all VTEC transactions:** The account's remaining balance will be returned. **For a VTEC batch function:** use this method to retrieve the number of activations done that day and the total amounts of activations in the following format <# of transaction>, <amount>.>. For a BPS **Redemption** transaction, returns the retrieval reference number.
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.

Method Name	Returned Value	Description - PCCGiftCard Methods
GetRET	String	**For GVEX:** Returns the loyalty balance. **For VLNK:** Returns the trace number. **For a VTEC batch function: :** use this method to retrieve the number of Gift Transactions Voids performed that day. You can call GetVoidBalance to determine the total amount of the voids.
GetSaleCount	String	Returns the number of redemptions in the current batch
GetSaleTotalAmount	String	Returns the total dollar amount of redemptions in the current batch
GetTI	String	The GetTicket method returns the Ticket field from the .oux file. The Ticket field will return the ticket for all transactions except for a VTEC batch function. **For a VTEC batch function:** use this method to retrieve the number of gift card that has been de-activated that day and the total amounts of de-activations in the following format <# of transaction>, <amount>.>.
GetTicket	String	The GetTicket method returns the Ticket field from the .oux file. The Ticket field will return the ticket for all transactions except for a VTEC batch function. **For a VTEC batch function:** use this method to retrieve the number of gift card that has been de-activated that day and the total amounts of de-activations in the following format <# of transaction>, <amount>.>.
GetTIM	String	Returns the Time of the transaction. This value is not returned by all processing companies. For VTEC, returns the Amount Due.
GetTipCount	String	Returns the number of Tip transactions in the current batch
GetTipTotalAmount	String	Returns the total dollar amount of Tip transactions in the current batch
GetTransDateTime	String	Returns the transaction date and time when passed back by a processor.
GetTransNum	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
GetTroutD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
GetUpdateData	*String*	*Used internally*
GetVoidBalance	String	Returns the Void Balance
GetVoidCount	String	Returns the number of voids in the current batch
GetVoidTotalAmount	String	Returns the total dollar amount of Voids in the current batch
GetXMLResponse	String	The GetXMLResponse method is used to echo the text that is returned in the response file associated with the transaction. The response (.oux) file contains XML string data. The text that is retrieved from the .oux file can be used by integrators that wish to parse the results of the transaction themselves or for troubleshooting purposes. Refer to the section **File Method** (see page 481) for a description of the tags and values that are returned. **Note:** This method must be called prior to calling the DeleteUserFiles method.
PccSysExists	Boolean	The PccSysExists method is used to determine if **PC**Charge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PC**Charge directory and **PC**Charge is not available to process transactions. TRUE usually indicates that **PC**Charge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PC**Charge is ready to process transactions.

Method Name	Returned Value	Description - PCCGiftCard Methods
Send	None	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the class will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has two optional parameters. The first parameter indicates whether the Send method will process transactions synchronously or asynchronously. **Note:** The object must defined to use events in order to allow asynchronous communication. **Valid Values:** True – process asynchronously **(Default)** False – process synchronously The second parameter indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send True, 3 **Note:** The other values that appear in the enumerated list are for internal use only-- do not attempt to use any values other than the ones listed above.
ValidCardLength	Boolean	Returns TRUE for card of correct length
ValidDate	Boolean	The ValidDate method returns TRUE if the expiration date provided in the ExpDate property is valid, or FALSE if it is not.
ValidIssuer	Boolean	Returns TRUE for valid card issuer
VerifyAmount	Boolean	The VerifyAmount method returns TRUE if the amount provided in the Amount property is in a valid format (DDDDDD.CC), or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
VerifyAmount2	Boolean	The VerifyAmount2 method returns TRUE if the amount provided in the Amount property is in a valid format (+/-DDDDD.CC). or FALSE if it is not. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150). The difference between VerifyAmount and VerifyAmount2 is that VerifyAmount2 allows a + or – to be in the first position of the Amount property. This is needed for Balance Adjustment transactions.
VerifyExpDate	Boolean	The VerifyExpDate method returns TRUE if the expiration date provided in the ExpDate property is correct and in the right format, or FALSE if it is not. VerifyExpDate calls the ValidDate function to validate the expiration date. If FALSE is returned, check the error code to determine the reason for failure. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
VerifyGiftCard	Boolean	Returns TRUE if card is correctly formatted
VerifyMerchantNumber	Boolean	The VerifyMerchantNumber method returns TRUE if the merchant number that is passed to it is set up in **PC**Charge, otherwise, FALSE is returned. Specifically, this method checks for the merchant number in the file TID.PCC, which is located in the **PC**Charge directory. The Path property must be set before calling this Method.
VerifyProcessor	Boolean	Returns TRUE if processor is valid

Method Name	Returned Value	Description - PCCGiftCard Methods
GetPreAuthCount	String	Only for GAPI, this returns the total number of gift card pre-auth transactions processed that day.
GetPreAuthAmount	String	Only for GAPI, this returns the total amount of gift card pre-auth transactions processed that day.
GetPostAuthCount	String	Only for GAPI, this returns the total number of gift card post-auth transactions processed that day.
GetPostAuthAmount	String	Only for GAPI, this returns the total amount of gift card post-auth transactions processed that day.
GetIssuanceCount	String	Only for GAPI, this returns the total number of gift cards issued that day.
GetIssuanceTotalAmount	String	Only for GAPI, returns the total amount of the gift cards issued that day.
GetDeactivateCount	String	Only for GAPI, this returns how many gift cards where deactivated that day.
GetDeactivateTotalAmount	String	Only for GAPI, this returns the total amount of gift card deactivations that day.
GetBalanceAdjustCount	String	Only for GAPI, this returns the number of gift cards that were balance adjusted that day.
GetBalanceAdjustTotalAmount	String	Only for GAPI, this returns the total amount of balance adjustments on gift cards that day.
GetBalanceMergeCount	String	Only for GAPI, this returns the total number of the gift cards that were balance merged that day.
GetBalanceMergeTotalAmount	String	Only for GAPI, this returns the total amount of gift card balance merges that day.
GetReportLostStolenCount	String	Only for GAPI, returns the total reported stolen or lost gift cards that day.
GetReportLostStolenTotalAmount	String	Only for GAPI, returns the total amount of all stolen or reported lost gift cards that day.
GetCashoutTotalAmount	String	Only for GAPI, returns the total amount of all cashout transactions processed that day.
GetCashoutCount	String	Only for GAPI, returns the total number of the cashout transactions processed that day.
GetReactivateCount	String	Only for GAPI, returns the total number of gift cards that have been reactivated that day.
GetReactivateTotalAmount	String	Only for GAPI, the total amount of all gift cards that have been reactivated that day.

PCCGiftCard Events

Event Name	Description - PCCGiftCard Events
Error	The Error event is fired any time an error occurs in the class. Once an Error event has fired, call GetErrorCode and GetErrorDesc to determine what kind of error has occurred. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
Finish	The Finish event will fire when the transaction has been completed. This means that **PC**Charge has processed the transaction successfully and has placed a file with the extension of .oux in the **PC**Charge directory. The name of the .oux file will be what was set in the User property of the transaction request. Call the GetResult method to determine whether or not the transaction was approved. A list of valid results can be found in the **Transaction Result Constants** section (see page 156).

Note: When doing asynchronous transactions in an event-driven programming model, it is important to place all result or error routines in either the Finish or Error events. Do not place any code that uses the .get methods after invoking the Send method.

PccBatch Class

The PccBatch class is used to perform inquire and close operations on Host based merchant accounts. To perform a batch operation, follow the procedure below:

1. Assign the appropriate values to at least the minimum set of properties (marked with a ° in the **PccBatch Properties** table).

2. Call the BatchFunction method to initiate the operation. BatchFunction will respond when the operation is complete.

3. Grab values from the various return properties. Response contains a string that indicates how the operation was resolved.

PccBatch Properties

Property Name	Data Type	Description - PccBatch Properties
Action°	Integer	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
ActiveID*	Integer	Returns the Index number of the merchant number associated with the batch operation. For example, the first Merchant number that is set up in **PC**Charge is assigned the index of "1" .
AmexAmount*	String	Returns the total dollar amount of Amex transactions in current batch
AmexAuthSettlement	Boolean	Amex Auth Settlement flag
AmexCount*	String	Returns Number of Amex transactions in current batch
Balance*	String	Returns the batch balance (the sum of all transactions in the batch)
BatchCloseType	Byte	Flag that determines what type of batch close will occur. This flag only supported by FDMS Atlanta and Fifth-Third when using action code 30 or 31 **Valid values:** 1 – Standard End of Day Batch Close (Default) 2 – Shift Close 3 – Fifth-Third Terminal Based Batch Close of Debit, EBT, or Gift
BatchDate*	String	Returns the date batch was closed (not supported by all processors)
BatchNumber*	String	Returns the batch number for the current batch
CIC*	String	Returns the compliance indicator code
DebitAmount*	String	Returns the Debit Amount
DebitCount*	String	Returns the Debit Count
DebitRetAmount*	String	Returns the Debit Returns Amount
DebitRetCount*	String	Returns the Debit Returns Count
EBTAmount*	String	Returns the EBT Amount
EBTCount*	String	Returns the EBT Count
HangUpDelay	Integer	This value specifies if the modem should be hung up on subsequent calls to the BatchFunction method. The integer value passed in will cause a delay of that amount, in seconds, after the modem hangup command is called. This allows time for the modem to hangup before batch functions are allowed to call out. Setting the value to '0', the default, will prevent the modem from being hung up.
ItemCount*	String	Returns the total number of transactions in the current batch
MerchantNumber°	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Credit Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.

Property Name	Data Type	Description - PccBatch Properties
Path°	String	The path to the directory in which the **PCC**harge executable resides. This property must be set prior to calling the BatchFunction method. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Processor°	String	The code for the processing company that will be used when performing batch operations. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PCC**harge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
PurchaseAmount*	String	Returns the total dollar amount of all purchases (sales) in the batch.
PurchaseCount*	String	Returns the total number of all purchases (sales) in the batch.
Response*	String	Response string returned by processor.
ResultCode*	Byte	Returns the result of the batch operation. **Valid Values**: 2 – Batch Closed/Settled 6 – Batch Declined 8 – Batch Deferred
ReturnAmount*	String	Returns the total dollar amount of all returns (credits) in the batch.
ReturnCount*	String	Returns the total number of all returns (credits) in the batch.
Status*	String	Status of current batch.
SplitProcessor	String	Only used when settling the processor CITI for private label transactions. Set this property to the main credit card processor ID code being used.
Store*	String	Store number associated with merchant account.
Terminal*	String	Terminal ID associated with merchant account.
TotalsType*	Byte	Totals Type
VisaMCAmount*	String	Total amount of VISA/MasterCard transactions.
VisaMCCount*	String	Number of VISA/MasterCard transactions.
VoidAmount*	String	Returns the total dollar amount of all void transactions in the batch.
VoidCount*	String	Returns the total number of all void transactions in the batch.
CardType	String	Submit with a Batch Inquiry to receive summary results based on Card Type. Valid Values: MC – MasterCard VISA – Visa AMEX – American Express DISC – Discover Card DEBIT – Debit Card EBT – Electronic Benefits Card PL – Private Label JCB – JCB Card

° These properties are required to process a transaction.

* The processor returns these values.

PccBatch Methods

Method Name	Returned Value	Description - PccBatch Methods
AmexBatch	*Boolean*	*Used internally*
BancTecBatch	*Boolean*	*Used internally*
BatchFunction	Boolean	Call this method to start batch operation
BPASBatch	*Boolean*	*Used internally*
BPASTCPBatchCall	*None*	*Used internally*

Method Name	Returned Value	Description - PccBatch Methods
Cancel	None	Cancels the current batch operation
CCRDBatch	*Boolean*	*Used internally*
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during use of various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetEBTCount	String	Number of EBT Transactions
GetEBTAmount	String	Total Dollar Amount of EBT Transactions
GetAMEXCount	String	Number of AMEX Transactions
GetAMEXAmount	String	Total Dollar Amount of AMEX Transactions
GetTotalCount	String	Number of Transactions
GetTotalAmount	String	Total Dollar Amount of Transactions in the batch
GetVISACount	String	Number of VISA Transactions
GetVISAAmount	String	Total Dollar Amount of VISA Transactions
GetMCCount	String	Number of MC Transactions
GetMCAmount	String	Total Dollar Amount of MC Transactions
GetDISCCount	String	Number of DISC Transactions
GetDISCAmount	String	Total Dollar Amount of DISC Transactions
GetPLCount	String	Number of PL Transactions
GetPLAmount	String	Total Dollar Amount of PL Transactions
GetJCBCount	String	Number of JCB Transactions
GetJCBAmount	String	Total Dollar Amount of JCB Transactions

PccSettle Class

This class is used for settling batches with Terminal based systems. The batch must be settled regularly (at least once per day, ideally) to ensure that the funds from transactions are deposited before the authorizations expire.

PccSettle Properties

Property Name	Data Type	Description - PccSettle Properties
ActiveID*	Integer	Returns the Index number of the merchant number associated with the batch operation. For example, the first Merchant number that is set up in **PC**Charge is assigned the index of "1" .
AmexAuthSettlement	Boolean	Amex Auth Settlement Flag
Balance*	String	Returns the batch balance (the sum of all transactions in the batch)
Batches*	String	Returns the number of batches settled
BatchNumber*	String	Returns the batch number.
Error_Record*	Integer	Returns the index number of the transaction that cannot settle if an error occurs settling the batch
Indeterminate*	Boolean	Returns TRUE if the batch fails and returns an indeterminate response.
ItemCount*	String	Returns the number of items in current batch
MerchantNumber°	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Credit Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
MessageCount*	Integer	Returns the total number of messages sent to the processor
NoFile*	Boolean	Returns TRUE if **PC**Charge has no record of transactions to settle
NoSettle*	Boolean	Returns TRUE if the processor has no record of transactions to settle
Path°	String	The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the SettleBatch method. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Processor°	String	The code for the processing company that will be used when performing batch operations. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
record	*Integer*	*Used internally*
ResultCode	Byte	Returns the result of the batch operation. **Valid Values**: 2 – Batch Closed/Settled 6 – Batch Declined 8 – Batch Deferred
SettleNumber*	String	Returns the Settlement number from the processor
SettleResponse*	String	Returns the Response from the processor
Status*	String	Returns the Status of the batch
TCount*	String	Returns the Number of transactions in current batch

° These properties are required to process a transaction.

* The processor returns these values.

PccSettle Methods

Method Name	Returned Value	Description - PccSettle Methods
Cancel	None	Cancels the current batch operation
Clear	None	The Clear method will clear the values in all properties and methods.
ClearResponse	None	The Clear method will clear the values in all response related properties and methods.
CloseAmexSettleFile	*Boolean*	*Used internally*
CreateAmexSettleFile	*Boolean*	*Used internally*
CreateCESSettleFile	*None*	*Used internally*
CreateFDCNSettleFile	*None*	*Used internally*
CreateFDCSettleFile	*None*	*Used internally*
CreateGSARSettleFile	*None*	*Used internally*
CreateNBSettleFile	*None*	*Used internally*
CreateNDCSettleFile	*None*	*Used internally*
CreateNOVASettleFile	*None*	*Used internally*
CreateNPCSettleFile	*None*	*Used internally*
CreateNVUSSettleFile	*None*	*Used internally*
CreateTELMSettleFile	*None*	*Used internally*
CreateVISAKSettleFile	*None*	*Used internally*
DecryptSettleFile	*Boolean*	*Used internally*
EncryptAmexArchiveBatches	*Boolean*	*Used internally*
EncryptSettleFile	*Boolean*	*Used internally*
GetAmexBatchTotal	Boolean	Calculates the Amex Batch total. The ItemCount and Balance properties will contain the returned data. TRUE is returned if the operation is successful, FALSE otherwise.
GetBatchTotal	Boolean	Initiates an inquiry; returns TRUE if the inquiry was successful. Use this to determine how many batches are waiting to be settled. If multiple batches, a response will be returned for each batch. A loop must be initiated to read the response from each batch.
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during use of various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
SettleAmex	Boolean	Initiates AMEX direct settlement; returns TRUE if the settlement was successful.
SettleBatch	Boolean	Initiates a settlement; returns TRUE if the settlement was successful
UpdateSettleDB	*None*	*Used internally*
WriteAmexSettleRecord	*Boolean*	*Used internally*
WriteCESSettleDB	*None*	*Used internally*
WriteFDCNSettleDB	*None*	*Used internally*
WriteFDCSettleDB	*None*	*Used internally*
WriteGSARSettleDB	*None*	*Used internally*
WriteNBSettleDB	*None*	*Used internally*
WriteNDCSettleDB	*None*	*Used internally*
WriteNOVASettleDB	*None*	*Used internally*
WriteNPCSettleDB	*None*	*Used internally*
WriteNVUSSettleDB	*None*	*Used internally*
WriteTELMSettleDB	*None*	*Used internally*

Method Name	Returned Value	Description - PccSettle Methods
WriteVisaKSettleDB	*None*	*Used internally*

Note: In the event there are multiple batches waiting to be settled in one settlement, the integrated application will need to be designed to loop through the settlement response to retrieve the response for each batch.

PccSettleGift Class

This class is used for settling batches with Gift Card Processors. The batch must be settled regularly (at least once per day, ideally) to ensure that the funds from transactions are deposited before the authorizations expire.

PccSettleGift Properties

Property Name	Data Type	Description - PccSettleGift Properties
ActiveID*	Integer	Returns the Index number of the merchant number associated with the batch operation. For example, the first Merchant number that is set up in **PCC**harge is assigned the index of "1" .
Balance*	String	Returns the batch balance (the sum of all transactions in the batch)
Batches*	String	Returns the number of batches settled
Error_Record*	Integer	Returns the index number of the transaction that cannot settle if an error occurs settling the batch
ItemCount*	String	Returns the number of items in current batch
MerchantNumber°	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Credit Card Setup** window of **PCC**harge. **Max Length**: 32 characters. This value can be alphanumeric.
MessageCount*	Integer	Returns the total number of messages sent to the processor
NoFile*	Boolean	Returns TRUE if **PCC**harge has no record of transactions to settle
NoSettle*	Boolean	Returns TRUE if the processor has no record of transactions to settle
Path°	String	The path to the directory in which the **PCC**harge executable resides. This property must be set prior to calling the SettleBatch method. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
Processor°	String	The code for the processing company that will be used when performing batch operations. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PCC**harge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
record	*Integer*	*Used internally*
SettleNumber*	String	Returns the Settlement number from the processor
SettleResponse*	String	Returns the Response from the processor
Status*	String	Returns the Status of the batch
TCount*	Integer	Returns the Number of transactions in current batch

° These properties are required to process a transaction.

* The processor returns these values.

PccSettleGift Methods

Method Name	Returned Value	Description - PccSettleGift Methods
Cancel	None	Cancels the current batch operation
Clear	None	The Clear method will clear the values in all properties and methods.
GetBatchTotal	Boolean	Initiates an inquiry; returns TRUE if the inquiry was successful

Method Name	Returned Value	Description - PccSettleGift Methods
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during use of various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetGiftTIDSeqNumber	Integer	This method returns the index of the credit card merchant number that relates to a gift card merchant number. Two parameters, the gift card merchant number and the gift card processor code must be passed when calling this method.
SettleBatch	Boolean	Initiates a settlement; returns TRUE if the settlement was successful

PccPinPad Class

The PccPinPad class allows integration to PINpad devices connected to the serial port.

This is a Multi Use class.

The PccPinPad class provides the functionality to Initialize a PINpad and to retrieve the PIN from the cardholder. The PINpad must be initialized once prior to retrieving the PIN. If the PINpad's power is cycled, it must be re-initialized. If the object is destroyed and then instantiated, the PINpad must be re-initialized.

PccPinPad Properties

Property Name	Data Type	Description - PccPinPad Properties
AccType	GPSAccType	The customer's bank account type used when processing the transaction. 1 or ENUM_ACC_CHEQ - Chequing 2 or ENUM_ACC_SAV – Savings **Note:** When populating this property to build the initial Interac request string, the customer's bank account type will not be known by the merchant (the customer will enter it once prompted). This value must be hard-coded. It is suggested to hard-code this value to "1" if most of the merchant's customers use their checking accounts when purchasing products or "2" if most of the merchant's customers use their saving account when purchasing products. **Note:** A new MAC value must be requested from the PINpad if the account chosen by the merchant differs from the account chosen by the customer. Set this property to the account chosen by the customer, call the BuildInteracRequest method again and use the RequestMAC method in the PinSC5000 class to request a new MAC value from the PINpad.
Action	String	Set this property to the type of transaction being processed. This should be set to the same transaction type specified in the TransNameID property. 0 – Purchase (default) 4 – Refund
AllDataReceived	Variant	Indicates whether all data has been received
Amount**	String	This amount displayed to customer on the PINpad for approval The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
Argements	*String*	*N/A*
Baud*	Variant	Baud rate **Example:** "1200"
Card**	String	Debit card number
Command	*String*	*N/A*
DataBits*	String	Databits **Example:** "7"
DataWaiting	*Variant*	*N/A*
Device*	Variant	The type of PINpad to be used. Types of PINpads are listed below. **Example:** "1"

Property Name	Data Type	Description - PccPinPad Properties
ExpDate	*String*	*N/A*
Gratuity	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. This is the Gratuity Amount of the transaction returned by the PINpad.
IdleMessage	String	The default message for the PINpad that is connected to the machine that PccPinPad will be communicating with. For example, if this is set to "Welcome", "Welcome" will appear on the screen at times of inactivity.
IncomingData	*String*	*N/A*
KeyManagement*	PinPadManagement	Type of encryption method to be used.
KeySerialNumber	String	The Key Serial Number of the transaction
LangCode	GPSLangCode	Language Code used when communicating with and displaying messages on the PINpad. Verifone SC5000 only **Valid Values:** 0 – English (**default**) 1 – French
LastChar	*Variant*	*N/A*
MACData	String	MAC data from the PINpad. Verifone SC5000 only.
MasterKey	Integer	The Master Key for Master Session encryption. Not all processors who do Master Session encryption will have a Master Key. Set MasterKey to "0" if no Master Key is present.
member	String	Name of cardholder
OLEPinState	Byte	This value must be set to "1" in order to communicate with the PINpad properly
Parity*	String	Parity setting for PINpad. ("E" for even, "O" for odd, "N" for none)
Pin	String	The encrypted PIN entered by user.
Port*	String	Number of the COM port to be used (**Example:** "1" used for COM port 1)
ReceivingData	Variant	Indicates whether device is in process of receiving data
Response	String	Response from processor
SerialNumber	String	Chip Serial Number from the PINpad.
StartTime	*Long*	*N/A*
State	PinPadState	States of PINpads are listed below
TermID	String	Set this property to the Chip Serial Number of the PINpad. Use the GetSerialBlock method in the PinSC5000 class to acquire the Chip Serial Number of the PINpad. Verifone SC5000 only.
TimeOut	Long	Length of time after a request for PIN before **PC**Charge times out. It is highly recommended that integrators review the section **Timeouts** (see page 59). **Note:** The TimeOut properly is only applicable when programming in an asynchronous manner.
TipType	Integer	Determines if the PINpad will prompt the customer to enter a tip amount and also determines if the PINpad will display a "suggested" tip amount prior to prompting for the tip amount. **Valid Values:** 0 – The customer is not prompted to enter a tip amount (Default) 1 – The customer is prompted to enter a tip amount. 2 through 99 – The PINpad will calculate a "suggested" tip amount and display it on the PINpad prior to prompting the customer for the tip amount. The integer value passed in represents the percentage that will be used to calculate the tip. For example, 20 would calculate a 20% tip amount. The customer would see this "suggested" tip amount and would then be prompted to key in the actual tip amount.
TrackData	String	Raw swiped track data from card
WorkingKey	String	Property for future use

* These properties must be set before Initialize can be called.
** These properties must be set before GetPin can be called.

PccPinPad Methods

Method Name	Returned Value	Description - PccPinPad Methods
AtallaAsciiString	*String*	*No longer supported*
AtallaComm	*Boolean*	*No longer supported*
AtallaIO	*Boolean*	*No longer supported*
Cancel	Boolean	Returns TRUE if transaction was canceled
ClearData	Boolean	Clears all data from device
ClearPort	Boolean	Clears PINpad buffer
ClosePort	Boolean	Releases COM port to which PINpad is connected
CreatPinPadFile	*None*	*Used internally*
GetPin	Boolean	The GetPin method initiates the PIN entry prompt on the PINpad. If successful, TRUE will be returned. Once TRUE is returned, the PIN will be returned in the .Pin property and the Key Serial Number will be returned in the .KeySerialNumber property.
Initialize	Boolean	Initializes PINpad
LoadKey	*None*	*N/A*
MAC	Boolean	Perform MAC with the PINpad
OpenPort	Boolean	Captures COM port to which PINpad is connected
ParseData	Boolean	Parse swipe data
ReceiveData	*Boolean*	*N/A*
Send	*None*	*Not used. Use GetPin to retrieve PIN from PINpad*

The following is a list of available PINpad types supported.

PinPadDevice Properties

Property Name	Data Type	Description - PinPadDevice Properties
ppdIVI_CheckMate_eNTouch1000	Enum	Value = 10
ppdIVICheckmate_2100	Enum	Value = 5
ppdIVIeNCrypt_BMON	Enum	Value = 7
ppdIVIPOSPad	Enum	Value = 6
ppdIVISentinel	Enum	Value = 3
PPDNone	Enum	Value = 0
ppdPenWare_3100	Enum	Value = 8
ppdVerifone_101	Enum	Value = 1
PPDVerifone_2000	Enum	Value = 2
ppdVerifone_Everest	Enum	Value = 4
ppdVerifone_SC500_C	Enum	Value = 9
ppdVerifone_3730	Enum	Value = 12

The following is a list of all available PINpad states.

PinPadState Properties

Property Name	Data Type	Description - PinPadState Properties
PPSCancel	Enum	Value = 2
PPSIdle	Enum	Value = 0
PPSProcess	Enum	Value = 1

PccSC550 Class

PccSC550 Class Properties

Property Name	Data Type	Description - PccSC550 Class Properties
AccType°	GPSAccType	The customer's bank account type used when processing the transaction. 1 or ENUM_ACC_CHEQ - Chequing 2 or ENUM_ACC_SAV – Savings **Note:** When populating this property to build the initial Interac request string, the customer's bank account type will not be known by the merchant (the customer will enter it once prompted). This value must be hard-coded. It is suggested to hard-code this value to "1" if most of the merchant's customers use their checking accounts when purchasing products or "2" if most of the merchant's customers use their saving account when purchasing products. **Note:** A new MAC value must be requested from the PINpad if the account chosen by the merchant differs from the account chosen by the customer. Use the RequireReMac property in the PinSC5000 class to determine if ReMACing must occur.
Action	*String*	*Used internally*
Amount°	String	Set this property to the Amount of the transaction to be processed. This property should not contain any decimals or commas. **Example:** to specify a $1.00 transaction, set this property to "100"
Baud	Variant	Baud Rate used to communicate with the PINpad. **Default Value:** "9600"
ChipSerialNum	String	This property will contain the Chip Serial Number of the PINpad. This field is populated by running the GetChipSerialNum method. The Chip Serial Number is passed as a parameter to the Debit classes' GetPOSSequenceNumber method.
DataBits	Variant	DataBits used to communicate with the PINpad. **Default Value:** "8"
DisplayAmount°	String	Set this property to the amount of the transaction. The amount specified is displayed to the customer for confirmation on the PINpad. This amount should include the decimal point and trailing zeros, if applicable. **Example:** to specify a $1.00 transaction, set this property to "1.00"
LanguageCode	GPSLangCode	Language Code used when communicating with and displaying messages on the PINpad. **Valid Values:** 0 – English (**default**) 1 – French
MACBlock°°	String	After the ParseResponseData method has been called, this property will contain the MAC block information that was returned from the PINpad.
Parity	Variant	Parity used to communicate with the PINpad. **Valid Values:** E – even O – odd N – None (default)
PINBlock°°	String	After the ParseResponseData method has been called, this property will contain the customer's encrypted PIN that was returned from the PINpad.
Port	String	Port Number to be used to communicate with PINpad. **Default Value:** "COM1"
SC550Form	*Object*	*Used internally*
SequenceNum°	String	The POS Sequence number. Set this property to the value that was returned by **PC**Charge when calling the GetPOSSequenceNumber method in the Debit.OCX control.
TermID°	String	Set this property to the Chip Serial Number of the PINpad. Use the GetSerialBlock method in the PinSC550 class to acquire the Chip Serial Number of the PINpad.

Property Name	Data Type	Description - PccSC550 Class Properties
TipAmount°°	String	After the ParseResponseData method has been called, this property will contain the optional customer-specified tip amount. If this property is populated with a value greater than 0, a ReMAC must occur.
TipType°	Integer	Determines if the PINpad will prompt the customer to enter a tip amount and also determines if the PINpad will display a "suggested" tip amount prior to prompting for the tip amount. **Valid Values:** 0 – The customer is not prompted to enter a tip amount (Default) 1 – The customer is prompted to enter a tip amount. 2 through 99 – The PINpad will calculate a "suggested" tip amount and display it on the PINpad prior to prompting the customer for the tip amount. The integer value passed in represents the percentage that will be used to calculate the tip. For example, 20 would calculate a 20% tip amount. The customer would see this "suggested" tip amount and would then be prompted to key in the actual tip amount.
TrackII°	String	Set this property to the track II data from the magnetic stripe of the card. For example: ;1234123412341234=08121234123412340001?
TransCode°	GPSTransCode	Set this property to the type of transaction being processed. This should be set to the same transaction type specified in the TransNameID property. 0 or ENUM_TCODE_PURCH_NORM – Purchase (default) 4 or ENUM_TCODE_REFUND – Refund
TransNameID°	GPSTransID	Set this property to the type of transaction being processed. The property indicates to the PINpad to displays the type of transaction being processed to the customer. This should be set to the same transaction type specified in the TransCode property. **Valid Values:** 0 or ENUM_TID_PURCH – Displays "PURCHASE" (default) 1 or ENUM_TID_REFUND – Displays "REFUND"

° These properties must be set prior to calling the BuildInteracRequest method.
°° These properties will be set after the ParseResponseData method completes successfully.

PccSC550 Class Methods

Property Name	Returned Value	Description - PccSC550 Class Methods
BuildInteracReqString	String	Builds and returns the Interac request string that will be sent to the PINpad. The properties in the **SC550.clsInteracReq Class Properties** table that are marked with a ° must be set prior to calling this method.
GetChipSerialNum	Boolean	Obtains the Chip Serial Number from the PINpad. This method populates the ChipSerialNum property with the PINpad's Chip Serial Number.
GetPin	Boolean	Gets the PIN from the PINpad. This will prompt the user to enter his/her PIN. GetPin will then return the encrypted PIN. GetPin will return nothing if a timeout or cancel occurs.
Initialize	Boolean	Initializes the PINpad. Returns TRUE if the initialization was successful, FALSE if not. Initialize has an optional parameter than can be passed in that will allow checking of the com port.
KeyChangeRequest	None	Requests a key change from the PINpad.

PccBin Class

This functions in this class will return information that indicates if a credit card is a Commercial Card, and what type of Commercial Card it is. See the section **Commercial Card Transactions** (see page 93) for more information.

This is a Multi Use class.

PccBin Properties

Property Name	Data Type	Description - PccBin Properties
Canceled	*Boolean*	*Used internally*
CommercialCard	Boolean	To test if the card is a Level II card (Business, Purchase, Corporate, or Fleet), pass the card number to this function. The function will return TRUE if the card's BIN range appears in the Bin.mdb database that resides in the **PC**Charge directory. FALSE will be returned if it doesn't. If TRUE is returned, check CommercialCardType to determine what type of Level II card it is.
CommercialCardType	String	This property indicates the card type. After calling the CommercialCard function, the CommercialCardType property will be populated with one of the following values: B - Business P,L,G - Purchase C - Corporate F - Fleet N - not a Level II card If processing a commercial card, the CmrclCardFlag property in the PccCharge class should be populated with this value prior to submitting the transaction.
Index	*Integer*	*No longer supported*
MC	*Collection*	*No longer supported*
Other	*Collection*	*No longer supported*
VS	*Collection*	*No longer supported*

PccBin Methods

Method Name	Returned Value	Description - PccBin Methods
Load	*Boolean*	*No longer supported*
Save	*Boolean*	*No longer supported*
Show	*Boolean*	*No longer supported*

Reporting

The PccCharge class may be used by integrators to submit report requests. A report request can have **PC**Charge print a report to it's default report printer or have **PC**Charge generate a file containing the report output. If generating a file, the **PC**Charge reporting interface supports three different file types:

1. Portable Document Format (.pdf)
2. Rich Text Files (.rtf)
3. Standard Text files (.txt)

Note: The reporting interface cannot be configured to send reports directly to the screen.

The following outlines the properties used for submitting report requests to PCCharge with the PccCharge class. The properties in PccCharge that are not documented below should be left blank when submitting report requests.

Property	Data Type	Description - PccCharge Class Reporting Properties
Action°	Long	The action code that identifies what type of report will be requested. **Valid Values:** 81-84. **Example:** If running a credit card detail report, set the action code to "81". Consult the section **DevKit Constants** for a list of valid values (see page 143).
Card	String	**User name filter**. If a valid user name is set in the Card property, the report will be filtered by that user name. The report returned will consist of only those transactions processed by the user name specified. **Example:** "User1". If this property is left blank, the report will show transactions processed by all users.
Manual	Long	**Result filter**. Use this filter to create a report consisting of only those transactions with the result specified. **Valid Values:** 0 = all (default), 1 = approved, 2 = declined **Example:** 1
member	String	**Ending Date/Time filter**. Specifies the end date and end time of the report. **Format**: Date: MM/DD/YY Time: HH:MM:SS PM. When used in conjunction with Street; will create a report consisting of only those transactions processed between the start and end date/time specified (inclusive). If an end date is not specified, today's date is assumed. If an end time is not specified, 11:59:59 PM is assumed. The end date can be passed without the end time. However, the end time cannot be passed without the end date. **Examples:** "07/06/05 06:00:00 PM" or "07/06/05"
MerchantNumber	String	**Merchant Number filter**. Set this property to filter the report by the merchant number specified. Setting this property will generate a report consisting of only those transactions processed via the merchant number specified. To generate a report that includes all merchant numbers in **PC**Charge, set this property to "ALL" or leave blank. **Example:** "99999999911"
Path°	String	The path to the directory in which the **PC**Charge executable resides. This property must be set prior to calling the Send, PccSysExists, and other methods that require accessing the **PC**Charge directory. **Example:** C:\Program Files\PCCW\ or C:\Program Files\Active-Charge\ **Path Formats:** UNC, MS-DOS(8 Characters) and Long. 100 characters maximum. Must end with a "\".
PeriodicPayment	String	**Report Output setting**. Determines if the report will be printed by **PC**Charge or written to a file. **Valid Values:** "0" = print to default printer specified in **PC**Charge (default). "1" = print to file using filename specified in TransID and path specified in TRACK.

Property	Data Type	Description - PccCharge Class Reporting Properties
Street	String	**Starting Date/Time Filter** (Optional) Specifies the start date and start time of the report. **Format**: Date: MM/DD/YY Time: HH:MM:SS PM. Use to create a report consisting of only those transactions processed on or after the date specified. If a start date is not specified, today's date is assumed. If a start time is not specified, 12:00:00 AM is assumed. The start date can be passed without the start time. However, the start time cannot be passed without the start date. **Examples:** "03/04/05 09:00:00 AM" or "03/04/05"
TimeOut	Long	The number of seconds after which a timeout error will be returned from **PCC**harge. The count will start when the Send method is called. It is highly recommended that integrators review the section **Timeouts** (see page 59). **Note:** The TimeOut properly is only applicable when programming in an asynchronous manner.
Track	String	**Destination Directory for Report File**. Specifies the destination directory where the report file will be generated by **PCC**harge (if PeriodicPayment is set to "1"). **Example:** "C:\My Documents\PCCReports\" **Path Formats:** UNC, MS-DOS(8 Characters) and Long. **Max Length:** 40 characters (if the Destination Directory is longer than 40 characters, use CustCode for the additional characters. Must end with a "\" unless the directory name will be continued in the CustCode property. **Note:** If running in a Client/Server environment, this property is the path from the server running **PCC**harge, not the client. For example, if a client submitted a report request that specified "C:\" as the destination directory, the report would be written to the local hard drive of the server running **PCC**harge, not to the client's hard drive.
CustCode	String	**Destination Directory for Report File (continued)**. Continuation of the destination directory (if the directory name is greater than 40 characters). **Max Length:** 25 characters. Must end with a "\"
TRANSID	String	**Report File name/Report File Type**. Specifies the filename and extension of the report file generated by **PCC**harge (if PeriodicPayment is set to "1"). Also determines what file type will be used when **PCC**harge generates the report. To specify the file type, set the extension to one of the following: **.pdf** – Create the report file in the Portable Document Format. **Ex.** Report.pdf **.rtf** – Create the report file in Rich Text. **Ex.** Report.rtf **.txt** – Create a report file in flat text. **Ex.** Report.txt **Default: .txt** (If an extension other than the ones listed is passed, the report will be returned as flat text and a .txt extension will be added to the filename)
User°	String	Sets the **PCC**harge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62).

° These properties are required to submit a report request.

The following outlines the methods used to process report requests. The methods in PccCharge that are not documented below will not be used when processing report requests.

Method	Returned Value	Description - PccCharge Class Reporting Methods
Cancel	None	The Cancel method attempts to cancel the transaction in progress. Calling the Cancel method does not guarantee that the transaction will be canceled; it simply attempts to cancel the transaction.

Method	Returned Value	Description - PccCharge Class Reporting Methods
DeleteUserFiles	None	The DeleteUserFiles method attempts to delete all request and response files associated with the transaction. It will delete the files based on the value set in the User property. The DeleteUserFiles method should be called after the results have been retrieved from the transaction. If an error occurs while attempting to delete the files, the Error event will be triggered (if asynchronous) and the GetErrorDesc method will give a brief description of the error. Consult the section **System Error Codes and Descriptions** for a list of valid error codes and descriptions that will be returned (see page 150).
GetAuth	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
GetErrorCode	Long	The GetErrorCode method returns an error code if an error was encountered during the use of various methods such as the Send, Cancel, DeleteUserFiles, and PccSysExists. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetErrorDesc	String	The GetErrorDesc method returns a string representation of the error that was encountered during the use of the various methods. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
PccSysExists	Boolean	The PccSysExists method is used to determine if **PCC**harge is available to process transactions. If PccSysExists returns TRUE, the file SYS.PCC exists in the **PCC**harge directory and **PCC**harge is not available to process transactions. TRUE usually indicates that **PCC**harge is either not running, is performing a batch or database function, or is in an error state. The GetErrorCode and GetErrorDesc methods will provide information as to why the file exists. Consult the section **System Error Codes and Descriptions** for a list of valid error codes that will be returned (see page 150). If PccSysExists returns FALSE, then **PCC**harge is ready to process transactions.

Method	Returned Value	Description - PccCharge Class Reporting Methods
Send	Integer	The Send method creates a text file containing the transaction request and places the file in the **PC**Charge directory. The Send method will check the action code specified and perform the transaction type indicated. If an error occurs while Send executes, the class will set the error code and description, raise the Error event, and terminate processing. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). The Send method has two optional parameters. The first parameter indicates whether the Send method will process transactions synchronously or asynchronously. **Note:** The object must defined to use events in order to allow asynchronous communication. **Valid Values:** True – process asynchronously **(Default)** False – process synchronously The second parameter indicates what message format will be used for the request and response files. This parameter may be specified by using a numerical value (or an enumerated value if the programming language being used supports enumerated values). **IMPORTANT NOTE**: It is highly recommended that the XML message format parameter is set when calling the Send method. All DevKit documentation assumes that the XML message format parameter has been set. (The legacy INP message format is selected by default for backwards compatibility reasons.) **Valid values:** 3 (TTYPE_XML) – XML message format – (**RECOMMENDED**) **Example:** Send True, 3 **Note:** The other values that appear in the enumerated list are for internal use only-- do not attempt to use any values other than the ones listed above.
VerifyMerchantNumber	Boolean	The VerifyMerchantNumber method returns TRUE if the merchant number that is passed to it is set up in **PC**Charge, otherwise, FALSE is returned. Specifically, this method checks for the merchant number in the file TID.PCC, which is located in the **PC**Charge directory. The Path property must be set before calling this Method.

PccCharge Events

Event	Description - PccCharge Class Events
Error	The Error event is fired any time an error occurs in the class. Once an Error event has fired, call GetErrorCode and GetErrorDesc to determine what kind of error has occurred. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150).
Finish	The Finish event will fire when the transaction has been completed. This means that **PC**Charge has processed the transaction successfully and has placed a file with the extension of .oux in the **PC**Charge directory. The name of the .oux file will be what was set in the User property of the transaction request. Call the GetResult method to determine whether or not the transaction was approved. A list of valid results can be found in the **DevKit Constants** section (see page 143).

Note: When doing asynchronous transactions in an event-driven programming model, it is important to place all result or error routines in either the Finish or Error events. Do not place any code that uses the .get methods after invoking the Send method.

Utility Related Classes

OS

The OS object contains information about the operating system on the local machine.

This is a Multi Use class.

OS Properties

Property Name	Data Type	Description - OS Properties
BuildNumber	Long	Returns the build number of the operating system on the machine that **PC**Charge is installed
CSDVersion	String	Returns the CSD Version (Service Pack level) of the operating system on the on machine that **PC**Charge is installed
MajorVersion	Long	Returns the Major version of the operating system on the machine that **PC**Charge is installed
MinorVersion	Long	Returns the Minor version of the operating system on the machine that **PC**Charge is installed
OSVersion	EOS_VERSION	Returns the version of the operating system on the machine that **PC**Charge is installed
Platform	String	Returns the Platform of the operating system on the machine that **PC**Charge is installed
PlatformID	Long	Returns the Platform ID of the operating system on the machine that **PC**Charge is installed
Version	String	Returns the full version information of the operating system on the machine that **PC**Charge is installed
Win2000	Boolean	Indicates whether operating system of the local machine is Windows 2000.
Win2003	Boolean	Indicates whether operating system of the local machine is Windows 2003.
Win95	Boolean	Indicates whether operating system of the local machine is Windows 95.
Win98	Boolean	Indicates whether operating system of the local machine is Windows 98.
WinME	Boolean	Indicates whether operating system of the local machine is Windows ME or was derived from Windows ME
WinNT	Boolean	Indicates whether operating system of the local machine is Windows NT or was derived from Windows NT (**Example:** Windows 2000)
WinXP	Boolean	Indicates whether operating system of the local machine is Windows XP or was derived from Windows XP

PccActiveCharge

This class contains information about the current instance of **PC**Charge.

This is a Multi Use class.

PccActiveCharge Properties

Property Name	Data Type	Description - PccActiveCharge Properties
Check	Collection	Contains merchant check processing information
CompanyName	Variant	Merchant company information
Credit	Collection	Contains merchant credit card processing information
CSZ	Variant	This value is read-only
Debit	Collection	Contains merchant debit processing information
EBT	Collection	Contains merchant EBT processing information
EmailHelp	Variant	Private label information specified in sys.cfg
Fax	Variant	Private label information specified in sys.cfg
GiftCard	Collection	Contains merchant gift card processing information
HndUI	Long	This value is read-only. Contains the handle for the main UI window. If null, the UI is not loaded. In that case, set ShowAcUI to "1", then re-read HndUI.
License	Variant	Private label information specified in sys.cfg
PccModem	PccModem	Merchant modem information
PccUtilities	PccUtilities	Database Utilities
Phone	Variant	Private label information specified in sys.cfg
ProgramName	Variant	Private label information specified in sys.cfg
ShowAcUI	Boolean	This value is write-only. If set to "1", the **PC**Charge COM form becomes visible.
Street	Variant	Private label information specified in sys.cfg
Version	String	Private label information specified in sys.cfg
Web	Variant	Private label information specified in sys.cfg

PccActiveCharge Methods

Method Name	Returned Value	Description - PccActiveCharge Methods
GetActiveIndex	Integer	Returns index of active TID
GetIndex	Integer	Returns index of specified arguments
GetMaxMerchants	Integer	Returns number of TIDs in **PC**Charge
ReInitialize	Boolean	Re-initializes **PC**Charge without shutting the program down.
ShutDown	Boolean	Shuts down current instance of **PC**Charge
StartUp**	Boolean	Starts up and initializes application

** Be sure to use this method to ensure that all variables and data structures are properly initialized.

PccDBBackup

The PccDBBackup object allows access to **PC**Charge's transaction archive capabilities. This feature is briefly described below, however, a detailed explanation of this feature can be found in the **PC**Charge Pro or Payment Server User Manuals.

PCCharge offers transaction archive capabilities via integration. Archived transactions are moved from **PC**Charge's working database (pccw.mdb) into the archive database (pccwhist.mdb). The following archive integration methods are supported:

- OCX Method
- OLE/COM Method
- File Method
- TCP Method

The action code ZA specifies a transaction archive request. Consult the section **DevKit Constants** for descriptions of values (see page 143), and consult the section **File Method** for descriptions of transaction fields (see page 481).

This is a Multi Use class.

PccDBBackup Properties

Property Name	Data Type	Description - PccDBBackup Properties
ArchivePre5_6	Boolean	Flag to indicate whether pre-PCCharge version 5.6 transactions should be archived. TRUE – Archive Pre 5.6 Transactions FALSE – Do not archive Pre 5.6 Transactions
Cancel	Boolean	Set to TRUE to cancel the archive operation in progress.
Enabled	Boolean	Enable or disable current configuration **Valid values:** 1 – Enable 0 – Disable
KeepDays	Integer	Transaction archive preservation range. All transactions within the past number of "keep days" will remain in the pccw.mdb database following a transaction archive command.
Path	String	Specify path for saved output files (**Example:** backed up transaction database). Must end with a backslash "\".
SizeLimit	Long	Transaction archive size limit for GUI archive prompting and validation. Specified in megabytes.

PccDBBackup Methods

Method Name	Returned Value	Description - PccDBBackup Methods
Backup(GUIPrompt)	Boolean	Perform transaction archive. Returns TRUE if successful, FALSE otherwise. Set GUIPrompt to FALSE during a transaction archive request to suppress a GUI error message if the request fails.
LimitExceeded	Boolean	Transaction database exceeds the configured archive limit.
Load	Boolean	Load the saved archive configuration. Returns TRUE if successful, FALSE otherwise.
PCCWDatabaseSize	Long	Current transaction database size in bytes.
Save	Boolean	Save the current archive configuration. Returns TRUE if successful, FALSE otherwise.

PccUtilities

Provides various **PC**Charge utilities to the integrator.

PccUtilitiesMethods

Method Name	Returned Value	Description - PccUtilities Methods
`BackUp`	Boolean	Backs up the merchant configuration files and **PC**Charge database to a ZIP file. Requires the path to the **PC**Charge directory as a parameter.
`Compact`	Boolean	Compacts the database and the settlement file. An optional parameter, the index of the merchant number, can be passed in. If this value is passed, the compact function will only compact the transactions related to that index.
`Repair`	Boolean	Calls the database engine repair function. Repairs the **PC**Charge database.
`Restore`	Boolean	Restores the merchant configuration files and **PC**Charge database into the **PC**Charge directory. Requires the path to the directory that contains the ZIP file as a parameter

Setup Related Classes

PccAddv

This class contains address verification system settings for the current instance of **PC**Charge. This is a Multi Use class.

PccAddv Properties

Property Name	Data Type	Description - PccAddv Properties
Add5Zip	Boolean	Indicates whether to capture transaction when address and 5-digit zip code match.
Add9Zip	Boolean	Indicates whether to capture transaction when address and 9-digit zip code match.
AddNoZip	Boolean	Indicates whether to capture transaction when address matches but zip code does not.
Canceled	*Boolean*	*Used internally*
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
NoAdd5Zip	Boolean	Indicates whether to capture transaction when 5-digit zip code matches but address does not.
NoAdd9Zip	Boolean	Indicates whether to capture transaction when 9-digit zip code matches but address does not.
NoMatch	Boolean	Indicates whether to capture transaction when address and zip code do not match.
NotAvailable	Boolean	Indicates whether to capture transaction when address information is not available.
Retry	Boolean	Indicates whether to retry address verification when processor used by system is down.
ServiceNotAvailable	Boolean	Indicates whether system used to verify the address is available.

PccAddv Methods

Method Name	Returned Value	Description - PccAddv Methods
CreateAddressFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
ShouldShow	Boolean	Indicates whether AVS setup form is visible
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PccADSISetup

This class contains Alliance Data Systems extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccADSISetup Properties

Property Name	Data Type	Description - PccADSISetup Properties
BatchNumber	String	The Current batch number. This value is incremented by the processor after each successful settlement.
BType	*Integer*	*N/A*
Canceled	*Boolean*	*Used internally*
CompanyID	String	The company ID assigned by the merchant's bank or processor
CompanyIdentifier	String	A unique company identifier assigned to each merchant by Alliance Data Systems.
DebitReturnAmount	String	Amount of Debit and EBT Returns
DebitReturnCount	String	Count of Debit and EBT Returns
DebitSalesAmount	String	Amount of Debit and EBT Sales
DebitSalesCount	String	Count of Debit and EBT Sales
DebitVoidAmount	String	Amount of Debit and EBT Voids
DebitVoidCount	String	Count of Debit And EBT Voids
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
MCSalesAmount	String	Amount of MC Sales
MCSalesCount	String	Count of MC Sales
PLabel	Boolean	Flag that indicates if the merchant will accept Private Label Cards.
PLabelSalesAmount	String	Amount of Private Label Sales
PLabelSalesCount	String	Count of Private Label Sales
PurchasedAmount	String	Amount of credit purchase
PurchasedCount	String	Count of credit purchases or post authorizations
ReturnedAmount	String	Amount of credit Returns
ReturnedCount	String	Count of credit Returns
SequenceNumber	String	Unique sequence number that is assigned to each transaction. Also called a reference number.
VisaSalesAmount	String	Amount of Visa Sales
VisaSalesCount	String	Count of Visa Sales
VoidedAmount	String	Amount of credit Voids
VoidedCount	String	Count of credit Voids
VoidReturnAmount	String	Amount of Credit Void Returns
VoidReturnCount	String	Count of credit Void Returns

PccADSISetup Methods

Method Name	Returned Value	Description - PccADSISetup Methods
CreateADSIExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PccAmexDialSetup

This class contains Amex split-dial settings for the current instance of **PC**Charge. This is a Multi Use class.

PccAmexDialSetup Properties

Property Name	Data Type	Description - PccAmexDialSetup Properties
AMEXDirect	Boolean	Indicates whether split dial will be used
BType	String	The merchant's business type. **Valid values:** 0 – Retail / MOTO 4 – Restaurant
Canceled	*Boolean*	*Used internally*
ExpansionFactor	String	Percent by which restaurant-based transactions will be incremented during gratuity-related transactions. This setting only applies when transactions are processed using the **PC**Charge GUI.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
MerchantNumber	String	The American Express-assigned Service Establishment (SE) Number. **Length:** 10 bytes
PrimaryPhone	String	The Primary number that will be used when processing transactions via dial-up modem.
RequireServerID	String	Indicates whether **PC**Charge will require a Server ID during gratuity-related restaurant transactions. Consult the section **Restaurant Transactions** for more information (see page 105). **Valid Values:** 0 – Server ID not required 1 – Server ID required
SecondaryPhone	String	The Secondary number that will be used when processing transactions via dial-up modem.
TerminalID	String	The American Express-assigned Terminal ID. **Length:** 2 bytes

PccAmexDialSetup Methods

Method Name	Returned Value	Description - PccAmexDialSetup Methods
CreateAmexspltFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PccAmexSettleSetup

This class contains Amex Settlement settings for the current instance of **PC**Charge. This is a Multi Use class.

PccAmexSettleSetup Properties

Property Name	Data Type	Description - PccAmexSettleSetup Properties
AmexSettle	*Boolean*	*Used internally*
BatchNumber	*Integer*	*Used internally*
BType	*Byte*	*Used internally*
Canceled	*Boolean*	*Used internally*
CurrencyCode	String	The Currency Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's settlement currency. **Length:** 3 digits. **Valid Value:** 840 – U.S. Dollars
DefaultDescriptor	String	General Description of what type of transactions will be processed by this merchant. This value should be determined by the merchant and American Express. **Length:** 23 Characters. **Example:** MENS WEAR, HARDWARE, ACCESSORIES
FileSeqNumber	Integer	The File Sequence Number is the PCID-specific, unique sequence number for a file. That means that if you deliver transactions to American Express for more than one PCID, each PCID's transactions are forwarded in a separate Financial Settlement File with its own file sequence numbering scheme. Note: A file sequence number must not be repeated during a calendar year. **Length:** 6 bytes. **Example:** 000001
HostPort	String	The FTP port used when processing AMEX settlements. This value is provided by American Express.
HostURL	String	The FTP address used when processing AMEX settlements. This value is provided by American Express.
Index	String	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
InvoiceBatchCode	String	The Invoice Batch Code assigned by American Express. This code specifies processing options applicable to the batch. The code to be entered in this field is supplied by American Express at the same time the Process Control ID (PCID) is assigned. **Length:** 3 bytes
InvoiceSubcode	String	The Invoice Subcode assigned by American Express. This code specifies additional Amex controls and processing requirements. The code to be entered in this field is supplied by American Express at the same time the Process Control ID (PCID) and Invoice Batch code are assigned. **Length:** 2 bytes
Password	String	The password used to send settlements to American Express via FTP. This value is provided by American Express.
PCID	String	The Process Control ID / Username assigned by American Express. This parameter is a unique identifier code assigned to merchants, locations, and Authorized Third Party Processors so that they can directly access the American Express Financial Settlement system to deliver settlement data. In some cases, a merchant may not receive a unique PCID, because many franchises and authorized processors submit files for multiple locations under one PCID. **Length:** 6 bytes
RecSeqNumber	*Integer*	*Used internally*
RequestTimeout	*String*	*Not Yet Implemented*
SettlementFileName	*String*	*Used internally*
TotalTransAmt	*Long*	*Used internally*
TransCount	*Long*	*Used internally*

Property Name	Data Type	Description - PccAmexSettleSetup Properties
Username	String	The Process Control ID / Username assigned by American Express. This parameter is a unique identifier code assigned to merchants, locations, and Authorized Third Party Processors so that they can directly access the American Express Financial Settlement system to deliver settlement data. In some cases, a merchant may not receive a unique PCID, because many franchises and authorized processors submit files for multiple locations under one PCID. **Length:** 6 bytes

PccAmexSettleSetup Methods

Method Name	Returned Value	Description - PccAmexSettleSetup Methods
CreateArchiveDir	*Boolean*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PccBPASSetup

This class contains FDMS Atlanta extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccBPASSetup Properties

Property Name	Data Type	Description - PccBPASSetup Properties
AppID	*String*	*Used internally*
BType	Integer	The merchant's business type. **Valid values:** 0 = Retail 1 = Mail order 2 = Electronic commerce 3 = Restaurant
Canceled	*Boolean*	*Used internally*
Connect	String	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP EFSnet XML 2 – TCP/IP EFSnet Leased Line 3 – TCP/IP EFSnet Pass Through
CurrencyCode	String	The Currency Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's settlement currency. **Length:** 3 digits. **Valid Value:** 840 – U.S. Dollars
DeviceNumber	String	Device number assigned by the merchant's bank or processor. **Length:** 3 digits. **Example:** 001.
DeviceType	String	Device type assigned by the merchant's bank or processor. **Length:** 2 characters. **Example:** 5S.
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
ExpansionFactor	String	Percent by which restaurant-based transactions will be incremented during gratuity-related transactions. This setting only applies when transactions are processed using the **PC**Charge GUI.
FSInquiry	*Boolean*	*Used internally*
GiftConnect	Integer	Indicates method of connection to processor when processing gift cards. **Valid values:** 0 – Dial-up 1 – TCP/IP EFSnet XML 2 – TCP/IP EFSnet Leased Line 3 – TCP/IP EFSnet Pass Through
GiftCurrencyCode	String	The Currency Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's settlement currency. **Length:** 3 digits. **Valid Value:** 840 – U.S. Dollars
GiftDeviceNumber	String	Device number assigned by the merchant's bank or processor. **Length:** 3 digits. **Example:** 001.
GiftDeviceType	String	Device type assigned by the merchant's bank or processor. **Length:** 2 characters. **Example:** 5S.
GiftDialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
GiftPort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
GiftSequenceNumber	String	A unique sequence number assigned to each transaction by **PC**Charge. **Length:** 6 digits

Property Name	Data Type	Description - PccBPASSetup Properties
GiftSettleTimeOut	String	The Internet Settlement Timeout Value. If GiftDialBackup is set to TRUE, GiftSettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
GiftStateCode	String	The merchant's State Code. **Length:** 2 digits **Valid Values:** Alabama - 01; Alaska - 02; Arizona - 04; Arkansas - 05; California - 06; Colorado - 08; Connecticut - 09; Delaware - 10; District of Columbia - 11; Florida - 12; Georgia - 13; Puerto Rico - 14; Hawaii - 15; Idaho - 16; Illinois - 17; Indiana - 18; Iowa - 19; Kansas - 20; Kentucky - 21; Louisiana - 22; Maine - 23; Maryland - 24; Massachusetts - 25; Michigan - 26; Minnesota - 27; Mississippi - 28; Missouri - 29; Montana - 30; Nebraska - 31; Nevada - 32; New Hampshire - 33; New Jersey - 34; New Mexico - 35; New York - 36; North Carolina - 37; North Dakota - 38; Ohio - 39; Oklahoma - 40; Oregon - 41; Pennsylvania - 42; Rhode Island - 44; South Carolina - 45; South Dakota - 46; Tennessee - 47; Texas - 48; Utah - 49; Vermont - 50; Virginia - 51; Virgin Islands - 52; Washington - 53; West Virginia - 54; Wisconsin - 55; Wyoming - 56
GiftStoreID	String	EFSnet store number assigned by the merchant's bank or processor. **Length:** 32 digits.
GiftStoreKey	String	EFSnet store password assigned by the merchant's bank or processor. **Length:** 64 digits
GiftTimeOut	String	The Internet Authorization Timeout Value. If GiftDialBackup is set to TRUE, GiftTimeOut determines how long **PC**Charge will wait for a gift card transaction to time out before attempting the transaction via dial **Format:** Seconds
GiftURL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
PCard	Boolean	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE - Enable fields FALSE - Disable fields
Port	String	The system/socket port used to connect to the processor when processing via TCP/IP.
ReversalOption	String	0 - Retry reversals the specified number of times set in the .RetryReversalCount property. 1 - Retry reversals indefinitely
ReversalNumberRetry	String	The number of times a reversal is retried
RetryReversalSeconds	String	The amount of time (in seconds) between reversal retries
ReversalSeconds	String	The amount of time (in seconds) between reversal processing
RequireServerID	String	Indicates whether **PC**Charge will require a Server ID during gratuity-related restaurant transactions. Consult the section **Restaurant Transactions** for more information (see page 105). **Valid Values:** 0 - Server ID not required 1 - Server ID required
SequenceNumber	String	A unique sequence number assigned to each transaction by **PC**Charge. **Length:** 6 digits
SettlePwd	String	Password required for settlement
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds

Property Name	Data Type	Description - PccBPASSetup Properties
StateCode	String	The merchant's State Code. **Length:** 2 digits **Valid Values:** Alabama - 01; Alaska - 02; Arizona - 04; Arkansas - 05; California - 06; Colorado - 08; Connecticut - 09; Delaware - 10; District of Columbia - 11; Florida - 12; Georgia - 13; Puerto Rico - 14; Hawaii - 15; Idaho - 16; Illinois - 17; Indiana - 18; Iowa - 19; Kansas - 20; Kentucky - 21; Louisiana - 22; Maine - 23; Maryland - 24; Massachusetts - 25; Michigan - 26; Minnesota - 27; Mississippi - 28; Missouri - 29; Montana - 30; Nebraska - 31; Nevada - 32; New Hampshire - 33; New Jersey - 34; New Mexico - 35; New York - 36; North Carolina - 37; North Dakota - 38; Ohio - 39; Oklahoma - 40; Oregon - 41; Pennsylvania - 42; Rhode Island - 44; South Carolina - 45; South Dakota - 46; Tennessee - 47; Texas - 48; Utah - 49; Vermont - 50; Virginia - 51; Virgin Islands - 52; Washington - 53; West Virginia - 54; Wisconsin - 55; Wyoming - 56
StoreID	String	EFSnet store number assigned by the merchant's bank or processor. **Length:** 32 digits
StoreKey	String	EFSnet store password assigned by the merchant's bank or processor. **Length:** 64 digits.
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.

PccBPASSetup Methods

Method Name	Returned Value	Description - PccBPASSetup Methods
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.
ShowGC	Boolean	Shows a GUI form that allows the end-user to enter extended or advanced Gift Card configuration information such as the business type, communication method, or other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowGC.
SynchronizeSequenceNum	*None*	*Used internally*

PccBPSGiftSetup

This class contains Fifth-Third – St. Pete gift card extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccBPSGiftSetup Properties

Property Name	Data Type	Description - PccBPSGiftSetup Properties
BankID	String	Acquiring Institution Identification Code assigned by the merchant's bank or processor. **Length:** 4 digits
BType	*Integer*	*Used internally*
Canceled	*Boolean*	*Used internally*
Connect	String	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
DialBackup	*Boolean*	*N/A*
GiftConnect	Integer	Indicates method of connection to processor when processing gift cards. **Valid values:** 0 – Dial-up 1 – TCP/IP
GiftDialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
GiftPassword	String	The Password assigned by the merchant's bank or processor for TCP/IP processing.
GiftPort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
GiftSettleTimeOut	String	The Internet Settlement Timeout Value. If GiftDialBackup is set to TRUE, GiftSettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
GiftTimeOut	String	The Internet Authorization Timeout Value. If GiftDialBackup is set to TRUE, GiftTimeOut determines how long **PC**Charge will wait for a gift card transaction to time out before attempting the transaction via dial **Format:** Seconds
GiftReversalNumberRetry	String	The number of times to retry a reversal
GiftRetryReversalSeconds	String	The amount of time (in seconds) between reversal retries
GiftReversalSeconds	String	The amount of time (in seconds) between reversal processing
GiftURL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
GiftUsername	String	The Username assigned by the merchant's bank or processor for TCP/IP processing.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
Password	*String*	*N/A*
PCard	*Boolean*	*Used internally*
Port	*String*	*N/A*
PrimaryAuthIP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Retrieval	String	Retrieval Reference Number. Used to identify and track the original transaction. This value is assigned by the merchant's processor. **Length:** 8 digits

Property Name	Data Type	Description - PccBPSGiftSetup Properties
SettleTimeOut	*String*	*N/A*
TerminalID	String	Contains terminal ID for merchant
TimeOut	*Long*	*N/A*
URL	*String*	*N/A*
Username	*String*	*N/A*

PccBPSGiftSetup Methods

Method Name	Returned Value	Description - PccBPSGiftSetup Methods
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Integer	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.

PccBPSSetup

This class contains Fifth-Third – St. Pete extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccBPSSetup Properties

Property Name	Data Type	Description - PccBPSSetup Properties
BankID	String	Acquiring Institution Identification Code assigned by the merchant's bank or processor. **Length:** 4 digits
BType	Integer	The merchant's business type. **Valid values:** 0 – Retail 1 – Mail order 2 – Electronic Commerce
Canceled	*Boolean*	*Used internally*
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
Password	String	The Password assigned by the merchant's bank or processor for TCP/IP processing.
PCard	Boolean	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE – Enable fields FALSE – Disable fields
Port	String	The system/socket port used to connect to the processor when processing via TCP/IP.
PrimaryAuthIP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Retrieval	String	Retrieval Reference Number. Used to identify and track the original transaction. This value is assigned by the merchant's processor. **Length:** 8 digits
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
TerminalID	String	Card Acceptor Terminal ID code. This value identifies the terminal at the merchant (card acceptor) location at which the transaction was entered. **Length:** 3 digits
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Username	String	The Username assigned by the merchant's bank or processor for TCP/IP processing.

PccBPSSetup Methods

Method Name	Returned Value	Description - PccBPSSetup Methods
CreateBPSExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Integer	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.

PccCESSetup

This class contains FDMS North/Cardnet extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccCESSetup Properties

Property Name	Data Type	Description - PccCESSetup Properties
BType	String	The merchant's business type. **Valid values:** 0 – Retail 1 – Mail order 2 – Electronic Commerce
Canceled	*Boolean*	*Used internally*
CheckService	Integer	Indicates if merchant is set up for check services. **Valid Values:** 0 – None 1 – ETC 2 – Equifax 3 – NPC 4 – TeleCheck
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-Up 1 – First Data IPN (Datawire – TCP/IP)
CSNumber	String	The Merchant's customer service phone number. This number will be printed on the customer's statement if PrintCSNumber is set to TRUE. Not applicable for Retail transactions.
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
DID	String	The Datawire ID. The value is provided to the merchant by their Merchant Service Provider or Processing company. The DID is required to process transactions via the Internet using the Datawire network. This value will be unique for each merchant number used.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
IP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
MaxBatchSize	String	Specifies the maximum number of transactions per batch that **PC**Charge will send to the processor. If the number of transactions to be settled is greater than the number specified in this setting, **PC**Charge will split the batch into multiple batches, each containing (at most) the number transactions specified in this setting. The batches are then sent to the processor one at a time. **Example:** A merchant has 250 transaction to settle and the MaxBatchSize is set to 100. **PC**Charge will send two 100-transaction batches and one 50-transaction batch. **Max Value:** 999
MCReversal	*Integer*	*N/A*
PCard	Boolean	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE – Enable fields FALSE – Disable fields
PrintCSNumber	Boolean	Flag that indicates whether to print customer service phone number on receipt. Not applicable for Retail transactions.

Property Name	Data Type	Description - PccCESSetup Properties
RetryCount	*Integer*	*Used internally*
SecondaryIP	*Boolean*	*Used internally*
SendError	*Boolean*	*Used internally*
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
URL2	String	The Secondary Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
URLAddress	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
VSReversal	*Boolean*	*N/A*

PccCESSetup Methods

Method Name	Returned Value	Description - PccCESSetup Methods
CreateCESAdvanceFile	*None*	*Used internally*
CreateCESExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.

PccCheckSetup

This class contains information about the credit card company setup in the current instance of **PC**Charge.
This is a Global Multi Use class.

Note: The extended information for check processors that support conversion and guarantee can be accessed by using the `PccCreditSetup` class.

PccCheckSetup Properties

Property Name	Data Type	Description - PccCheckSetup Properties
Canceled	*Boolean*	*Used internally*
Company	*Integer*	*Used internally*
DemoMode	*Boolean*	*Used internally*
Index	Integer	The Merchant Number index. If `Index` is set to a value greater than `0`, the `tid.pcc` file will be accessed and the merchant number at that index in the file will be used. `Index` should be set prior to calling the `Load`, `Save`, or `Show` methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
PrimaryPhone	String	The Primary number that will be used when processing transactions via dial-up modem.
Processor	String	The code for the processing company. This value can be no more than four characters and must be capitalized. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
Records	Integer	Read-only property that Indicates how many merchant numbers have been set up in the current `tid.pcc` file.
SecondaryPhone	String	The Secondary number that will be used when processing transactions via dial-up modem.
Service	Integer	Indicates the type of check service that is supported. **Valid Values:** `0` – MICR `1` – Checks on Delivery `2` – Driver's License `3` – Double ID
TID	String	The check company Site ID / merchant number assigned by the merchant's bank or processor.
Version	*String*	*Used internally*

PccCheckSetup Methods

Method Name	Returned Value	Description - PccCheckSetup Methods
CreateCheckFile	*None*	*Used internally*
CreateChkExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved.

PccCompany

This class contains information about the company name setup for the current instance of **PC**Charge and the associated active merchant number index. This is a Multi Use class.

PccCompany Properties

Property Name	Data Type	Description - PccCompany Properties
Canceled	*Boolean*	*Used internally*
City	String	The name of city in which merchant's company is located.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
LogonScreen	*Boolean*	*N/A*
Name	String	The name of merchant's company.
State	String	The abbreviation of state in which merchant's company is located.
Street	String	The street address of merchant's company.
Zip	String	The zip code of region in which merchant's company is located.

PccCompany Methods

Method Name	Returned Value	Description - PccCompany Methods
CreateCompanyFile	*None*	*Used internally*
IsCompanyInfoValid	Boolean	Performs a check that determines whether or not the Merchant's company information has been properly entered. Returns TRUE if the company information is valid, FALSE otherwise.
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Load. **Note:** Set the Index property prior to calling Show.

PccConfig

This class contains information about the global preferences for the current instance of **PC**Charge. This is a Multi Use class.

PccConfig Properties

Property Name	Data Type	Description - PccConfig Properties
AcceptAll	String	Flag that indicates whether merchant accept all card types. **Valid Values:** 0 – Do not accept all card types (specify individual card types using the other AcceptXXXX properties and the PrivateLabel property) 1 – Accept all card types
AcceptAMEX	String	Flag that Indicates whether merchant will accept American Express cards. **Valid Values:** 0 – Do not accept American Express cards 1 – Accept American Express cards
AcceptCBLN	String	Flag that Indicates whether merchant will accept Carte Blanche cards. **Valid Values:** 0 – Do not accept Carte Blanche cards 1 – Accept Carte Blanche cards
AcceptDCCB	String	Flag that Indicates whether merchant will accept Diner's Club cards. **Valid Values:** 0 – Do not accept Diner's Club cards 1 – Accept Diner's Club cards
AcceptDISC	String	Flag that Indicates whether merchant will accept Discover cards. **Valid Values:** 0 – Do not accept Discover cards 1 – Accept Discover cards
AcceptENRT	String	Flag that Indicates whether merchant will accept Enroute cards. **Valid Values:** 0 – Do not accept Enroute cards 1 – Accept Enroute cards
AcceptJAL	String	Flag that Indicates whether merchant will accept Japanese Airlines (JAL) cards. **Valid Values:** 0 – Do not accept Japanese Airlines (JAL) cards 1 – Accept Japanese Airlines (JAL) cards
AcceptJCB	String	Flag that Indicates whether merchant will accept JCB cards. **Valid Values:** 0 – Do not accept JCB cards 1 – Accept JCB cards
AcceptMC	String	Flag that Indicates whether merchant will accept MasterCard cards. **Valid Values:** 0 – Do not accept MasterCard cards 1 – Accept MasterCard cards
AcceptVISA	String	Flag that Indicates whether merchant will accept Visa cards. **Valid Values:** 0 – Do not accept Visa cards 1 – Accept Visa cards
AddCustomer	Boolean	Flag that indicates whether to prompt user to have **PC**Charge add customer to the customer database. **Valid Values:** TRUE – Prompt FALSE – Do not prompt **Note:** This setting only applies to transaction processing using the **PC**Charge GUI.
AddVerify	*Boolean*	*Used internally*
BillPayPrompt	Boolean	Flag that indicates whether to prompt user to indicate whether the transaction is a bill payment.

Property Name	Data Type	Description - PccConfig Properties
CCUser	String	Flag available for credit card duplicate checking criteria. Enables or disables duplicate checking by user name. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** Off **Valid Values:** 0 – Off 1 - On
CCMerchant	String	Flag available for credit card duplicate checking criteria. Enables or disables duplicate checking by merchant number. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** On **Valid Values:** 0 – Off 1 - On
CCCard	String	Flag available for credit card duplicate checking criteria. Enables or disables duplicate checking by card number. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** On **Valid Values:** 0 – Off 1 - On
CCExpDate	String	Flag available for credit card duplicate checking criteria. Enables or disables duplicate checking by expiration date. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** On **Valid Values:** 0 – Off 1 - On
CCTicket	String	Flag available for credit card duplicate checking criteria. Enables or disables duplicate checking by ticket number. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** Off **Valid Values:** 0 – Off 1 - On
CCAmount	String	Flag available for credit card duplicate checking criteria. Enables or disables duplicate checking by amount. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** On **Valid Values:** 0 – Off 1 - On
Canceled	*Boolean*	*Used internally*
chkAuthAmounts	String	Flag that indicates whether to process a Post-Auth if the amount of the Post-Auth is greater than the amount of the corresponding Pre-Auth. **Valid Values:** 0 – Do not enable the processing of Post-Auths with a greater amounts than their corresponding Pre-Auths. 1 – Enable the processing Post-Auths with a greater amounts than their corresponding Pre-Auths. **Note:** Settling a transaction with an amount that is different (whether less than or greater than) than the original authorization amount can cause transactions to downgrade.
ChkContracts	Boolean	Flag that indicates whether to notify the user when recurring billing contracts are due at startup. **Valid Values:** TRUE – Notify user FALSE – Do not notify user **Note:** This setting only applies to transaction processing using the **PC**Charge GUI.

Property Name	Data Type	Description - PccConfig Properties
ChkDuplicates	Boolean	Flag that indicates whether **PC**Charge requires duplicate transactions to be forced. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Valid Values:** TRUE – Require duplicate transactions to be forced FALSE – Do not require duplicate transactions to be forced
ChkIP	Boolean	Flag that Indicates whether to activate the TCP Interface for incoming transaction requests. Consult the **TCP Interface** section (see page 522) for more information on using the TCP/IP Interface. **Valid Values:** TRUE – Activate TCP Interface FALSE – (Default) Do not activate TCP Interface
ChkMSV	String	Flag that indicates whether to activate Magnetic Strip Verification. **Valid Values:** 0 – Do not enable Magnetic Strip Verification 1 – Enable Magnetic Strip Verification **Note:** This setting only applies to transactions performed using the **PC**Charge GUI.
chkPrompt	Boolean	Flag that indicates whether to prompt user for CPS 2000 Qualifiers (ticket number and zip code). **Valid Values:** TRUE – Prompt for CPS 2000 Qualifiers FALSE – Do not prompt for CPS 2000 Qualifiers **Note:** This setting only applies to transaction processing using the **PC**Charge GUI.
DCUser	String	Flag available for debit card duplicate checking criteria. Enables or disables duplicate checking by user name. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** Off **Valid Values:** 0 – Off 1 - On
DCMerchant	String	Flag available for debit card duplicate checking criteria. Enables or disables duplicate checking by merchant number. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** On **Valid Values:** 0 – Off 1 - On
DCCard	String	Flag available for debit card duplicate checking criteria. Enables or disables duplicate checking by card number. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** On **Valid Values:** 0 – Off 1 - On
DCExpDate	String	Flag available for debit card duplicate checking criteria. Enables or disables duplicate checking by expiration date. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** On **Valid Values:** 0 – Off 1 - On
DCTicket	String	Flag available for debit card duplicate checking criteria. Enables or disables duplicate checking by ticket number. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** Off **Valid Values:** 0 – Off 1 - On

Property Name	Data Type	Description - PccConfig Properties
DCAmount	String	Flag available for debit card duplicate checking criteria. Enables or disables duplicate checking by amount. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** On **Valid Values:** 0 – Off 1 - On
Days	String	Flag available for duplicate checking criteria. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** 01
DefaultTID	Boolean	Use Default Processor Flag. This flag indicates whether to use the first merchant number set up in **PCC**harge if one is not specified in the transaction request. Consult the **Multi-Merchant Support** section (see page 68) for more information on the "Use Default Processor" option. **Valid Values:** TRUE – Use the first merchant number. FALSE – Do not use the first merchant number.
EBUser	String	Flag available for EBT card duplicate checking criteria. Enables or disables duplicate checking by user name. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** Off **Valid Values:** 0 – Off 1 - On
EBMerchant	String	Flag available for EBT card duplicate checking criteria. Enables or disables duplicate checking by merchant number. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** On **Valid Values:** 0 – Off 1 - On
EBCard	String	Flag available for EBT card duplicate checking criteria. Enables or disables duplicate checking by card number. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** On **Valid Values:** 0 – Off 1 - On
EBTicket	String	Flag available for EBT card duplicate checking criteria. Enables or disables duplicate checking by ticket number. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** Off **Valid Values:** 0 – Off 1 - On
EBAmount	String	Flag available for EBT card duplicate checking criteria. Enables or disables duplicate checking by amount. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** On **Valid Values:** 0 – Off 1 - On

Property Name	Data Type	Description - PccConfig Properties
EnableTCPIPClientReversals	Boolean	Flag that indicates whether to enable client-side TCP/IP reversals. Client-side TCP/IP reversals are generated for some processors when the connection to the client is lost before **PC**Charge is able to return the transaction response to the client. Client-side reversals, when enabled, will help prevent double charges in cases where the client does not receive the response from the client as expected. **Valid Values:** TRUE – Enable TCP/IP Client Reversals FALSE – Disable TCP/IP Client Reversals **Note:** This option should be disabled if using a processor other than FDMS Atlanta.
EnableTotTime	String	Flag that indicates whether to return the Total Elapsed Time with the transaction response. **Valid Values:** 0 – Do not return the Total Elapsed Time 1 – Return the Total Elapsed Time
EnableTransTime	String	Flag that indicates whether to return the Transaction Elapsed Time with the transaction response. **Valid Values:** 0 – Do not return the Transaction Elapsed Time 1 – Return the Transaction Elapsed Time
Encrypt	*Boolean*	*N/A -Transaction data is always encrypted based on CISP guidelines. Refer to the **Important Security Notice** (see page 9) for more information.*
Hours	String	Flag available for duplicate checking criteria. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** 00
EncryptSettle	*Boolean*	*N/A -Transaction data is always encrypted based on CISP guidelines. Refer to the **Important Security Notice** (see page 9) for more information.*
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
IPAddress	*String*	*N/A*
LastValidYear	String	The last year that will be considered a valid expiration date. Currently, the default value is "09". **Length:** 2 digits. **Format:** YY **Example:** If LastValidDate is set to 05, then cards between 06 and 99 are considered to be 1906 to 1999, and cards between 00 and 05 are 2000 to 2005.
Minutes	String	Flag available for duplicate checking criteria. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** 00
MTimer	String	Multi-trans wait flag. This flag indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid Values:** 0 – Do not leave the modem connection open after each transaction 1 – Attempt to leave the modem connection open after each transaction See the section **Multi-trans Wait** for more information (see page 67). This Flag has no effect if processing will occur over IP or leased line.
Pass	String	The password for the 'system' user ID. Setting a password activates cashier permissions (this feature is only available in the **PC**Charge GUI) Note: It is recommended that a password always be set when using PCCharge in a live environment.
Port	Integer	If the TCP Interface is activated (ChkIP = TRUE), this value is the TCP port that will be used for incoming transaction requests. Consult the **TCP Interface** section (see page 522) for more information on using the TCP/IP Interface. **Default Value:** 31419 **Note:** The default port of 31419 should be changed to maximize security when processing transactions in a live environment.
PrivateKey	*String*	*N/A*

Property Name	Data Type	Description - PccConfig Properties
PrivateLabel	Boolean	Flag that Indicates whether merchant will accept Private Label cards. **Valid Values:** TRUE – Accept Private Label cards FALSE – Do not accept Private Label cards
ProxyServer	*Boolean*	*N/A*
QTimer	String	This setting determines how often **PC**Charge will poll its directory for incoming transaction requests (.inx and .inp files). The default value of 00.50 should not be changed unless the client machine is slow or there is network lag. **Format:** seconds.
ReAuthAttempts	*String*	*Used internally*
Seconds	String	Flag available for duplicate checking criteria. See the description of "Duplicate Transactions" in the **Warnings, Tips, and Guidelines** section (see page 55) for more information. **Default Value:** 00
SecureCustomerDB	Boolean	Flag that indicates whether to mask credit card number on the Customer Database screen. **Valid Values:** TRUE – Mask credit card numbers FALSE – Do not mask credit card numbers **Note:** This setting only applies to transaction processing using the **PC**Charge GUI.
UseProxyServer	*Boolean*	*N/A*
Version	*String*	*Used internally*

PccConfig Methods

Method Name	Returned Value	Description - PccConfig Methods
CreateConfigFile	*None*	*Used internally*
GetMerchantInfo	String	The GetMerchantInfo method returns a string containing all of the merchant numbers and processors set up in **PC**Charge. The string will also indicate whether the processor is Host based (H), Terminal based (T), or a hybrid (Y). The string will begin with STX and will end with ETX. GS will separate each record, and FS will separate fields within a record. **Example:** <STX>CES <FS>0000009279962967<FS>T<GS>GSAR<FS> 999999999999519<FS>T<GS>VISA<FS>999999999911<FS>T<ETX> Refer to the section **Multi-Merchant Support** (see page 68) for more information on the GetMerchantInfo method.
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PccCreditSetup

This class contains information about the credit card company setup in the current instance of **PC**Charge. This is a Multi Use class.

PccCreditSetup Properties

Property Name	Data Type	Description - PccCreditSetup Properties
Canceled	*Boolean*	*Used internally*
Company	*Integer*	*Used internally*
Default	Boolean	Flag that indicates whether to use default phone numbers. If this value is set to TRUE, any changes to the values in PrimaryPhone and SecondaryPhone will not take effect.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
MerchantNumber	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. **Note:** This property cannot be modified using the Save method.
PccADSISetup	PccADSISetup	Provides extended setup information for Alliance Data Systems, Inc.
PccAmexDialSetup	PccAmexDialSetup	Provides extended setup information for American Express split dial
PccAmexSettleSetup	PccAmexSettleSetup	Provides extended setup information for American Express direct settlement
PccASISetup	*PccASISetup*	*No longer Supported*
PccBMONSetup	*PccBMONSetup*	*No longer Supported*
PccBPASSetup	PccBPASSetup	Provides extended setup information for FDMS Atlanta, Inc.
PccBPSSetup	PccBPSSetup	Provides extended setup information for Fifth-Third Bank – St. Pete
PccCESSetup	PccCESSetup	Provides extended setup information for FDMS North / Cardnet
PccDiscDialSetup	*PccDiscDialSetup*	*No longer Supported*
PccEchoSetup	PccEchoSetup	Provides extended setup information for ECHO
PccENCNSetup	*PccENCNSetup*	*No longer Supported*
PccEZCKSetup	PccEZCKSetup	Provides extended setup information for Check Services powered by RMRS (Check Conversion settings)
PccFDCNSetup	PccFDCNSetup	Provides extended setup information for FDMS Nashville / Envoy
PccFDCSetup	PccFDCSetup	Provides extended setup information for FDMS Omaha / FDR
PccGsarSetup	PccGsarSetup	Provides extended setup information for Chase Paymentech
PccHPTSSetup	PccHPTSSetup	Provides extended setup information for Heartland Payment Systems
PccIPGSSetup	*PccIPGSSetup*	*No longer Supported*
PccISDNSetup	*PccISDNSetup*	*No longer Supported*
PccLYNKSetup	PccLYNKSetup	Provides extended setup information for RBS WorldPay, Inc.
PccMDISetup	*PccMDISetup*	*No longer Supported*
PccMPSSetup	*PccMPSSetup*	*No longer Supported*
PccNBSetup	PccNBSetup	Provides extended setup information for FDMS South / NaBanco
PccNBSSetup	PccNBSSetup	Provides extended setup information for National Bankcard Services
PccNDCSetup	PccNDCSetup	Provides extended setup information for Global Payments-East
PccNovaSetup	PccNovaSetup	Provides extended setup information for Elavon (NOVA)
PccNovusSetup	*PccNovusSetup*	*No longer Supported*
PccNPCSetup	PccNPCSetup	Provides extended setup information for National Processing Company

Property Name	Data Type	Description - PccCreditSetup Properties
PccRMRSSetup	PccRMRSSetup	Provides extended setup information for National Check Network (Check Conversion settings)
PccSPSSetup	PccSPSSetup	Provides extended setup information for Secure Payment Systems (Check Guarantee settings)
PccTelmSetup	*PccTelmSetup*	*No longer Supported*
PccTMHSetup	*PccTMHSetup*	*No longer Supported*
PccVisaSetup	PccVisaSetup	Provides extended setup information for TSYS
PFlag	*String*	*Used internally*
PreFix	*String*	*Used internally*
PrimaryPhone	String	The Primary number that will be used when processing transactions via dial-up modem.
Processor	String	The code for the processing company. This value can be no more than four characters and must be capitalized. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152). **Note:** This property cannot be modified using the Save method.
SecondaryPhone	String	The Secondary number that will be used when processing transactions via dial-up modem.
Version	*String*	*Used internally*

PccCreditSetup Methods

Method Name	Returned Value	Description - PccCreditSetup Methods
AddMerchantNumber	*Boolean*	*Used internally*
AddNewTID	*None*	*Used internally*
CreateCreditFile	*None*	*Used internally*
GetIndex	*Integer*	*Used internally*
GetPCCVersion	String	Returns the version number of the PCCharge that is currently running.
GetRecords	Integer	Returns how many merchant numbers have been set up in the current tid.pcc file.
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
ProcessorSetup	*Object*	*Used internally*
RemoveMerchantNumber	*Boolean*	*Used internally*
RemoveTID	*None*	*Used internally*
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	None	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved.
VerifyTID	Boolean	The VerifyTID method returns TRUE if the processor code merchant number that is passed to it is set up in **PC**Charge, otherwise, FALSE is returned. Specifically, this method checks for the merchant number in the file TID.PCC, which is located in the **PC**Charge directory.

PccDebitSetup

Contains debit card company information about the current instance of **PC**Charge. This is a Multi Use class.

PccDebitSetup Properties

Property Name	Data Type	Description - PccDebitSetup Properties
Canceled	*Boolean*	*Used internally*
Company	*String*	*Used internally*
Default	Boolean	Flag that indicates whether to use default phone numbers. If this value is set to TRUE, any changes to the values in PrimaryPhone and SecondaryPhone will not take effect.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
KeyManagement	*Integer*	*Used internally*
MasterKey	*Integer*	*Used internally*
MerchantNumber	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. **Max Length**: 32 characters. This value can be alphanumeric.
PccBPASSetup	PccBPASSetup	Provides extended setup information for FDMS Atlanta
PccHPTSSetup	PccHPTSSetup	Provides extended setup information for Heartland Payment Systems
PccLYNKSetup	PccLYNKSetup	Provides extended setup information for RBS WorldPay
PccMPSSetup	*PccMPSSetup*	*No longer Supported*
PccNBSetup	PccNBSetup	Provides extended setup information for FDMS South / NaBanco
PccNBSSetup	PccNBSSetup	Provides extended seup information for National Bankcard Services
PccNDCSetup	PccNDCSetup	Provides extended setup information for Global Payments - East
PccNPCSetup	PccNPCSetup	Provides extended setup information for National Processing Company
PccVisaSetup	PccVisaSetup	Provides extended setup information for TSYS
PrimaryPhone	String	The Primary number that will be used when processing transactions via dial-up modem.
Processor	String	The code for the processing company. This value can be no more than four characters and must be capitalized. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
SecondaryPhone	String	The Secondary number that will be used when processing transactions via dial-up modem.
SerialNumber	*String*	*Used internally*
WorkingKey	String	The working key. Only applicable for Debit processors using Master Session encryption.

PccDebitSetup Methods

Method Name	Returned Value	Description - PccDebitSetup Methods
CreateDebitFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved.

PccDMRKSetup

This class contains Datamark extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccDMRKSetup Properties

Property Name	Data Type	Description - PccDMRKSetup Properties
BatchNumber	String	The Current batch number. This value is incremented by the processor after each successful settlement.
BType	Integer	The merchant's business type. **Valid values:**
Canceled	*Boolean*	*Used internally*
Client	String	Client Number assigned by the merchant's bank or processor. **Length:** 4 digits
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
EDCType	*Integer*	*Used internally*
ExpansionFactor	String	Percent by which restaurant-based transactions will be incremented during gratuity-related transactions. This setting only applies when transactions are processed using the **PC**Charge GUI.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
LastRRefNum	String	Retrieval Reference Number. Used to identify and track the original transaction. This value is assigned by the merchant's processor. **Length:** 8 digits
Password	String	The Password assigned by the merchant's bank or processor for TCP/IP processing.
Port	Long	The system/socket port used to connect to the processor when processing via TCP/IP.
PrimaryAuthIP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
PrimaryAuthPort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
RequireServerID	String	Indicates whether **PC**Charge will require a Server ID during gratuity-related restaurant transactions. Consult the section **Restaurant Transactions** for more information (see page 105). **Valid Values:** 0 – Server ID not required 1 – Server ID required
Sequence	String	This field is for identifying transactions within the batch. Assigned by **PC**Charge **Length:** 3 digits
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
SIIP	Boolean	The Service Industries Incentive Program indicator flag. Used to indicate recurring payments for service industries such as insurance, telecom and utilities.
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds

Property Name	Data Type	Description - PccDMRKSetup Properties
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Username	String	The Username assigned by the merchant's bank or processor for TCP/IP processing.

PccDMRKSetup Methods

Method Name	Returned Value	Description - PccDMRKSetup Methods
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.

PccEBTSetup

This class contains information about the EBT company setup in the current instance of **PC**Charge. This is a Multi Use class.

PccEBTSetup Properties

Property Name	Data Type	Description - PccEBTSetup Properties
Canceled	*Boolean*	*Used internally*
Company	*String*	*Used internally*
Default	Boolean	Flag that indicates whether to use default phone numbers. If this value is set to TRUE, any changes to the values in PrimaryPhone and SecondaryPhone will not take effect.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
KeyManagement	*Integer*	*Used internally*
MasterKey	*Integer*	*Used internally*
MerchantNumber	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. **Max Length**: 32 characters. This value can be alphanumeric.
PccBPASSetup	PccBPASSetup	Provides extended setup information for FDMS Atlanta
PccNPCSetup	PccNPCSetup	Provides extended setup information for National Processing Company
PccVisaSetup	PccVisaSetup	Provides extended setup information for TSYS
PrimaryPhone	String	The Primary number that will be used when processing transactions via dial-up modem.
Processor	String	The code for the processing company. This value can be no more than four characters and must be capitalized. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
SecondaryPhone	String	The Secondary number that will be used when processing transactions via dial-up modem.
WorkingKey	*String*	*N/A*

PccEBTSetup Methods

Method Name	Returned Value	Description - PccEBTSetup Methods
CreateEBTFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PccEchoSetup

This class contains ECHO extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccEchoSetup Properties

Property Name	Data Type	Description - PccEchoSetup Properties
BType	Integer	The merchant's business type. **Valid values:** 0 – Retail 1 – Direct Marketing 4 – Telemerchant 5 – Electronic Commerce
Canceled	*Boolean*	*Used internally*
Connect	Integer	Indicates modem dial protocol. **Valid values:** 0 – Compuserve 1 – Dial 800 Number
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
PCard	Boolean	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE – Enable fields FALSE – Disable fields

PccEchoSetup Methods

Method Name	Returned Value	Description - PccEchoSetup Methods
CreateECHOExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PccEZCKSetup

This class contains Check Services powered by RMRS extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccEZCKSetup Properties

Property Name	Data Type	Description - PccEZCKSetup Properties
BounceFee	String	The fee used in cases of returned checks due to insufficient funds. **Format:** Dollars
Canceled	*Boolean*	*Used internally*
ConnectType	String	
FTPAddress	String	FTP address for check image upload.
FTPPassword	String	Password for access to image upload FTP
FTPUser	String	User ID for access to image upload FTP
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
OwnerCode	String	Owner code for image upload FTP
SalesBalance	String	The current Sales Balance
SalesCount	String	The current Sales Count
Truncation	Boolean	Flag that indicates whether check truncation / conversion will occur.
TruncationTID	String	This is a unique identifier assigned by the merchant's bank or processor that identifies the merchant. It will be in the format: site id-location id-rule set , where site id can be from 1 to 5 characters (numeric), location id can be from 1 to 6 characters (numeric), and rule set can be from 1 to 4 characters (numeric). **Example:** 78-123456-9999 (dashes are necessary; no spaces).
VoidsBalance	String	The current Voids Balance
VoidsCount	String	The current Voids Count

PccEZCKSetup Methods

Method Name	Returned Value	Description - PccEZCKSetup Methods
CreateEZCKExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PccFDCNSetup

This class contains FDMS Nashville / Envoy extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccFDCNSetup Properties

Property Name	Data Type	Description - PccFDCNSetup Properties
BatchNumber	String	The Current batch number. This value is incremented by the processor after each successful settlement.
BType	Integer	The merchant's business type. **Valid values:** 0 – Retail 1 – Mail order 2 – Electronic commerce
Canceled	*Boolean*	*Used internally*
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-Up 1 – First Data IPN (Datawire – TCP/IP)
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
DID	String	The Datawire ID. The value is provided to the merchant by their Merchant Service Provider or Processing company. The DID is required to process transactions via the Internet using the Datawire network. This value will be unique for each merchant number used.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
IP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
MaxBatchSize	String	Specifies the maximum number of transactions per batch that **PC**Charge will send to the processor. If the number of transactions to be settled is greater than the number specified in this setting, **PC**Charge will split the batch into multiple batches, each containing (at most) the number transactions specified in this setting. The batches are then sent to the processor one at a time. **Example:** A merchant has 250 transaction to settle and the MaxBatchSize is set to 100. **PC**Charge will send two 100-transaction batches and one 50-transaction batch. **Max Value:** 999
PCard	Boolean	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE – Enable fields FALSE – Disable fields
SecondaryIP	*Boolean*	*Used internally*
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.

Property Name	Data Type	Description - PccFDCNSetup Properties
URL2	String	The Secondary Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
URLAddress	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.

PccFDCNSetup Methods

Method Name	Returned Value	Description - PccFDCNSetup Methods
CreateFDCNAdvanceFile	*None*	*Used internally*
CreateFDCNExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.

PccFDCSetup

This class contains FDMS Omaha / FDR extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccFDCSetup Properties

Property Name	Data Type	Description - PccFDCSetup Properties
AVS	Boolean	Flag that indicates if merchant will use address verification when processing transactions.
BType	Integer	The merchant's business type. **Valid values:** 0 – Retail 1 – Mail order 2 – Electronic commerce
Canceled	*Boolean*	*Used internally*
Device	String	Device Identification. This value is a merchant assigned code identifying the device at the merchant's location. This field is required if there is one merchant number assigned to more than one terminal at a merchant's location. Length: 4 characters
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
DID	String	The Datawire ID. The value is provided to the merchant by their Merchant Service Provider or Processing company. The DID is required to process transactions via the Internet using the Datawire network. This value will be unique for each merchant number used.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
IP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
LBatch	String	The Current batch number. This value is incremented by the processor after each successful settlement.
LItem	String	Sequence number of the last transaction transmitted
PCard	Boolean	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE – Enable fields FALSE – Disable fields
Prompt	*Boolean*	*N/A*
SecondaryIP	*Boolean*	*Used internally*
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
Sprint	Boolean	Indicates whether to use the SprintNet network when processing transactions. **Note:** The phone number to be dialed must be a SprintNet phone number. If this option is enabled and the phone number is a direct FDMS Omaha / FDR phone number, transactions will fail. Valid values: 0 – Dial direct to FDMS Omaha / FDR 1 – Use SprintNet network
SprintAdd	String	The SprintNet address to be used if Sprint = TRUE.
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE,

		TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
URL	*String*	*N/A*
URL2	*String*	*N/A*
URLAddress	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.

PccFDCSetup Methods

Method Name	Returned Value	Description - PccFDCSetup Methods
CreateFDCExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.

PccGiftCardSetup

This class contains information about the gift card company setup in the current instance of **PC**Charge. This is a Multi Use class.

PccGiftCardSetup Properties

Property Name	Data Type	Description - PccGiftCardSetup Properties
Canceled	*Boolean*	*Used internally*
Company	*Integer*	*Used internally*
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
MerchantNumber	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. **Max Length**: 32 characters. This value can be alphanumeric.
PCCBBGFSetup	*PCCBBGFSetup*	*Used internally*
PccBPASSetup	PccBPASSetup	Provides extended setup information for FDMS Atlanta
PccBPSGiftSetup	PccBPSGiftSetup	Provides extended setup information for Fifth-Third Bank-St Pete
PccDMRKSetup	PccDMRKSetup	Provides extended setup information for Datamark Gift Card
PccGSARGiftSetup	PccGSARGiftSetup	Provides extended setup information for Chase Paymentech
PccGsarSetup	PccGsarSetup	Provides extended setup information for Chase Paymentech
PccGVEXSetup	PccGVEXSetup	Provides extended setup information for Givex
PccLYNKGiftSetup	PccLYNKGiftSetup	Provides extended setup information for RBS WorldPay, Inc.
PccMELLSetup	PccMELLSetup	Provides extended setup information for Mellennia
PCCSMTSSetup	PCCSMTSSetup	Provides extended setup information for Smart Transaction Systems
PCCSPSGiftSetup	PCCSPSGiftSetup	Provides extended setup information for Secure Payment Systems
PCCSVSISetup	PCCSVSISetup	Provides extended setup information for Stored Value Systems
PccVLNKSetup	PccVLNKSetup	Provides extended setup information for ValueLink
PccVTECSetup	PccVTECSetup	Provides extended setup information for Valutec
PccWRLDSetup	PccWRLDSetup	Provides extended setup information for World
PrimaryPhone	String	The Primary number that will be used when processing transactions via dial-up modem.
Processor	String	The code for the processing company. This value can be no more than four characters and must be capitalized. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
SecondaryPhone	String	The Secondary number that will be used when processing transactions via dial-up modem.

PccGiftCardSetup Methods

Method Name	Returned Value	Description - PccGiftCardSetup Methods
CreateGCTFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
ProcessorSetup	*Object*	*Used internally*
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PccGSARGiftSetup

This class contains Chase Paymentech Gift Card extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccGSARGiftSetup Properties

Property Name	Data Type	Description - PccGSARGiftSetup Properties
BatchNumber	String	The Current batch number. This value is incremented by the processor after each successful settlement.
BType	Integer	The merchant's business type. **Valid values:**
Canceled	*Boolean*	*Used internally*
Client	String	Client Number assigned by the merchant's bank or processor. **Length:** 4 digits
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP Leased Line 2 – TCP/IP
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
EDCType	*Integer*	*Used internally*
ExpansionFactor	String	Percent by which restaurant-based transactions will be incremented during gratuity-related transactions. This setting only applies when transactions are processed using the **PC**Charge GUI.
gcBatchNumber	String	The Current Gift Card batch number. This value is incremented by the processor after each successful settlement.
gcClient	String	Client Number assigned by the merchant's bank or processor for gift card processing. **Length:** 4 digits. **Note:** This gift card specific property exists in case the Chase Paymentech credit and gift card client numbers are different.
gcLastRRefNum	String	Retrieval Reference Number. Used to identify and track the original transaction. This value is assigned by the merchant's processor. **Length:** 8 digits
gcRequireClerkID	String	Flag the indicates whether the Clerk ID is required when processing gift transactions **Valid values:** 0 – Clerk ID not required 1 – Clerk ID required
gcSeqNum	String	The Gift Card sequence number. This number is automatically incremented after every transaction.
GiftConnect	Integer	Indicates method of connection to processor when processing gift cards. **Valid values:** 0 – Dial-up 1 – TCP/IP Leased Line 2 – TCP/IP
GiftPassword	String	The Password assigned by the merchant's bank or processor for TCP/IP processing.
GiftPrimaryAuthIP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
GiftPrimaryAuthPort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
GiftUsername	String	The Username assigned by the merchant's bank or processor for TCP/IP processing.

Property Name	Data Type	Description - PccGSARGiftSetup Properties
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
Password	String	The Password assigned by the merchant's bank or processor for TCP/IP processing.
PCard	Boolean	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE – Enable fields FALSE – Disable fields
Port	Long	The system/socket port used to connect to the processor when processing via TCP/IP.
PrimaryAuthIP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
PrimaryAuthPort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
PrimarySettleIP	String	The Hostname, URL, or IP address used to connect to the processor when performing settlement via TCP/IP.
PrimarySettlePort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
RequireServerID	String	Indicates whether **PC**Charge will require a Server ID during gratuity-related restaurant transactions. Consult the section **Restaurant Transactions** for more information (see page 105). **Valid Values:** 0 – Server ID not required 1 – Server ID required
SecondaryAuthIP	String	The secondary Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
SecondaryAuthPort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
SecondaryIP	*Boolean*	*Used internally*
SecondarySettleIP	String	The secondary Hostname, URL, or IP address used to connect to the processor when performing settlement via TCP/IP.
SecondarySettlePort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
Sequence	String	This field is for identifying transactions within the batch. Assigned by **PC**Charge **Length:** 3 digits
SettlePort	Long	The system/socket port used to connect to the processor when processing via TCP/IP.
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
SettleURL	String	The Hostname, URL, or IP address used to connect to the processor when performing settlement via TCP/IP.
SIIP	Boolean	The Service Industries Incentive Program indicator flag. Used to indicate recurring payments for service industries such as insurance, telecom and utilities.
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Username	String	The Username assigned by the merchant's bank or processor for TCP/IP processing.

PccGSARGiftSetup Methods

Method Name	Returned Value	Description - PccGSARGiftSetup Methods
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.
ShowGiftCard	Boolean	Shows a GUI form that allows the end-user to enter extended configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowGiftCard.

PccGsarSetup

This class contains Chase Paymentech extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccGsarSetupProperties

Property Name	Data Type	Description - PccGsarSetup Properties
BatchNumber	String	The Current batch number. This value is incremented by the processor after each successful settlement.
BType	Integer	The merchant's business type. **Valid values:** 0 – Retail 1 – Mail order 2 – Electronic commerce 4 – Restaurant
Canceled	*Boolean*	*Used internally*
Client	String	Client Number assigned by the merchant's bank or processor. **Length:** 4 digits
Connect	String	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP Leased Line 2 – TCP/IP
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
EDCType	*Integer*	*Used internally*
ExpansionFactor	String	Percent by which restaurant-based transactions will be incremented during gratuity-related transactions. This setting only applies when transactions are processed using the **PC**Charge GUI.
gcBatchNumber	String	The Current Gift Card batch number. This value is incremented by the processor after each successful settlement.
gcClient	String	Client Number assigned by the merchant's bank or processor for gift card processing. **Length:** 4 digits. **Note:** This gift card specific property exists in case the Chase Paymentech credit and gift card client numbers are different.
gcLastRRefNum	String	Retrieval Reference Number. Used to identify and track the original transaction. This value is assigned by the merchant's processor. **Length:** 8 digits
gcRequireClerkID	String	Flag the indicates whether the Clerk ID is required when processing gift transactions **Valid values:** 0 – Clerk ID not required 1 – Clerk ID required
gcSeqNum	String	The Gift Card sequence number. This number is automatically incremented after every transaction.
GiftConnect	Integer	Indicates method of connection to processor when processing gift cards. **Valid values:** 0 – Dial-up 1 – TCP/IP Leased Line 2 – TCP/IP
GiftPassword	String	The Password assigned by the merchant's bank or processor for TCP/IP processing.
GiftPrimaryAuthIP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
GiftPrimaryAuthPort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
GiftUsername	String	The Username assigned by the merchant's bank or processor for TCP/IP processing.

Property Name	Data Type	Description - PccGsarSetup Properties
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
Password	String	The Password assigned by the merchant's bank or processor for TCP/IP processing.
PCard	Boolean	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE – Enable fields FALSE – Disable fields
Port	String	The system/socket port used to connect to the processor when processing via TCP/IP.
PrimaryAuthIP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
PrimaryAuthPort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
PrimarySettleIP	String	The Hostname, URL, or IP address used to connect to the processor when performing settlement via TCP/IP.
PrimarySettlePort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
RequireServerID	String	Indicates whether **PC**Charge will require a Server ID during gratuity-related restaurant transactions. Consult the section **Restaurant Transactions** for more information (see page 105). **Valid Values:** 0 – Server ID not required 1 – Server ID required
SecondaryAuthIP	String	The secondary Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
SecondaryAuthPort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
SecondaryIP	*Boolean*	*Used internally*
SecondarySettleIP	String	The secondary Hostname, URL, or IP address used to connect to the processor when performing settlement via TCP/IP.
SecondarySettlePort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
Sequence	String	This field is for identifying transactions within the batch. Assigned by **PC**Charge **Length:** 3 digits
SettlePort	Long	The system/socket port used to connect to the processor when processing via TCP/IP.
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
SettleURL	String	The Hostname, URL, or IP address used to connect to the processor when performing settlement via TCP/IP.
SIIP	Boolean	The Service Industries Incentive Program indicator flag. Used to indicate recurring payments for service industries such as insurance, telecom and utilities.
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Username	String	The Username assigned by the merchant's bank or processor for TCP/IP processing.

PccGsarSetup Methods

Method Name	Returned Value	Description - PccGsarSetup Methods
CreateGSARExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.
ShowGiftCard	Boolean	Shows a GUI form that allows the end-user to enter extended configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowGiftCard.

PccGVEXSetup

This class contains Givex extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccGVEXetup Properties

Property Name	Data Type	Description - PccGVEXetup Properties
Canceled	*Boolean*	*Used internally*
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
Language	String	Language Code. **Valid values:** 0 – English (default)
Pin	String	Personal identification number issued to merchant by processor
Port	String	The system/socket port used to connect to the processor when processing via TCP/IP.
Serial	*String*	*Used internally*
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
TrackPoints	Boolean	Flag that indicates whether to use Points transactions. The Points service is used for tracking points in a points program. The merchant sends information on the card used and the amount of the transaction. This information is stored and is made available later.
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
UseUnits	Boolean	Flag that indicates whether to use units in conjunction with Points transactions. The points value is entered by the merchant and is tracked by the Givex system for the points program. Points could be the total number of items or some calculated unit value or anything else defined by the user. This can be used later when calculating the value to add to the card.

PccGVEXSetup Methods

Method Name	Returned Value	Description - PccGVEXSetup Methods
CreateGVEXExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowGC	Boolean	Shows a GUI form that allows the end-user to enter extended or advanced Gift Card configuration information such as the business type, communication method, or other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowGC.

PccHPTSSetup

This class contains Heartland Payment Systems extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccHPTSSetup Properties

Property Name	Data Type	Description - PccHPTSSetup Properties
ABA	String	The ABA number identifies the merchant to a direct debit switch. Applicable for Debit/EBT transactions. This value is provided to the merchant by their Merchant Services Provider or Processor. **Length:** 9 digits
Agent	String	The Agent Bank Number issued by the merchant's bank or processor. This parameter is used to identify a specific agent entity of the member bank or processor. **Length:** 6 digits. **Example:** 111111
AuthPrimaryPort	*String*	*N/A*
AuthPrimaryURL	*String*	*N/A*
Batch	String	The Current batch number. This value is incremented by the processor after each successful settlement.
Bin	String	The Bank Identification Number issued by the merchant's bank or processor. The BIN identifies the Merchant Service Provider that signed up the merchant. **Length:** 6 digits. **Example:** 999995
BType	Integer	The merchant's business type. **Valid values:** 0 – Retail 1 – Mail order 4 – Restaurant
Canceled	*Boolean*	*Used internally*
Category	String	The Merchant Category Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's industry classification. **Length:** 4 digits. **Example:** 5999
Chain	String	The Agent Chain Number issued by the merchant's bank or processor. This parameter is used to identify a specific chain of an agent organization. **Length:** 6 digits. **Example:** 000000
City	String	The merchant's postal/zip code as assigned by the merchant's bank or processor. **Length:** 5 or 9 digits. **Example:** 314193262
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
Country	String	The Country Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's country location. **Length:** 3 digits. **Valid Value:** 840 – U.S.
CreditTermType	*CreditTerminalType*	*N/A*
CSPhone	String	The Merchant Local Telephone Number. **Length:** 11 characters. **Format:** NNN-nnnnnnn where NNN is the area code and nnnnnnn is the telephone number. The hyphen is required. **Example:** 800-7259264
CurrencyCode	String	The Currency Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's settlement currency. **Length:** 3 digits. **Valid Value:** 840 – U.S. Dollars
DebitBIN	String	The Bank Identification Number issued by the merchant's bank or processor. The BIN identifies the Merchant Service Provider that signed up the merchant. **Length:** 6 digits. **Example:** 999995
DebitTermType	*CreditTerminalType*	*N/A*

Property Name	Data Type	Description - PccHPTSSetup Properties
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
DID	*String*	*N/A*
ebtABA	*String*	*N/A*
ebtBIN	*String*	*N/A*
ebtFCSID	*String*	*N/A*
ebtReimbursement	*String*	*N/A*
ebtSettleAgent	*String*	*N/A*
ebtSharingGroup	*String*	*N/A*
EBTTermType	*DebitTerminalType*	*N/A*
ExpansionFactor	String	Percent by which restaurant-based transactions will be incremented during gratuity-related transactions. This setting only applies when transactions are processed using the **PC**Charge GUI.
ExpressPay	Boolean	Express Pay flag. This setting only applies if the Business Type is set to retail. If this flag is set to TRUE and the amount of the transaction is less than the FloorLimit amount, **PC**Charge will not authorize the transaction—it will only place the transaction in the open batch. Express Pay is usually used in a quick service environment with small ticket items. **Note:** Using Express Pay will increase transaction processing costs.
FloorLimit	String	The floor limit amount. This setting is only applicable if the Business Type is set to retail. The floor limit is the maximum transaction amount that will be accepted by **PC**Charge when processing an Express Pay transaction
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
IP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Language	String	Language indicator. **Length:** 2 digits. **Valid Values:** 00 – English 01 – Spanish 02 – Portuguese 03 – Reserved for Irish 04 – Reserved for French 05 – Reserved for German 06 – Reserved for Italian 07 – Reserved for Dutch
Location	String	The Merchant Location Number provides additional information on the location of the merchant. **Length:** 5 characters. **Example:** 00001 **Note:** This value should be 00001 unless otherwise specified by the merchant's bank or processor.
MaxBatchSize	String	Specifies the maximum number of transactions per batch that **PC**Charge will send to the processor. If the number of transactions to be settled is greater than the number specified in this setting, **PC**Charge will split the batch into multiple batches, each containing (at most) the number transactions specified in this setting. The batches are then sent to the processor one at a time. **Example:** A merchant has 250 transaction to settle and the MaxBatchSize is set to 100. **PC**Charge will send two 100-transaction batches and one 50-transaction batch. **Max Value:** 999
MCReversal	*Boolean*	*N/A*
PCard	*Boolean*	*N/A*
Phone1	String	Primary phone number for settlement. If this value is set and Dial-up modem is the communication method, **PC**Charge will attempt to settle transactions using this phone number rather than the authorization phone number.

Property Name	Data Type	Description - PccHPTSSetup Properties
Phone2	String	Secondary phone number for settlement. If this value is set and Dial-up modem is the communication method, **PC**Charge will attempt to settle transactions using this phone number rather than the authorization phone number.
Port	String	The system/socket port used to connect to the processor when processing via TCP/IP.
Reimbursement	String	The Reimbursement Attribute designates the Reimbursement Fee applicable to a transaction. Applicable for Debit/EBT transactions. This value is assigned to the merchant by their Merchant Services Provider or Processor. **Length:** 1 character.
RequireServerID	String	Indicates whether **PC**Charge will require a Server ID during gratuity-related restaurant transactions. Consult the section **Restaurant Transactions** for more information (see page 105). **Valid Values:** 0 – Server ID not required 1 – Server ID required
SettleAgent	String	The Merchant Settlement Agent Number identifies the merchant settling agent. Applicable for Debit/EBT transactions. This value is provided to the merchant by their Merchant Services Provider or Processor. **Length:** 4 characters
SettleConnect	*String*	*N/A*
SettleDialBackUp	*String*	*N/A*
SettleMaxBlockCount	*String*	*N/A*
SettleMaxBlockSize	*String*	*N/A*
SettlePrimaryPort	*String*	*N/A*
SettlePrimaryURL	*String*	*N/A*
SettleTCP	*Boolean*	*N/A*
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
SharingGroup	String	The Sharing Group contains a listing of direct debit and EBT networks that may be accessed. This value is provided to the merchant by their Merchant Services Provider or Processor. The values must correspond to one of the Visa assigned direct debit network types. This data is part of the VisaNet direct debit data. **Length:** 1 to 30 characters.
Store	String	The Store Number assigned by the merchant's bank or processor. This parameter is used to identify a specific merchant store location. **Length:** 4 digits. **Example:** 0011
Terminal	String	The Terminal Number assigned by the merchant's bank or processor. This parameter is used to identify a specific store terminal. **Length:** 4 digits. **Example:** 9911.
TID	*String*	*N/A*
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
TimeZone	String	The Time Zone Differential as assigned by the merchant's bank or processor. This value provides the standard local time zone differential from Greenwich Mean Time (GMT). **Length:** 3 digits. **Valid Values:** 705 – Eastern 706 – Central 707 – Mountain 708 – Pacific **Note:** Replace the leading 7 with a 1 if Daylight Savings is not observed. **Example:** 107 – Arizona
URL	*String*	*N/A*
URL2	*String*	*N/A*
VSReversal	*Boolean*	*N/A*

PccHPTSSetup Methods

Method Name	Returned Value	Description - PccHPTSSetup Methods
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.
ShowDebit	Boolean	Shows a GUI form that allows the end-user to enter Debit configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowDebit.
ShowEBT	Boolean	Shows a GUI form that allows the end-user to enter EBT configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowEBT.

PccImpSetup

This class contains information about the Import File setup in the current instance of **PC**Charge. This is a Multi Use class.

PccImpSetup Properties

Property Name	Data Type	Description - PccImpSetup Properties
APPAVS	Boolean	Indicates whether to include AVS responses in .app files
BADAVS	Boolean	Indicates whether to include AVS responses in .bad files
Canceled	*Boolean*	*Used internally*
CustName	Boolean	Indicates whether to include customer name in import files
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
PCardInfo	Boolean	Indicates whether to include commercial / purchasing card information in import files

PccImpSetup Methods

Method Name	Returned Value	Description - PccImpSetup Methods
CreateImportFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PccLYNKGiftSetup

PccLYNKGiftSetup Properties

Property Name	Data Type	Description - PccLYNKGiftSetup Properties
Bank	String	The Agent Bank Number issued by the merchant's bank or processor. This parameter is used to identify a specific agent entity of the member bank or processor. **Length:** 6 digits. **Example:** 111111
Batch	String	The Current batch number. This value is incremented by the processor after each successful settlement.
Bin	String	The Bank Identification Number issued by the merchant's bank or processor. The BIN identifies the Merchant Service Provider that signed up the merchant. **Length:** 6 digits. **Example:** 999995
BType	*String*	*N/A*
Canceled	*Boolean*	*Used internally*
Category	String	The Merchant Category Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's industry classification. **Length:** 4 digits. **Example:** 5999
Chain	String	The Agent Chain Number issued by the merchant's bank or processor. This parameter is used to identify a specific chain of an agent organization. **Length:** 6 digits. **Example:** 000000
City	String	The merchant's postal/zip code as assigned by the merchant's bank or processor. **Length:** 5 or 9 digits. **Example:** 314193262
CompanyCity	String	The merchant's postal/zip code as assigned by the merchant's bank or processor. **Length:** 5 or 9 digits. **Example:** 314193262
CompanyName	String	The merchant's name as assigned by the merchant's bank or processor.
CompanyState	String	The merchant's state abbreviation as assigned by the merchant's bank or processor.
Connect	*Integer*	*N/A*
Country	String	The Country Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's country location. **Length:** 3 digits. **Valid Value:** 840 – U.S.
CSPhone	String	The Merchant Local Telephone Number. **Length:** 11 characters. **Format:** NNN-nnnnnnn where NNN is the area code and nnnnnnn is the telephone number. The hyphen is required. **Example:** 800-7259264
CurrencyCode	String	The Currency Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's settlement currency. **Length:** 3 digits. **Valid Value:** 840 – U.S. Dollars
DialBackup	*Boolean*	*N/A*
Expansion	*String*	*N/A*
GiftConnect	*Integer*	*N/A*
GiftDialBackup	*Boolean*	*N/A*
GiftPort	*String*	*N/A*
GiftSettleTimeOut	*String*	*N/A*
GiftTimeOut	*Integer*	*N/A*
GiftURL	*String*	*N/A*
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.

Property Name	Data Type	Description - PccLYNKGiftSetup Properties
Language	String	Language indicator. **Length:** 2 digits. **Valid Values:** 00 – English 01 – Spanish 02 – Portuguese 03 – Reserved for Irish 04 – Reserved for French 05 – Reserved for German 06 – Reserved for Italian 07 – Reserved for Dutch
Location	*String*	*N/A*
Phone1	*String*	*N/A*
Phone2	*String*	*N/A*
Port	*String*	*N/A*
SettleTimeOut	*String*	*N/A*
Store	String	The Store Number assigned by the merchant's bank or processor. This parameter is used to identify a specific merchant store location. **Length:** 4 digits. **Example:** 0011
Terminal	String	The Terminal Number assigned by the merchant's bank or processor. This parameter is used to identify a specific store terminal. **Length:** 4 digits. **Example:** 9911.
TimeOut	*Integer*	*N/A*
TimeZone	String	The Time Zone Differential as assigned by the merchant's bank or processor. This value provides the standard local time zone differential from Greenwich Mean Time (GMT). **Length:** 3 digits. **Valid Values:** 705 – Eastern 706 – Central 707 – Mountain 708 – Pacific **Note:** Replace the leading 7 with a 1 if Daylight Savings is not observed. **Example:** 107 – Arizona
URL	*String*	*N/A*

PccLYNKGiftSetup Methods

Method Name	Returned Value	Description - PccLYNKGiftSetup Methods
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.
ValidateExt	*Boolean*	*N/A*

PccLYNKSetup

This class contains RBS WorldPay extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccLYNKSetup Properties

Property Name	Data Type	Description - PccLYNKSetup Properties
ABA	String	The ABA number identifies the merchant to a direct debit switch. Applicable for Debit/EBT transactions. This value is provided to the merchant by their Merchant Services Provider or Processor. **Length:** 9 digits
Agent	String	The Agent Bank Number issued by the merchant's bank or processor. This parameter is used to identify a specific agent entity of the member bank or processor. **Length:** 6 digits. **Example:** 111111
AuthPrimaryPort	*String*	*N/A*
AuthPrimaryURL	*String*	*N/A*
Batch	String	The Current batch number. This value is incremented by the processor after each successful settlement.
Bin	String	The Bank Identification Number issued by the merchant's bank or processor. The BIN identifies the Merchant Service Provider that signed up the merchant. **Length:** 6 digits. **Example:** 999995
BType	Integer	The merchant's business type. **Valid values:** 0 – Retail 1 – Mail order 4 – Restaurant
Canceled	*Boolean*	*Used internally*
Category	String	The Merchant Category Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's industry classification. **Length:** 4 digits. **Example:** 5999
Chain	String	The Agent Chain Number issued by the merchant's bank or processor. This parameter is used to identify a specific chain of an agent organization. **Length:** 6 digits. **Example:** 000000
City	String	The merchant's postal/zip code as assigned by the merchant's bank or processor. **Length:** 5 or 9 digits. **Example:** 314193262
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – ISDN 2 – TCP/IP
Country	String	The Country Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's country location. **Length:** 3 digits. **Valid Value:** 840 – U.S.
CreditTermType	*CreditTerminalType*	*N/A*
CSPhone	String	The Merchant Local Telephone Number. **Length:** 11 characters. **Format:** NNN-nnnnnnn where NNN is the area code and nnnnnnn is the telephone number. The hyphen is required. **Example:** 800-7259264
CurrencyCode	String	The Currency Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's settlement currency. **Length:** 3 digits. **Valid Value:** 840 – U.S. Dollars
DebitBIN	String	The Bank Identification Number issued by the merchant's bank or processor. The BIN identifies the Merchant Service Provider that signed up the merchant. **Length:** 6 digits. **Example:** 999995
DebitTermType	*CreditTerminalType*	*N/A*

Property Name	Data Type	Description - PccLYNKSetup Properties
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
DID	*String*	*N/A*
ebtABA	*String*	*N/A*
ebtBIN	*String*	*N/A*
ebtFCSID	*String*	*N/A*
ebtReimbursement	*String*	*N/A*
ebtSettleAgent	*String*	*N/A*
ebtSharingGroup	*String*	*N/A*
EBTTermType	*DebitTerminalType*	*N/A*
ExpansionFactor	String	Percent by which restaurant-based transactions will be incremented during gratuity-related transactions. This setting only applies when transactions are processed using the **PC**Charge GUI.
ExpressPay	Boolean	Express Pay flag. This setting only applies if the Business Type is set to retail. If this flag is set to TRUE and the amount of the transaction is less than the FloorLimit amount, **PC**Charge will not authorize the transaction—it will only place the transaction in the open batch. Express Pay is usually used in a quick service environment with small ticket items. **Note:** Using Express Pay will increase transaction processing costs.
FloorLimit	String	The floor limit amount. This setting is only applicable if the Business Type is set to retail. The floor limit is the maximum transaction amount that will be accepted by **PC**Charge when processing an Express Pay transaction
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
IP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Language	String	Language indicator. **Length:** 2 digits. **Valid Values:** 00 – English 01 – Spanish 02 – Portuguese 03 – Reserved for Irish 04 – Reserved for French 05 – Reserved for German 06 – Reserved for Italian 07 – Reserved for Dutch
Location	String	The Merchant Location Number provides additional information on the location of the merchant. **Length:** 5 characters. **Example:** 00001 **Note:** This value should be 00001 unless otherwise specified by the merchant's bank or processor.
MaxBatchSize	String	Specifies the maximum number of transactions per batch that **PC**Charge will send to the processor. If the number of transactions to be settled is greater than the number specified in this setting, **PC**Charge will split the batch into multiple batches, each containing (at most) the number transactions specified in this setting. The batches are then sent to the processor one at a time. **Example:** A merchant has 250 transaction to settle and the MaxBatchSize is set to 100. **PC**Charge will send two 100-transaction batches and one 50-transaction batch. **Max Value:** 999
MCReversal	*Boolean*	*N/A*
PCard	Boolean	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE – Enable fields FALSE – Disable fields

Property Name	Data Type	Description - PccLYNKSetup Properties
PccVisaSetup_ebtFCSID	*String*	*Used internally*
Phone1	String	Primary phone number for settlement. If this value is set and Dial-up modem is the communication method, **PC**Charge will attempt to settle transactions using this phone number rather than the authorization phone number.
Phone2	String	Secondary phone number for settlement. If this value is set and Dial-up modem is the communication method, **PC**Charge will attempt to settle transactions using this phone number rather than the authorization phone number.
Port	String	The system/socket port used to connect to the processor when processing via TCP/IP.
Reimbursement	String	The Reimbursement Attribute designates the Reimbursement Fee applicable to a transaction. Applicable for Debit/EBT transactions. This value is assigned to the merchant by their Merchant Services Provider or Processor. **Length:** 1 character.
RequireServerID	String	Indicates whether **PC**Charge will require a Server ID during gratuity-related restaurant transactions. Consult the section **Restaurant Transactions** for more information (see page 105). **Valid Values:** 0 – Server ID not required 1 – Server ID required
SettleAgent	String	The Merchant Settlement Agent Number identifies the merchant settling agent. Applicable for Debit/EBT transactions. This value is provided to the merchant by their Merchant Services Provider or Processor. **Length:** 4 characters
SettleConnect	*String*	*N/A*
SettleDialBackUp	*String*	*N/A*
SettleMaxBlockCount	*String*	*N/A*
SettleMaxBlockSize	*String*	*N/A*
SettlePrimaryPort	*String*	*N/A*
SettlePrimaryURL	*String*	*N/A*
SettleTCP	*Boolean*	*N/A*
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
SharingGroup	String	The Sharing Group contains a listing of direct debit and EBT networks that may be accessed. This value is provided to the merchant by their Merchant Services Provider or Processor. The values must correspond to one of the Visa assigned direct debit network types. This data is part of the VisaNet direct debit data. **Length:** 1 to 30 characters.
Store	String	The Store Number assigned by the merchant's bank or processor. This parameter is used to identify a specific merchant store location. **Length:** 4 digits. **Example:** 0011
Terminal	String	The Terminal Number assigned by the merchant's bank or processor. This parameter is used to identify a specific store terminal. **Length:** 4 digits. **Example:** 9911.
TID	*String*	*N/A*
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
TimeZone	String	The Time Zone Differential as assigned by the merchant's bank or processor. This value provides the standard local time zone differential from Greenwich Mean Time (GMT). **Length:** 3 digits. **Valid Values:** 705 – Eastern 706 – Central 707 – Mountain 708 – Pacific **Note:** Replace the leading 7 with a 1 if Daylight Savings is not observed. **Example:** 107 – Arizona
URL	*String*	*N/A*
URL2	*String*	*N/A*

Property Name	Data Type	Description - PccLYNKSetup Properties
VSReversal	Boolean	Indicates whether VISA reversals will be processed.

PccLYNKSetup Methods

Method Name	Returned Value	Description - PccLYNKSetup Methods
CreateLYNKAdvanceFile	*None*	*Used internally*
CreateLynkDebitExtFile	*None*	*Used internally*
CreateLynkExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
PccVisaSetup_CreateVisaDebitExtFile	*None*	*Used internally*
PccVisaSetup_CreateVisaExtFile	*None*	*Used internally*
PccVisaSetup_CreateVisaIPNExtFile	*None*	*Used internally*
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.
ShowDebit	Boolean	Shows a GUI form that allows the end-user to enter Debit configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowDebit.
ShowEBT	Boolean	Shows a GUI form that allows the end-user to enter EBT configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowEBT.

PccMELLSetup

This class contains Mellennia extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccMELLSetup Properties

Property Name	Data Type	Description - PccMELLSetup Properties
BType	String	The merchant's business type. **Valid values:** 0 – Restaurant 1 – Other
Canceled	*Boolean*	*Used internally*
Connect	*Integer*	*N/A*
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
RequireClerkID	String	Indicates whether the Server ID needs to be filled for a transaction to completed. **Valid Values:** 0 – Not required 1 – Required

PccMELLSetup Methods

Method Name	Returned Value	Description - PccMELLSetup Methods
CreateMELLExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowGC	Boolean	Shows a GUI form that allows the end-user to enter extended or advanced Gift Card configuration information such as the business type, communication method, or other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowGC.

PccModem

This class allows modem configuration.

Note: This class can only be accessed through the `PccActiveCharge` class. Restricting its instantiation eliminates the possibility of multiple instancing. See the following code fragment.

```
'In Declare section:

   Dim clsActiveCharge As ActiveCharge.PccActiveCharge
   Dim WithEvents clsModem As ActiveCharge.PccModem

'In Implementation section:

   Set clsActiveCharge = New ActiveCharge.PccActiveCharge
   Set clsModem = clsActiveCharge.PccModem
   With clsModem
     .Load
     .Validate ("AT")
   End With

‘Once OnUpdate event fires:

   MsgBox Status
```

This is a Public Not Creatable class.

PccModem Properties

Property Name	Data Type	Description - PccModem Properties
Baud	String	The baud rate (300, 1200).
Canceled	*Boolean*	*Used internally*
CFGFile	Collection	Allows viewing the modem strings contained in the MODEM.CFG file. Requires the index number of the modem string as a parameter.
DialBackup	*Boolean*	*N/A*
DialMethod	String	The dial method. **Valid values:** T – Tone P – Pulse
DialPrefix	String	The dial prefix. **Example:** ,9
DialString	*String*	*Used internally*
EncLogFile	Boolean	Flag that indicates whether to encrypt the communication log file. This option should be set to TRUE if providing the communication log file to VeriFone, Inc.’s technical support department.
Index	*Integer*	*N/A*
InitString	String	The modem initialization string
LogFile	Boolean	Flag that indicates whether to create a communications log file.
LogFileBacking	Boolean	Flag that indicates whether to create a backup of the communications log file.
ModemType	String	Name of the modem initialization string
Port	Integer	Specifies COM port used for modem connection
PurgeSize	Long	Specifies the size limit of the communications log to begin purging. **Format:** KB
RespDelay	String	The Response delay used when dialing. This value should be increased if experiencing problems with WinModems. **Format:** seconds
TAPIFriendlyName**	String	Name of modem as listed by TAPI drivers
TAPIIndex**	Integer	Which modem to use as specified by local machine TAPI modem index

Property Name	Data Type	Description - PccModem Properties
TAPIUseDefaultSettings	Boolean	Flag that indicates to use the Windows modem settings
UseISDN	*Boolean*	*N/A*
UseTAPI	Boolean	Indicates whether to use TAPI for modem connection
WinModem	Boolean	Flag that indicates whether to re-initialize the modem before each transaction. Activating this setting fixes issues with some WinModems.

** To use these properties, UseTapi must be set to TRUE.

PccModem Methods

Method Name	Returned Value	Description - PccModem Methods
Cancel	Boolean	The Cancel method attempts to cancel the test transaction in progress. Calling the Cancel method does not guarantee that the test transaction will be canceled; it simply attempts to cancel the test transaction.
CreateModemFile	*None*	*Used internally*
GetBaud	*String*	*Used internally*
GetOK	*Boolean*	*Used internally*
GetPort	*Integer*	*Used internally*
InitModem	*Boolean*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
PccTest	Boolean	PccTest runs a transaction test using the currently selected merchant number and communication type. PccTest can test either TCP/IP or dial-up communication—it tests whichever communication method is selected in the extended configuration of the currently selected merchant number.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
SetBaud	*Integer*	*Used internally*
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
TAPIGetModemCount	*Integer*	*Used internally*
TAPIGetModemIndex	*Integer*	*Used internally*
TAPIGetModemName	*String*	*Used internally*
Validate	Boolean	Validates a modem initialization string. This method requires a modem initialization string to be passed as a parameter. This method passes each portion of the initialization string to the modem. If the modem responds to a portion with an OK response, that portion is kept. If the modem responds to a portion with an ERROR response, that portion is discarded. The method then constructs a new modem initialization string with the remaining kept portions and places that string in the InitString property. TRUE is returned if the operation is successful, FALSE otherwise.

PccModem Events

Method Name	Returned Value	Description - PccModem Methods
OnUpdate	Boolean	The OnUpdate event will fire when the PccModem class provides status messages. Once the ActionUpdate event has fired the Status value is set with the current status message provided by the PccModem class.

PccNBSetup

This class contains FDMS South / NaBanco merchant number extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccNBSetup Properties

Property Name	Data Type	Description - PccNBSetup Properties
Address1	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Address2	String	The secondary Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
AMEX	String	American Express service establishment number. **Length:** 9 or10 digits
BatchNumber	String	The Current batch number. This value is incremented by the processor after each successful settlement.
BatchSerial	String	Unique Terminal Serial Number for the current merchant number. **Length:** 2 digits.
BType	Integer	The merchant's business type. **Valid values:** 0 – Retail 1 – Mail Order 4 – Restaurant
Canceled	*Boolean*	*Used internally*
Carte	String	Carte Blanche service establishment number. **Length:** 10 digits
Category	String	The Merchant Category Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's industry classification. **Length:** 4 digits. **Example:** 5999
Connect	String	Indicates method of connection to processor. **Valid values:** 0 = dial-up 1 – TCP/IP Lease Line 2 = First Data IPN (Datawire – TCP/IP)
Country	String	The Country Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's country location. **Length:** 3 digits. **Valid Value:** 840 – U.S.
DebitDID	String	The Datawire ID. The value is provided to the merchant by their Merchant Service Provider or Processing company. The DID is required to process transactions via the Internet using the Datawire network. This value will be unique for each merchant number used.
DebitSalesCount	String	Count of Debit Sales
DebitTotalSalesAmt	String	Total amount of Debit Sales
DialBackup	String	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
DID	String	The Datawire ID. The value is provided to the merchant by their Merchant Service Provider or Processing company. The DID is required to process transactions via the Internet using the Datawire network. This value will be unique for each merchant number used.
Diners	String	Diner's Club service establishment number. **Length:** 10 digits
Discover	String	Discover service establishment number. **Length:** 10 digits
ExpansionFactor	String	Percent by which restaurant-based transactions will be incremented during gratuity-related transactions. This setting only applies when transactions are processed using the **PC**Charge GUI.

Property Name	Data Type	Description - PccNBSetup Properties
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
IP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
JAL	String	JAL service establishment number. **Length:** 10 digits
JCB	String	JCB service establishment number. **Length:** 10 digits
Mastercard	*Boolean*	*N/A*
MaxBatchSize	String	Specifies the maximum number of transactions per batch that **PC**Charge will send to the processor. If the number of transactions to be settled is greater than the number specified in this setting, **PC**Charge will split the batch into multiple batches, each containing (at most) the number transactions specified in this setting. The batches are then sent to the processor one at a time. **Example:** A merchant has 250 transaction to settle and the MaxBatchSize is set to 100. **PC**Charge will send two 100-transaction batches and one 50-transaction batch. **Max Value:** 999
MCI	Boolean	Indicates whether merchant is using the MCI dialing option rather than a direct connection to the processor.
PCard	String	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE – Enable fields FALSE – Disable fields
Port	String	The system/socket port used to connect to the processor when processing via TCP/IP.
Port1	String	The system/socket port used to connect to the processor when processing via TCP/IP.
Port2	String	The secondary system/socket port used to connect to the processor when processing via TCP/IP.
ProcessingDebit	*Boolean*	*Used internally*
Qual	String	The Qual code identifies the merchant's plan code and types cards that are accepted. This value is assigned by the merchant's bank or processor. **Length:** 6 digits.
ReceiptNumber	*String*	*Used internally*
RequireServerID	String	Indicates whether **PC**Charge will require a Server ID during gratuity-related restaurant transactions. Consult the section **Restaurant Transactions** for more information (see page 105). **Valid Values:** 0 – Server ID not required 1 – Server ID required
SecondaryIP	*Boolean*	*Used internally*
Serial	String	Unique Terminal Serial Number for the current merchant number. **Length:** 2 digits.
SerialNumber	*String*	*N/A*
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
SICCode	String	The Merchant Category Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's industry classification. **Length:** 4 digits. **Example:** 5999
State	String	The State Code This value is provided to the merchant by their Merchant Service Provider or Processing company. This value identifies the merchant's. Length: 2 digits
TermID	*String*	*Used internally*
Timeout	String	Timeout value for leased-line connectivity. **Valid range:** 1-45 (seconds)

Property Name	Data Type	Description - PccNBSetup Properties
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Visa	*Boolean*	*N/A*
Zip	String	The merchant's zip code

PccNBSetup Methods

Method Name	Returned Value	Description - PccNBSetup Methods
CreateNBExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowDebit	Boolean	Shows a GUI form that allows the end-user to enter Debit configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowDebit.

PccNBSSetup

This class contains National Bankcard Services merchant number extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccNBSSetup Properties

Property Name	Data Type	Description - PccNBSSetup Properties
AcceptVMCFleet	String	The setting to accept Visa/MC Fleet Cards. **Valid Values:** 0 = No 1 = Yes
BatchNumber	String	The batch number.
BType	String	The business type. Currently, NBS only supports retail. **Valid Values:** 0 = Retail
CATType	String	The CAT setting. **Valid Values:** 0 = No 1 = Yes
Connect	String	The connection type. **Valid Values:** 0 = Dial 1 = Datawire
DebitCATType	String	The debit CAT setting. **Valid Values:** 0 = No 1 = Yes
DID	String	The Datawire ID.
DialBackup	String	The dial backup indicator. **Valid Values:** 0 = No 1 = Yes
EnableProductDetail	String	The setting to enabled Product Details. Note: Must be checked if Fleet Cards will be accepted. **Valid Values:** 0 = No 1 = Yes
LoggedOn	String	Indicator used for logging into the NBS system.
MaxBatchSize	String	The maximum batch size.
PCard	String	The PCard (commercial card) acceptance setting. **Valid Values:** 0 = No 1 = Yes
RegistrationURL	String	The URL for Datawire registration. This is defaulted in PCCharge.
SecondaryURL	String	The secondary Datawire URL. This is defaulted in PCCharge.
SettleTimeout	String	The settlement timeout value. The default value is 240 seconds.
TimeOut	String	The authorization timeout value. The default value is 60 seconds.
TimeZone	String	The time zone. This is the offset from GMT. This is in the format HHMM for the first four digits, and the last digit is the Daylight Savings Time indicator (0 = No, 1 = Yes).
URL	String	The primary Datawire URL. This is defaulted in PCCharge.

PccNBSSetup Methods

Method Name	Returned Value	Description - PccNBSSetup Methods
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowDebit	Boolean	Shows a GUI form that allows the end-user to enter Debit configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowDebit.
GetDefaultURL	String	Returns the default Datawire URL.
GetDefaultRegistrationURL	String	Returns the default registration URL.

PccNDCSetup

This class contains Global Payments-East extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccNDCSetup Properties

Property Name	Data Type	Description - PccNDCSetup Properties
BankID	String	The Bank ID assigned by the merchant's bank or processor. **Length:** 6 digits.
BatchAmount	String	Balance of current batch
BatchNumber	String	The Current batch number. This value is incremented by the processor after each successful settlement.
BType	Integer	The merchant's business type. **Valid values:** 0 – Retail 1 – Mail Order 2 – E-Commerce 4 – Restaurant
Canceled	*Boolean*	*Used internally*
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
DataPack	Boolean	Flag that indicates if the DataPac Communications Network will be used to connect to the processor. Set to TRUE if connecting to the processor in Canada.
DebitPurchaseAmount	String	Amount of debit purchases.
DebitPurchaseCount	String	Number of debit purchases.
DebitReturnAmount	String	Amount of debit returns.
DebitReturnCount	String	Number of debit returns.
Duplicate	String	Used only for Canadian Debit Transactions. Determines if duplicate transactions are allowed with processing Canadian debit transactions. See the section **Canadian (Interac) Debit Transactions** (see page 113) for more information. **Valid Values:** 0 – Do not allow Duplicate transactions 1 – Allow Duplicate transactions
EDCType	*String*	*Used internally*
ExpansionFactor	String	Percent by which restaurant-based transactions will be incremented during gratuity-related transactions. This setting only applies when transactions are processed using the **PC**Charge GUI.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
ItemCount	String	Number of transactions in current batch
Location	String	The Merchant Location Number provides additional information on the location of the merchant. **Length:** 5 characters. **Example:** 00001 **Note:** This value should be 00001 unless otherwise specified by the merchant's bank or processor.
Manual	Boolean	Indicates whether batches will be manually opened and closed
MerchantType	*String*	*N/A*
PCard	Boolean	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE – Enable fields FALSE – Disable fields
PurchaseAmt	String	Amount of sale or post-authorization transactions
PurchaseCount	String	Number of sale or post-authorization transactions

Property Name	Data Type	Description - PccNDCSetup Properties
RequireServerID	String	Indicates whether **PC**Charge will require a Server ID during gratuity-related restaurant transactions. Consult the section **Restaurant Transactions** for more information (see page 105). **Valid Values:** 0 – Server ID not required 1 – Server ID required
ReturnAmt	String	Amount of credits
ReturnCount	String	Number of credits
SecondaryIP	*Boolean*	*Used internally*
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
Sprint	Boolean	Flag that indicates if SprintNet will be used to connect to the processor. Set to TRUE if connecting to SprintNet.
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
TipPercent	String	Used only for Canadian Debit Transactions. The percentage of tip that will be displayed on the PINpad. See the section **Canadian (Interac) Debit Transactions** (see page 113) for more information.
TipType	String	Used only for Canadian Debit Transactions. The type of tip calculation that will occur. See the section **Canadian (Interac) Debit Transactions** (see page 113) for more information.
TranItem	String	Current transaction item number
URL	*String*	*N/A*
URL2	*String*	*N/A*
URLAddress	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.

PccNDCSetup Methods

Method Name	Returned Value	Description - PccNDCSetup Methods
CreateNDCExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.

Method Name	Returned Value	Description - PccNDCSetup Methods
ShowDebit	Boolean	Shows a GUI form that allows the end-user to enter Debit configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowDebit.

PccNovaSetup

This class contains Elavon (NOVA) extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccNovaSetup Properties

Property Name	Data Type	Description - PccNovaSetup Properties
Balance	String	Balance of current batch
BankID	String	The Bank ID assigned by the merchant's bank or processor. **Length:** 6 digits.
BatchNumber	String	The Current batch number. This value is incremented by the processor after each successful settlement.
BType	Integer	The merchant's business type. **Valid values:** 0 – Retail 1 – Mail order 2 – Electronic Commerce
Canceled	*Boolean*	*Used internally*
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
Counter	String	The record count (The number of transactions in the batch)
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
EDCType	Integer	Flag that indicates whether processing will occur on Elavon's (NOVA) host-based system or the terminal-based system. This setting must match what Elavon (NOVA) has set up for the merchant (i.e. a Elavon (NOVA) terminal-based merchant account will not work on the host-based system) **Valid values:** 0 – Host 1 – Terminal
HostName	String	The Hostname, URL, or IP address used to connect to Elavon's (NOVA) Test System when processing via TCP/IP.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
IPDebug	*Boolean*	*N/A*
IPTest	Boolean	Flag that indicates whether to send transactions to Elavon's (NOVA) test system or to the live system. TRUE – Send to Elavon's (NOVA) Test System FALSE – Send to Elavon's (NOVA) live system
Manual	Boolean	Indicates whether batches will be manually opened and closed
PCard	Boolean	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE – Enable fields FALSE – Disable fields

Sequence	*String*	*N/A*
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PCC**harge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds

PccNovaSetup Methods

Method Name	Returned Value	Description - PccNovaSetup Methods
CreateNOVAExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PccNPCSetup

This class contains National Processing Company extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccNPCSetup Properties

Property Name	Data Type	Description - PccNPCSetup Properties
ABA	String	The ABA number identifies the merchant to a direct debit switch. Applicable for Debit/EBT transactions. This value is provided to the merchant by their Merchant Services Provider or Processor. **Length:** 9 digits
Agent	String	The Agent Bank Number issued by the merchant's bank or processor. This parameter is used to identify a specific agent entity of the member bank or processor. **Length:** 6 digits. **Example:** 111111
AuthPrimaryPort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
AuthPrimaryURL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Batch	String	The Current batch number. This value is incremented by the processor after each successful settlement.
Bin	String	The Bank Identification Number issued by the merchant's bank or processor. The BIN identifies the Merchant Service Provider that signed up the merchant. **Length:** 6 digits. **Example:** 999995
BType	Integer	The merchant's business type. **Valid values:** 0 – Retail 1 – Mail order 2 – Electronic Commerce
Canceled	*Boolean*	*Used internally*
Category	String	The Merchant Category Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's industry classification. **Length:** 4 digits. **Example:** 5999
Chain	String	The Agent Chain Number issued by the merchant's bank or processor. This parameter is used to identify a specific chain of an agent organization. **Length:** 6 digits. **Example:** 000000
City	String	The merchant's postal/zip code as assigned by the merchant's bank or processor. **Length:** 5 or 9 digits. **Example:** 314193262
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
Country	String	The Country Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's country location. **Length:** 3 digits. **Valid Value:** 840 – U.S.
CreditTermType	*CreditTerminalType*	*N/A*
CSPhone	*String*	*N/A*
CurrencyCode	String	The Currency Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's settlement currency. **Length:** 3 digits. **Valid Value:** 840 – U.S. Dollars
DebitBIN	String	The Bank Identification Number issued by the merchant's bank or processor. The BIN identifies the Merchant Service Provider that signed up the merchant. **Length:** 6 digits. **Example:** 999995
DebitTermType	*CreditTerminalType*	*N/A*

Property Name	Data Type	Description - PccNPCSetup Properties
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
DID	*String*	*N/A*
ebtABA	String	The ABA number identifies the merchant to a direct debit switch. Applicable for Debit/EBT transactions. This value is provided to the merchant by their Merchant Services Provider or Processor. **Length:** 9 digits
ebtBIN	String	The Bank Identification Number issued by the merchant's bank or processor. The BIN identifies the Merchant Service Provider that signed up the merchant. **Length:** 6 digits. **Example:** 999995
ebtFCSID	String	The Food and Consumer Identifier identifies the Merchant as being certified and approved to accept Food Stamps. Applicable to EBT transactions only. **Length:** 0 to 7 characters.
ebtReimbursement	String	The Reimbursement Attribute designates the Reimbursement Fee applicable to a transaction. Applicable for Debit/EBT transactions. This value is assigned to the merchant by their Merchant Services Provider or Processor. **Length:** 1 character.
ebtSettleAgent	String	The Merchant Settlement Agent Number identifies the merchant settling agent. Applicable for Debit/EBT transactions. This value is provided to the merchant by their Merchant Services Provider or Processor. **Length:** 4 characters
ebtSharingGroup	String	The Sharing Group contains a listing of direct debit and EBT networks that may be accessed. This value is provided to the merchant by their Merchant Services Provider or Processor. The values must correspond to one of the Visa assigned direct debit network types. This data is part of the VisaNet direct debit data. **Length:** 1 to 30 characters.
EBTTermType	*DebitTerminalType*	*N/A*
ExpansionFactor	*String*	*N/A*
ExpressPay	*Boolean*	*N/A*
FloorLimit	*String*	*N/A*
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
IP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Language	String	Language indicator. **Length:** 2 digits. **Valid Values:** 00 – English 01 – Spanish 02 – Portuguese 03 – Reserved for Irish 04 – Reserved for French 05 – Reserved for German 06 – Reserved for Italian 07 – Reserved for Dutch
Location	*String*	*N/A*
MaxBatchSize	*String*	*N/A*
MCReversal	*Boolean*	*N/A*
PCard	Boolean	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE – Enable fields FALSE – Disable fields
Phone1	String	Primary phone number for settlement. If this value is set and Dial-up modem is the communication method, **PC**Charge will attempt to settle transactions using this phone number rather than the authorization phone number.

Property Name	Data Type	Description - PccNPCSetup Properties
Phone2	String	Secondary phone number for settlement. If this value is set and Dial-up modem is the communication method, **PC**Charge will attempt to settle transactions using this phone number rather than the authorization phone number.
Port	*String*	*The system/socket port used to connect to the processor when processing via TCP/IP.*
Reimbursement	String	The Reimbursement Attribute designates the Reimbursement Fee applicable to a transaction. Applicable for Debit/EBT transactions. This value is assigned to the merchant by their Merchant Services Provider or Processor. **Length:** 1 character.
RequireServerID	String	N/A
SettleAgent	String	The Merchant Settlement Agent Number identifies the merchant settling agent. Applicable for Debit/EBT transactions. This value is provided to the merchant by their Merchant Services Provider or Processor. **Length:** 4 characters
SettleConnect	String	The type of connection used for settlement: **Valid values:** 0 – Dial-up 1 – TCP/IP SSL 2 – TCP/IP Lease Line
SettleDialBackUp	String	For settlement, flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
SettleMaxBlockCount	String	The maximum block count used for settlement. **Default value:** 5
SettleMaxBlockSize	String	The maximum block size used for settlement. **Default value:** 12000 **Format:** bytes
SettlePrimaryPort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
SettlePrimaryURL	String	The Hostname, URL, or IP address used to connect to the processor when performing settlement via TCP/IP.
SettleTCP	Boolean	Flag that determines whether settlement will occur using dial-up modem or TCP/IP. TRUE – Settle via TCP/IP FALSE – Settle via Dial-up Modem
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
SharingGroup	String	The Sharing Group contains a listing of direct debit and EBT networks that may be accessed. This value is provided to the merchant by their Merchant Services Provider or Processor. The values must correspond to one of the Visa assigned direct debit network types. This data is part of the VisaNet direct debit data. **Length:** 1 to 30 characters.
Store	String	The Store Number assigned by the merchant's bank or processor. This parameter is used to identify a specific merchant store location. **Length:** 4 digits. **Example:** 0011
Terminal	String	The Terminal Number assigned by the merchant's bank or processor. This parameter is used to identify a specific store terminal. **Length:** 4 digits. **Example:** 9911.
TID	String	Terminal ID number for merchant account
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
TimeZone	String	The Time Zone Differential as assigned by the merchant's bank or processor. This value provides the standard local time zone differential from Greenwich Mean Time (GMT). **Length:** 3 digits. **Valid Values:** 705 – Eastern 706 – Central 707 – Mountain 708 – Pacific **Note:** Replace the leading 7 with a 1 if Daylight Savings is not observed. **Example:** 107 – Arizona

Property Name	Data Type	Description - PccNPCSetup Properties
URL	*String*	*N/A*
URL2	*String*	*N/A*
VSReversal	Boolean	Indicates whether VISA reversals will be processed.

PccNPCSetup Methods

Method Name	Returned Value	Description - PccNPCSetup Methods
CreateNPCDebitExtFile	*None*	*Used internally*
CreateNPCExtFile	*None*	*Used internally*
IsExtendedInfoValid	Boolean	Validates that each configuration field has been entered. Returns FALSE if any of the fields are left blank.
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.
ShowDebit	Boolean	Shows a GUI form that allows the end-user to enter Debit configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowDebit.
ShowEBT	Boolean	Shows a GUI form that allows the end-user to enter EBT configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowEBT.

PccPrivateSetup

This class contains information about the Private Label company setup in the current instance of **PC**Charge. This is a Multi Use class.

PccPrivateSetup Properties

Property Name	Data Type	Description - PccPrivateSetup Properties
Authorize	Boolean	Flag that indicates whether the merchant will activate the processing of Private Label cards.
BINEnd	String	The ending BIN range of the private label card
BINStart	String	The starting BIN range of the private label card
CardDescription	String	Description of the type of private label card to be used
CardIndex	Integer	The card index
CardType	String	The type of private label card to be used
CheckDigitType	String	The type of check digit routine used to validate private label cards.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
IsSetup	*Boolean*	*Used internally*
MaxCards	*Integer*	*Used internally*
MaxLength	Integer	Sets maximum length of the private label card numbers.
MerchantNumber	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. **Max Length**: 32 characters. This value can be alphanumeric.
MinLength	Integer	Sets minimum length of the private label card numbers.
Multiplier	String	The multiplier that is used to verify that a private label card is valid.
PrimaryPhone	String	The Primary number that will be used when processing transactions via dial-up modem.
Processor	String	The code for the processing company. This value can be no more than four characters and must be capitalized. Valid values: PRPN – Periphonic ADSI – Alliance Data Systems
SecondaryPhone	String	The Secondary number that will be used when processing transactions via dial-up modem.
Settle	*Boolean*	*Used internally*

PccPrivateSetup Methods

Method Name	Returned Value	Description - PccPrivateSetup Methods
CreatePrivateFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PccRMRSSetup

This class contains National Check Network extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccRMRSSetup Properties

Property Name	Data Type	Description - PccRMRSSetup Properties
BounceFee	String	The fee used in cases of returned checks due to insufficient funds. **Format:** Dollars
Canceled	*Boolean*	*Used internally*
FTPAddress	String	FTP address for check image upload.
FTPPassword	String	Password for access to image upload FTP
FTPUser	String	User ID for access to image upload FTP
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
OwnerCode	String	Owner code for image upload FTP
SalesBalance	String	The current Sales Balance
SalesCount	String	The current Sales Count
Truncation	Boolean	Flag that indicates whether check truncation / conversion will occur.
TruncationTID	String	This is a unique identifier assigned by the merchant's bank or processor that identifies the merchant. It will be in the format: site id-location id-rule set , where site id can be from 1 to 5 characters (numeric), location id can be from 1 to 6 characters (numeric), and rule set can be from 1 to 4 characters (numeric). **Example:** 78-123456-9999 (dashes are necessary; no spaces).
VoidsBalance	String	The current Voids Balance
VoidsCount	String	The current Voids Count

PccRMRSSetup Methods

Method Name	Returned Value	Description - PccRMRSSetup Methods
CreateRMRSExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PCCSMTSSetup

This class contains Smart Transaction Systems extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PCCSMTSSetup Properties

Property Name	Data Type	Description - PCCSMTSSetup Properties
BType	String	The merchant's business type. **Valid values:** R – Retail / Other F – Restaurant
Canceled	*Boolean*	*Used internally*
Connect	String	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
ExpansionFactor	String	Percent by which restaurant-based transactions will be incremented during gratuity-related transactions. This setting only applies when transactions are processed using the **PC**Charge GUI.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
Port	String	The system/socket port used to connect to the processor when processing via TCP/IP.
RequireServerID	String	Indicates whether **PC**Charge will require a Server ID during gratuity-related restaurant transactions. Consult the section **Restaurant Transactions** for more information (see page 105). **Valid Values:** 0 – Server ID not required 1 – Server ID required
TerminalID	String	This is a unique identifier assigned by the merchant's bank or processor that identifies the merchant. This value may or may not be required. The merchant should check with their bank or processor.
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.

PCCSMTSSetup Methods

Method Name	Returned Value	Description - PCCSMTSSetup Methods
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.

PCCSPSGiftSetup

This class contains Secure Payment Systems Gift extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PCCSPSGiftSetup Properties

Property Name	Data Type	Description - PCCSPSGiftSetup Properties
BType	*String*	*N/A*
Canceled	*Boolean*	*Used internally*
Connect	String	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
ExpansionFactor	String	Percent by which restaurant-based transactions will be incremented during gratuity-related transactions. This setting only applies when transactions are processed using the **PC**Charge GUI.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
Port	String	The system/socket port used to connect to the processor when processing via TCP/IP.
RequireServerID	String	Indicates whether **PC**Charge will require a Server ID during gratuity-related restaurant transactions. Consult the section **Restaurant Transactions** for more information (see page 105). **Valid Values:** 0 – Server ID not required 1 – Server ID required
TerminalID	String	This is a unique identifier assigned by the merchant's bank or processor that identifies the merchant. This value may or may not be required. The merchant should check with their bank or processor.
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.

PCCSPSGiftSetup Methods

Method Name	Returned Value	Description - PCCSPSGiftSetup Methods
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.

PccSPSSetup

This class contains Secure Payment Systems Check extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccSPSSetup Properties

Property Name	Data Type	Description - PccSPSSetup Properties
Canceled	*Boolean*	*Used internally*
DLLimit	String	The limit of the transaction amount after which a driver's license number is required. **Format:** Dollars.
FTPAddress	String	FTP address for check image upload.
FTPPassword	String	Password for access to image upload FTP
FTPUser	String	User ID for access to image upload FTP
Guarantee	Boolean	Flag that indicates whether check guarantee will occur.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
Port	String	The system/socket port used to connect to the processor when processing via TCP/IP.
SalesBalance	String	The current Sales Balance
SalesCount	String	The current Sales Count
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
Truncation	*Boolean*	*N/A*
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
VoidsBalance	String	The current Voids Balance
VoidsCount	String	The current Voids Count

PccSPSSetup Methods

Method Name	Returned Value	Description - PccSPSSetup Methods
CreateSPSExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

Method Name	Returned Value	Description - PccSPSSetup Methods
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.

PccSVSISetup

This class contains Stored Value Systems Gift extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccSVSISetup Properties

Property Name	Data Type	Description - PCCSVSISetup
Canceled	*Boolean*	*Used internally*
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
Division	String	Assigned by the merchant. Identifies the division at which the card is being accepted.
Store	String	Assigned by the merchant. Uniquely identifies the location at which the card is being accepted.
CurrencyCode	String	Indicates which currency to be used with the card.
Timeout	String	Determines how long PCCharge will wait for an authorization to timeout.
Terminal	String	Assigned by the merchant. Identifies the terminal where the transaction took place.
AllowTips	String	Indicates if tips are supported by the merchant: Valid Values: 0 – No Tips Allowed 1 – Tips Allowed
RoutingIndicator	String	Six character range of values in the dial-up service provider's POS routing tables provided by the processor.
UseTestSystem	String	Indicates if transaction be sent to the development system. Valid values: 0 – Do not send to development system 1 – Send to development system

PccSVSISetup Methods

Method Name	Returned Value	PCCSVSISetup Methods
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

PCCVisaSetup

This class contains TSYS/Vital extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccVisaSetup Properties

Property Name	Data Type	Description - PccVisaSetup Properties
ABA	String	The ABA number identifies the merchant to a direct debit switch. Applicable for Debit/EBT transactions. This value is provided to the merchant by their Merchant Services Provider or Processor. **Length:** 9 digits
Agent	String	The Agent Bank Number issued by the merchant's bank or processor. This parameter is used to identify a specific agent entity of the member bank or processor. **Length:** 6 digits. **Example:** 111111
AuthPrimaryPort	*String*	*N/A*
AuthPrimaryURL	*String*	*N/A*
Batch	String	The Current batch number. This value is incremented by the processor after each successful settlement.
Bin	String	The Bank Identification Number issued by the merchant's bank or processor. The BIN identifies the Merchant Service Provider that signed up the merchant. **Length:** 6 digits. **Example:** 999995
BType	Integer	The merchant's business type. **Valid values:** 0 – Retail 1 – Mail order 2 – Electronic Commerce 4 – Restaurant
Canceled	*Boolean*	*Used internally*
Category	String	The Merchant Category Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's industry classification. **Length:** 4 digits. **Example:** 5999
Chain	String	The Agent Chain Number issued by the merchant's bank or processor. This parameter is used to identify a specific chain of an agent organization. **Length:** 6 digits. **Example:** 000000
City	String	The merchant's postal/zip code as assigned by the merchant's bank or processor. **Length:** 5 or 9 digits. **Example:** 314193262
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
Country	String	The Country Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's country location. **Length:** 3 digits. **Valid Value:** 840 – U.S.
CreditTermType	CreditTerminalType	Cardholder-Activated Terminal Type. This setting determines which Cardholder Identification Code that will be passed to the processor. This value determines the method used to authenticate the identify of the cardholder for the transaction. **Valid values:** 0 – Non-Terminal 1 – Self-Serve Limited Amount 2 – Self-Serve Terminal
CSPhone	String	The Merchant Local Telephone Number. **Length:** 11 characters. **Format:** NNN-nnnnnnn where NNN is the area code and nnnnnnn is the telephone number. The hyphen is required. **Example:** 800-7259264
CurrencyCode	String	The Currency Code assigned by the merchant's bank or processor. This parameter is used to identify the merchant's settlement currency. **Length:** 3 digits. **Valid Value:** 840 – U.S. Dollars
DebitBIN	String	The Bank Identification Number issued by the merchant's bank or processor. The BIN identifies the Merchant Service Provider that signed up the merchant. **Length:** 6 digits. **Example:** 999995

Property Name	Data Type	Description - PccVisaSetup Properties
DebitTermType	CreditTerminalType	Cardholder-Activated Terminal Type. This setting determines which Cardholder Identification Code that will be passed to the processor. This value determines the method used to authenticate the identify of the cardholder for the transaction. **Valid values:** 0 – Non-Terminal 1 – Automated Dispensing
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
DID	*String*	*N/A*
ebtABA	String	The ABA number identifies the merchant to a direct debit switch. Applicable for Debit/EBT transactions. This value is provided to the merchant by their Merchant Services Provider or Processor. **Length:** 9 digits
ebtBIN	String	The Bank Identification Number issued by the merchant's bank or processor. The BIN identifies the Merchant Service Provider that signed up the merchant. **Length:** 6 digits. **Example:** 999995
ebtFCSID	String	The Food and Consumer Identifier identifies the Merchant as being certified and approved to accept Food Stamps. Applicable to EBT transactions only. **Length:** 0 to 7 characters.
ebtReimbursement	String	The Reimbursement Attribute designates the Reimbursement Fee applicable to a transaction. Applicable for Debit/EBT transactions. This value is assigned to the merchant by their Merchant Services Provider or Processor. **Length:** 1 character.
ebtSettleAgent	String	The Merchant Settlement Agent Number identifies the merchant settling agent. Applicable for Debit/EBT transactions. This value is provided to the merchant by their Merchant Services Provider or Processor. **Length:** 4 characters
ebtSharingGroup	String	The Sharing Group contains a listing of direct debit and EBT networks that may be accessed. This value is provided to the merchant by their Merchant Services Provider or Processor. The values must correspond to one of the Visa assigned direct debit network types. This data is part of the VisaNet direct debit data. **Length:** 1 to 30 characters.
EBTTermType	DebitTerminalType	Cardholder-Activated Terminal Type. This setting determines which Cardholder Identification Code that will be passed to the processor. This value determines the method used to authenticate the identify of the cardholder for the transaction. **Valid values:** 0 – Non-Terminal 1 – Automated Dispensing
ExpansionFactor	String	Percent by which restaurant-based transactions will be incremented during gratuity-related transactions. This setting only applies when transactions are processed using the **PC**Charge GUI.
ExpressPay	Boolean	Express Pay flag. This setting only applies if the Business Type is set to retail. If this flag is set to TRUE and the amount of the transaction is less than the FloorLimit amount, **PC**Charge will not authorize the transaction—it will only place the transaction in the open batch. Express Pay is usually used in a quick service environment with small ticket items. **Note:** Using Express Pay will increase transaction processing costs.
FloorLimit	String	The floor limit amount. This setting is only applicable if the Business Type is set to retail. The floor limit is the maximum transaction amount that will be accepted by **PC**Charge when processing an Express Pay transaction
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
IP	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.

Property Name	Data Type	Description - PccVisaSetup Properties
Language	String	Language indicator. **Length:** 2 digits. **Valid Values:** 00 – English 01 – Spanish 02 – Portuguese 03 – Reserved for Irish 04 – Reserved for French 05 – Reserved for German 06 – Reserved for Italian 07 – Reserved for Dutch
Location	String	The Merchant Location Number provides additional information on the location of the merchant. **Length:** 5 characters. **Example:** 00001 **Note:** This value should be 00001 unless otherwise specified by the merchant's bank or processor.
MaxBatchSize	String	Specifies the maximum number of transactions per batch that **PC**Charge will send to the processor. If the number of transactions to be settled is greater than the number specified in this setting, **PC**Charge will split the batch into multiple batches, each containing (at most) the number transactions specified in this setting. The batches are then sent to the processor one at a time. **Example:** A merchant has 250 transaction to settle and the MaxBatchSize is set to 100. **PC**Charge will send two 100-transaction batches and one 50-transaction batch. **Max Value:** 999
MCReversal	*Boolean*	*N/A*
PCard	Boolean	Flag that indicates to enable or disable the Commercial / Purchasing card fields (Tax and Customer code) in the **PC**Charge GUI. TRUE – Enable fields FALSE – Disable fields
Phone1	String	Primary phone number for settlement. If this value is set and Dial-up modem is the communication method, **PC**Charge will attempt to settle transactions using this phone number rather than the authorization phone number.
Phone2	String	Secondary phone number for settlement. If this value is set and Dial-up modem is the communication method, **PC**Charge will attempt to settle transactions using this phone number rather than the authorization phone number.
Port	String	The system/socket port used to connect to the processor when processing via TCP/IP.
Reimbursement	String	The Reimbursement Attribute designates the Reimbursement Fee applicable to a transaction. Applicable for Debit/EBT transactions. This value is assigned to the merchant by their Merchant Services Provider or Processor. **Length:** 1 character.
RequireServerID	String	Indicates whether **PC**Charge will require a Server ID during gratuity-related restaurant transactions. Consult the section **Restaurant Transactions** for more information (see page 105). **Valid Values:** 0 – Server ID not required 1 – Server ID required
SettleAgent	String	The Merchant Settlement Agent Number identifies the merchant settling agent. Applicable for Debit/EBT transactions. This value is provided to the merchant by their Merchant Services Provider or Processor. **Length:** 4 characters
SettleConnect	*String*	*N/A*
SettleDialBackUp	*String*	*N/A*
SettleMaxBlockCount	*String*	*N/A*
SettleMaxBlockSize	*String*	*N/A*
SettlePrimaryPort	*String*	*N/A*
SettlePrimaryURL	*String*	*N/A*
SettleTCP	*Boolean*	*N/A*
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
SharingGroup	String	The Sharing Group contains a listing of direct debit and EBT networks that may be accessed. This value is provided to the merchant by their Merchant Services Provider or Processor. The values must correspond to one of the Visa assigned direct debit network types. This data is part of the VisaNet direct debit data. **Length:** 1 to 30 characters.

Property Name	Data Type	Description - PccVisaSetup Properties
Store	String	The Store Number assigned by the merchant's bank or processor. This parameter is used to identify a specific merchant store location. **Length:** 4 digits. **Example:** 0011
Terminal	String	The Terminal Number assigned by the merchant's bank or processor. This parameter is used to identify a specific store terminal. **Length:** 4 digits. **Example:** 9911.
TID	*String*	*N/A*
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
TimeZone	String	The Time Zone Differential as assigned by the merchant's bank or processor. This value provides the standard local time zone differential from Greenwich Mean Time (GMT). **Length:** 3 digits. **Valid Values:** 705 – Eastern 706 – Central 707 – Mountain 708 – Pacific **Note:** Replace the leading 7 with a 1 if Daylight Savings is not observed. **Example:** 107 – Arizona
URL	*String*	*N/A*
URL2	*String*	*N/A*
VSReversal	Boolean	Indicates whether VISA reversals will be processed.

PccVisaSetup Methods

Method Name	Returned Value	Description - PccVisaSetup Methods
CreateVisaAdvanceFile	*None*	*Used internally*
CreateVisaDebitExtFile	*None*	*Used internally*
CreateVisaEBTExtFile	*None*	*Used internally*
CreateVisaExtFile	*None*	*Used internally*
CreateVisaIPNExtFile	*None*	*Used internally*
IsExtendedInfoValid	Boolean	Validates that each configuration field has been entered. Returns FALSE if any of the fields are left blank.
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.
ShowDebit	Boolean	Shows a GUI form that allows the end-user to enter Debit configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowDebit.
ShowEBT	Boolean	Shows a GUI form that allows the end-user to enter EBT configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowEBT.

PccVLNKSetup

This class contains ValueLink extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccVLNKSetup Properties

Property Name	Data Type	Description - PccVLNKSetup Properties
AltMerchNum	String	Alternate merchant ID number. **Recommended use:** merchant designated store/location number.
BType	String	The merchant's business type. **Valid values:** 0 – Retail 1 – Restaurant 2 – Electronic Commerce
Canceled	*Boolean*	*Used internally*
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
ExpansionFactor	String	Percent by which restaurant-based transactions will be incremented during gratuity-related transactions. This setting only applies when transactions are processed using the **PC**Charge GUI.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
PreAuth	Boolean	Flag that indicates whether to simulate Pre-/Post-Auth process
RequireClerkID	String	Flag the indicates whether the Clerk/Server ID is required when processing gift transactions **Valid values:** 0 – Clerk ID not required 1 – Clerk ID required
SplitTender	Boolean	Flag that indicates whether to allow split-tender scenario
TerminalID	String	The Terminal Number assigned by the merchant's bank or processor. This parameter is used to identify a specific store terminal. **Length:** 4 digits.

PccVLNKSetup Methods

Method Name	Returned Value	Description - PccVLNKSetup Methods
CreateVLNKExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.

Method Name	Returned Value	Description - PccVLNKSetup Methods
ShowGC	Boolean	Shows a GUI form that allows the end-user to enter extended or advanced Gift Card configuration information such as the business type, communication method, or other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowGC.

PccVTECSetup

This class contains Valutec extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccVTECSetup Properties

Property Name	Data Type	Description - PccVTECSetup Properties
Canceled	*Boolean*	*Used internally*
CashierFlag	*String*	*Used internally*
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
DialBackup	*Boolean*	*N/A*
ExpDateFlag	*String*	*Used internally*
GiftConnect	Integer	Indicates method of connection to processor when processing gift cards. **Valid values:** 0 – Dial-up 1 – TCP/IP
GiftDialBackup	*Boolean*	*N/A*
GiftPort	String	The system/socket port used to connect to the processor when processing via TCP/IP.
GiftSettleTimeOut	String	The Internet Settlement Timeout Value. If GiftDialBackup is set to TRUE, GiftSettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
GiftTimeOut	String	The Internet Authorization Timeout Value. If GiftDialBackup is set to TRUE, GiftTimeOut determines how long **PC**Charge will wait for a gift card transaction to time out before attempting the transaction via dial **Format:** Seconds
GiftURL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
Industry	String	The merchant's business type. **Valid values:** 1 – Retail 2 – Restaurant
MerchCardNum	String	The merchant card assigned by the merchant's bank or processor. **Length:** 10 digits.
ModeFlag	*String*	*Used internally*
Password1	*String*	*Used internally*
Password2	*String*	*Used internally*
Password3	*String*	*Used internally*
PasswordFlag	*String*	*Used internally*
Port	String	The system/socket port used to connect to the processor when processing via TCP/IP.
Receipt1	*String*	*Used internally*
Receipt2	*String*	*Used internally*
Receipt3	*String*	*Used internally*
Receipt4	*String*	*Used internally*
ReceiptFlag	*String*	*Used internally*

Property Name	Data Type	Description - PccVTECSetup Properties
SettleTimeOut	String	The Internet Settlement Timeout Value. If DialBackup is set to TRUE, SettleTimeOut determines how long **PC**Charge will wait for a settlement operation to time out before attempting the settlement via dial **Format:** Seconds
SplitTender	String	Flag that determines whether split tender is enabled. **Valid values:** 0 – Split Tender disabled 1 – Split Tender enabled
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
TipFlag	*String*	*Used internally*
TipPercent	*String*	*Used internally*
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.

PccVTECSetup Methods

Method Name	Returned Value	Description - PccVTECSetup Methods
CreateVTECAdvanceFile	*None*	*Used internally*
CreateVTECExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowAdvanced	Boolean	Shows a GUI form that allows the end-user to enter Advanced configuration information such as the communication method and other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowAdvanced.
ShowGC	Boolean	Shows a GUI form that allows the end-user to enter extended or advanced Gift Card configuration information such as the business type, communication method, or other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowGC.

PccWRLDSetup

This class contains World extended information for the current instance of **PC**Charge. This is a Public Not Creatable class.

PccWRLDSetup Properties

Property Name	Data Type	Description - PccWRLDSetup Properties
Canceled	*Boolean*	*Used internally*
Connect	Integer	Indicates method of connection to processor. **Valid values:** 0 – Dial-up 1 – TCP/IP
DialBackup	Boolean	Flag that indicates whether to use the backup dial connection if the Internet connection is not available. **Value Values:** TRUE – Use Dial-Up modem for Backup FALSE – Do not us Dial-Up modem for Backup
Index	Integer	The Merchant Number index. If Index is set to a value greater than 0, the tid.pcc file will be accessed and the merchant number at that index in the file will be used. Index should be set prior to calling the Load, Save, or Show methods. The index of the merchant number is determined by the order that it was added to **PC**Charge. For example, the first merchant number added to **PC**Charge will have an index of "1", the second, "2", etc.
Port	String	The system/socket port used to connect to the processor when processing via TCP/IP.
PrimaryURL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
RequireClerkID	String	Flag the indicates whether the Clerk ID is required when processing gift transactions **Valid values:** 0 – Clerk ID not required 1 – Clerk ID required
SecondaryIP	*Boolean*	*Used internally*
SecondaryURL	String	The Secondary Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.
TimeOut	String	The Internet Authorization Timeout Value. If DialBackup is set to TRUE, TimeOut determines how long **PC**Charge will wait for an authorization to time out before attempting the transaction via dial **Format:** Seconds
URL	String	The Hostname, URL, or IP address used to connect to the processor when processing via TCP/IP.

PccWRLDSetup Methods

Method Name	Returned Value	Description - PccWRLDSetup Methods
CreateWRLDExtFile	*None*	*Used internally*
Load	Boolean	Loads the configuration data from the configuration file(s) and populates the various setup properties with the data. The data in the properties can then be modified programmatically or can be modified by the end-user using the GUI form that is displayed by the Show method. If the data is modified programmatically, invoke the Save method to update the configuration file(s) with the new values. After calling Load, TRUE is returned if successful, otherwise FALSE is returned. **Note:** Set the Index property prior to calling Load.
Save	Boolean	Updates the configuration file(s) with the values currently stored in the various setup properties. Returns TRUE if successful, FALSE otherwise.
Show	Boolean	Shows a GUI form that allows the end-user to enter or modify configuration information. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling Show.
ShowGC	Boolean	Shows a GUI form that allows the end-user to enter extended or advanced Gift Card configuration information such as the business type, communication method, or other related settings. Returns TRUE if successful, FALSE otherwise. **Note:** If the end-user clicks OK after modifying configuration data, the data will be saved automatically. If the end-user clicks Cancel, the data will not be saved. **Note:** Set the Index property prior to calling ShowGC.

Classes No Longer Supported / Internal Use Classes

The following list contains various **PC**Charge OLE/COM classes that are either:

1) Used internally by **PC**Charge
2) No longer supported by **PC**Charge

In either case, integrators should not attempt to programmatically access the properties, methods, or events contained in any of following classes. These classes will remain exposed for backwards compatibility purposes.

clsPurgeDatabase
PccASISetup
PCCBBGFSetup
PccBMONSetup
PccCallBack
PccCardReader
PccCcrdTcpAuth
PccCCRDSetup
PccCheckReader
PccConnect
PccDataHandler
PccDiscDialSetup
PccEcom
PCCEnCheckMSR
PccENCNAuth
PccENCNSetup
PccEXPRSetup
PccFraud
PccFHAWSetup
PccFTMSSetup
PccGSARAuth
PccIndeterminateBatch
PccIPGSSetup
PccISDNSetup
PccMAPPSetup
PccMDISetup
PccMPSSetup
PccMVRKSetup
PccNovusSetup
PccOfflineTrans
pccPenWare
PccPrivateLabel
PccRBOCDebit
PccRBOCSetup
PccReportPrinter
PccResponse
PccSSLTCPAuth
PccSystem
PccTelmSetup
PccTMHSetup
pccTran
PccUsers
PccVerification
PccVTECAuth
Xtimer

File Method

Introduction

As of **PC**Charge version 5.6 and above, **PC**Charge's primary message format is XML. ***The XML message format has replaced the legacy "INP" message format so that integrators will not be limited to fixed length files for integration.*** All of **PC**Charge's integration methods support the XML message format and also support backward compatibility so that integrations using the INP message format will still be able to process transactions using that format. However, all new features that are added to **PC**Charge will be supported only by the XML message format.

All references to the INP message format have been removed from this manual. Although it is highly recommended that all new (and existing) integrations take advantage of the XML message format, older copies of the DevKit that outline the INP message format are available for integrators upon request. Contact VeriFone, Inc. at 1-800-725-9264 to request a copy of an older DevKit manual.

!	**WARNING:** VeriFone, Inc. takes security seriously therefore we have enhanced our integration methods to enhance security for sensitive data. If using **File Method** integration to PCCharge we strongly suggest you consider migrating towards a more secure integration method such as **TCP/IP SSL** socket integration.

File Method Integration

The File Method allows integrators to perform integration functions using flat text files. The files can be created on a machine running any operating system (Windows, UNIX, Mac, etc.). To process transactions via the File Method, the application should:

1. Check if `SYS.PCC` exists in the **PC**Charge directory. If it does, there is an error state. The number in the first byte of the file indicates the error type. The number is an error code listed in the **DevKit Constants** section (see page 143). If the file does not exist, then **PC**Charge is running and ready to receive transactions.

2. Write the transaction information to a file in the XML message format (ASCII text) using the file layouts described in this section.

3. Name the file <*user name*>`.inx` where <*user name*> is a user that is registered in **PC**Charge*.

4. Place the <*user name*>`.inx` file into the **PC**Charge directory. The transaction will now be processed by **PC**Charge.

5. Wait for <*user name*>`.oux` to appear in the **PC**Charge directory.

6. Read the values from <*user name*>`.oux`. The most important information is returned in the `RESULT` and `AUTH_CODE` tags.

7. Delete <*user name*>`.oux` and any other files with the same name, regardless of the extension. Make sure **not** to delete the `Users.pcc` file. **It is extremely important to delete these files.** Not deleting the `.oux` file or any associated transaction file could cause the client application to read the same results later.

- If **PC**Charge is set up with an unlimited user license, <*user name*> can also be any 8 character alphanumeric name. The user name must be in DOS file format, no spaces. Also, the filename must be the same as the value of that file's `USER_ID` tag.

Note: PC Charge is a single-threaded application. This means that **PC**Charge can only process one transaction at a time. Keep in mind that no two transaction requests can be submitted at the same time with the same user name.

!	**WARNING:** VeriFone, Inc. takes security seriously therefore we have enhanced our integration methods to enhance security for sensitive data. If using **File Method** integration to PCCharge we strongly suggest you consider migrating towards a more secure integration method such as **TCP/IP SSL** socket integration.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

File Layout Specifications

In order for a request to be processed, the request file (.inx) must open and close with an XML_FILE tag and each request within the file must open and close with an XML_REQUEST tag. Currently, **PC**Charge will only support one request per file; this may change in the future. Within the XML_REQUEST tag put the tags and values of all the data that will be needed to process the transaction.

Once the request has been processed, a response file (.oux) will be returned. The response file, like the request file, will open with an XML_FILE tag and the transaction response will be wrapped within the XML_REQUEST tag.

XML Data Validation

PCCharge provides request file data validation for integrators implementing the File Method. **PC**Charge uses the Microsoft XDR (XML-Data Reduced) schema to provide this data validation. The file used by **PC**Charge to implement the data validation is called stnd.xdr. This file is installed by **PC**Charge and resides in the \DTD folder within the **PC**Charge directory.

In order to perform XDR validation on the XML request:

1. The XML_FILE tag must reference the stnd.xdr file.
2. The tags submitted must be in the order that they appear in the stnd.xdr file.
3. Each tag must appear only once.
4. The content for each tag must be text only.

The following is an example of request that will be validated:

```
<XML_FILE xmlns="x-schema:.\dtd\stnd.xdr">
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<COMMAND>1</COMMAND>
		<PROCESSOR_ID>VISA</PROCESSOR_ID>
		<MERCH_NUM>999999999911</MERCH_NUM>
		<ACCT_NUM>5424180279791765</ACCT_NUM>
		<EXP_DATE>1211</EXP_DATE>
		<MANUAL_FLAG>0</MANUAL_FLAG>
		<TRANS_AMOUNT>1.00</TRANS_AMOUNT>
	</XML_REQUEST>
</XML_FILE>
```

Notice that the XML_FILE tag references the path of the stnd.xdr file. Also, the tags are in the same order that they appear in the stnd.xdr file. This file will be validated successfully.

If a request file *does not* pass validation, **PC**Charge will return an "Incomplete Trans" error.

The following is an example of a response that was returned because validation failed:

```
<XML_FILE>
	<XML_REQUEST>
		<USER_ID>User1</USER_ID>
		<RESULT>Error</RESULT>
		<AUTH_CODE>Incomplete Trans</AUTH_CODE>
	</XML_REQUEST>
</XML_FILE>
```

Many integrators do not wish to be limited by the requirements of XDR validation; specifically, the mandated order of the tags in the request file. If this is the case, simply remove the reference to the validation file, `stnd.xdr`, from the `XML_FILE` tag. The tags inside <XML_REQUEST> can now be placed in any order that the integrator sees fit.

The following is an example of request file that *will not* be validated but will still be processed successfully by PCCharge:

```
<XML_FILE>
	<XML_REQUEST>
		<ACCT_NUM>5424180279791765</ACCT_NUM>
		<EXP_DATE>1211</EXP_DATE>
		<MANUAL_FLAG>0</MANUAL_FLAG>
		<TRANS_AMOUNT>1.00</TRANS_AMOUNT>
		<PROCESSOR_ID>VISA</PROCESSOR_ID>
		<MERCH_NUM>999999999911</MERCH_NUM>
		<COMMAND>1</COMMAND>
		<USER_ID>User1</USER_ID>
	</XML_REQUEST>
</XML_FILE>
```

Notice that the `XML_FILE` tag does not reference the `stnd.xdr` file.

Note: Regardless of whether **PC**Charge validates the request data using the `stnd.xdr` file, the application should always perform input validation according to the API specifications prior to passing requests to **PC**Charge.

!	**WARNING:** VeriFone, Inc. takes security seriously therefore we have enhanced our integration methods to enhance security for sensitive data. If using **File Method** integration to PCCharge we strongly suggest you consider migrating towards a more secure integration method such as **TCP/IP SSL** socket integration.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

Credit File Layouts

This section describes the tags required to process credit card transactions.

Credit Input File (.inx)

Tag	Data Type	Description - Credit Input File (.inx)
USER_ID° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62). **Note:** The value passed in USER_ID must match the name of the .inx file. For example: <USER_ID>User2</USER_ID> and User2.inx.
COMMAND°	String	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
TROUTD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
PROCESSOR_ID° ***	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
MERCH_NUM° ***	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Credit Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
ACCT_NUM°	String	The credit card number that will be used when processing the transaction. **Max Length:** 20 characters. **Example:** 5424180279791765 **NOTE:** Is it the integrator's responsibility to remove spaces in the card number if there are spaces in the Track II card number data.
EXP_DATE°	String	The expiration date associated with the credit card number that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208
MANUAL_FLAG°	String	Flag that indicates whether the transaction was manually entered or swiped. If the transaction was swiped, the TRACK_DATA property must also be set. **Valid values:** 0 = manual transaction, 1 = swiped transaction
TRANS_AMOUNT°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".

Tag	Data Type	Description - Credit Input File (.inx)
REFERENCE	String	The reference number from the original transaction (returned by the processor). Set this property only if processing a Post-Authorization and the Post-Authorization is being used to add a Voice-Authorization to the batch or to "store" a Voice-Authorization. (For information on stored Voice-Authorizations, see page 91). The REFERENCE property does not need to be set if the Post-Authorization completes a standard Pre-Authorization using the TroutD value of the Pre-Authorization. See the section **Follow On Transactions** for more information (see page 70). **Max Length:** 8 characters. **Note:** NBS/ Fleet One cards require a Reference Number to be sent with each transaction. This is a minimum of 2 digits and a max of 15. This must be all numeric.
TRACK_DATA	String	The track II data captured from the magnetic strip of the credit card. The track II data is required to ensure the lowest per-transaction rate from the processing company when performing swiped transactions (Retail and Restaurant). Sending the track II data is not allowed if the merchant's industry type is MOTO or eCommerce. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in.
CUSTOMER_CODE	String	Customer code for purchasing/commercial cards. This property must be set for commercial card transactions in order to get the best discount rate. Additionally, the transaction's action code must indicate that the transaction is a commercial card transaction. **Note:** Global East (NDC), terminal based, requires the customer code be all upper case. **Max Length:** 25 characters, alphanumeric only.
CREDIT_PLAN_NUMBER	String	The credit plan numbers are established by the processor **CITI** for each merchant, they define the type of Disclaimer to print on receipts. This information will vary from merchant to merchant.
TAX_AMOUNT	String	The tax amount. This is the portion of the amount that is tax. Providing the tax amount is required to obtain the best rate on commercial card transactions. **Max Length**: 9 characters (including the decimal). **Format:** DDDDDD.CC. The transaction's action code must indicate that it is a commercial transaction. Tax amount should be included in the amount field.
PRINT_RECEIPTS_FLAG	String	The number of receipts that **PC**Charge should print for the transaction. This value will override the corresponding value in the **PC**Charge GUI. **PC**Charge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.
PERIODIC_PAYMENT_FLAG	String	Flag that indicates whether the transaction is a recurring transaction. **Valid values:** 1 = TRUE, 0 = FALSE **Note:** If periodic payment is set to true, the recurring billing flags must also be set to achieve the best processing rates.
OFFLINE_FLAG	String	Flag that indicates whether **PC**Charge should process the transaction offline. If the offline flag is set, **PC**Charge will put the transaction into a .BCH file that resides in the **PC**Charge directory for importing at a later time. The file can only be imported from the **PC**Charge GUI. **Valid values:** 1 = TRUE, 0 = FALSE
ZIP_CODE	String	The cardholder's zip code. The Zip property is used for address verification. **Max Length:** 9 digits. Address verification can only be performed on non-swiped transactions. **Note:** For manually keyed transactions, the Zip is required to qualify for the lowest transaction rates. **Note:** If submitting the 9-digit zip, do not include the dash. **Citi©** - When used with Citi Plan the zip is limited to 5-digits.
STREET	String	The cardholder's billing street address. The Street property is used for address verification. Address verification can only be performed on non-swiped transactions. **For FDC:** Use first 5 digits only. **Note:** For manually keyed transactions, Street is required to qualify for the lowest transaction rates. **Max Length:** 20 characters **Citi©** - When used with Citi Plan PO Box's are not allowed.

Tag	Data Type	Description - Credit Input File (.inx)
TICKET_NUM	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** For manually keyed transactions, TICKET_NUM is required to qualify for the lowest transaction rates. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.
CARDHOLDER	String	The cardholder's name. **Max Length:** 20 characters.
MCSN	String	**In a restaurant environment:** The server or cashier id. **Max Length:** 2. This field should be passed for reporting and reconciliation purposes. See the section **Restaurant Transactions** (see page 105) for more information. **Processor specific note:** The Server ID is required for AMEX card transactions. Also required when using the processor NB and GSAR in restaurant business type. **In a non-restaurant environment**, this field is the Multiple Count Sequence Number. This is the transaction number within the total number of payment installments in a recurring billing scenario. **Max Length**: 2 characters. **Example:** If there are 5 payments to be made and this transaction is the first transaction, set this property to "1". The first transaction should also include the CVV property, but this value should **not** be stored or sent for subsequent transactions.
MCSC	String	The Multiple Count Sequence Count. This is the total number of installments that will be charged in a non-restaurant recurring billing scenario. **Max Length**: 2 characters. **Example:** If there are 5 payments to be made, set this property to "5".
MULTI_FLAG	String	Flag that indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PC**Charge GUI. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid values:** 1 = TRUE, 0 = FALSE **Default value:** 0. See the section **Multi-trans Wait** for more information (see page 67). This Flag has no effect if processing will occur over IP or leased line.
CVV2	String	The CVV2 value for the transaction. The card verification value (CVV2 for Visa, CVC2 for MasterCard, and CID for AMEX and Discover) is a 3 or 4 digit number that is embossed in the signature panel for Visa, MasterCard, and Discover and on the front of the card for AMEX. All AMEX cards utilize a 4 digit CID. **Max Length:** 4 characters. CVV2 should only be passed on non-swiped transactions.
PRESENT_FLAG	String	**For Retail or Restaurant transactions**: Flag that indicates whether the card was present. **For eCommerce transactions**: Flag that indicates what type of transaction occurred. **Valid values:** 0 = Card not present, 1 = Card present (for Retail, MOTO, or Restaurant); D = Digital goods, P = Physical goods (for eCommerce)
ITEM_ID	String	The Item ID for the transaction. This field is only used for Chase Paymentech (GSAR) and can store five (5) four-digit codes that are defined by Chase Paymentech. **Example:** If ITEM_ID is set to 0001000020003000400005, it stores 5 item IDs (0001, 0002, 0003, 0004, and 0005). These numbers must be obtained from Chase Paymentech.
ID_NUMBER	String	Only required for Voyager cards, dependant on Restriction Code. Four to six digits. Note: Only used for Pre-Authorization transactions **Update:** Beginning with PCCharge version 5.8, this is also used for Citi© Private Label. This is the number on the driver's license or state ID.
ODOMETER	String	The odometer reading. Only required for Fleet One (7 digits), Voyager (7 digits), and Fuelman (6 digits) cards.
DRIVER_ID	String	Driver identification field. Only required for Wright Express, Voyager and Fleet One cards.
DRIVER_PIN	String	Driver personal identification number. Only required for Fuelman cards.

Tag	Data Type	Description - Credit Input File (.inx)
PRODUCT_DETAIL_AMOUNT_XX	String	**Note:** Only required for the processor NBS. This is the total dollar amount for PRODUCT_DETAIL_PRODUCT_CODE_XX being authorized. For example, PRODUCT_DETAIL_PRODUCT_CODE_1 has a PRODUCT_DETAIL_QUANTITY_1 = 2 and a PRODUCT_DETAIL_UNIT_PRICE_1 = $2.00, therefore the PRODUCT_DETAIL_AMOUNT_1 = $4.00
PRODUCT_DETAIL_COUNT	Stringq	**Note:** Only required for the processor NBS. All card types are configurable except for Fleet One which is limited to 7 records. Only 01 – 10 records are currently supported through PCCharge for all card types.
PRODUCT_DETAIL_CODE_XX	String	**Note:** This is the number of items for RODUCT_DETAIL_PRODUCT_CODE_XX. PCCharge will support 1 – 10. • NBS supports using 1 – 10. • BAMS Tandem (NPC) supports using 1 – 3.
PRODUCT_DETAIL_QUANTITY_XX	String	**Note:** Only used for the processor NBS. This is the unit price for PRODUCT_DETAIL_PRODUCT_CODE_XX. This is only used for Fleet One and Fuelman. Currently, PCCharge will support 1 – 10.
GRATUITY_AMNT	String	**For use with Restaurant transactions only.** The actual gratuity amount for a Sale with Gratuity (action code 14) , Gratuity (action code 13) , or Post-Authorization (action code 5) transaction. See the section **Restaurant Transactions** (see page 105) for more information.
GRATUITY_AMNT_EST	String	**For use with Restaurant transactions only.** The estimated gratuity amount for a Sale (action code 1) or Pre-Authorization (action code 4) transaction. If the GRATUITY_AMNT_EST is populated, **PC**Charge will submit the sum of the values in the TRANS_AMOUNT and GRATUITY_AMNT_EST fields for authorization. If the transaction is authorized, only the value in the TRANS_AMOUNT field will be placed in the **PC**Charge settlement file (if running a Sale). By using the GRATUITY_AMNT_EST, the merchant can help ensure that the customer has enough available credit on their card to leave a tip. Once the customer indicates the amount of the tip that will be left, a gratuity transaction (action code 13) must be performed on the sale prior to settlement in order to add the actual gratuity to the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. See the section **Restaurant Transactions** (see page 105) for more information. **Note:** It is recommended to check with the processor or merchant service provider for guidance on what amount to set this value to. Incorrectly setting this value can result in downgrades.
CMRCL_FLAG	String	The type of commercial card being submitted. See the section **Commercial Card Transactions** (see page 93) for more information. **Max Length**: 1 character **Valid values:** B – Business P,L,G -- Purchase C – Corporate F – Fleet
AMX_CHARGE_DESCRIPTION	String	The American Express Charge Description. This is a general description describing merchandise: the AMEX representative and the merchant will decide on an appropriate description. **Note:** Only Required for Retail, MOTO and Restaurant transactions when using AMEX direct settlement. **Max Length:** 23 bytes
AMX_DESCRIPTION_1	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement or TSYS **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS

Tag	Data Type	Description - Credit Input File (.inx)
AMX_DESCRIPTION_2	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AMX_DESCRIPTION_3	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
AMX_DESCRIPTION_4	String	American Express Description data. Additional description or information about merchandise—if populated, should be printed on the receipt. **Note:** Only used for Retail transactions when using AMEX direct settlement. **Max Length:** 40 bytes This field is optional and should only be provided if the transaction will be settled directly with Amex or TSYS
TRANS_STORE	String	Flag indicating whether a Voice Authorization transaction should be stored. This flag should only be submitted when performing a Post-Authorization transaction (action code 5) that includes an authorization code from the voice operator. For more information on stored Voice-Authorizations, see page 91. **Valid Value:** 1 - Store the Voice Authorization transaction.
TXN_TIMEOUT	String	The number of seconds after which a timeout error will be returned from **PC**Charge. The count will start when the transaction is submitted to **PC**Charge. An error will be returned if the transaction has not finished processing when the time period expires. It is highly recommended that integrators review the section **Timeouts** (see page 59). **Note:** This tag only works when using the TCP Interface.
AUTH_CODE	String	The Authorization code. This value is returned by the issuing bank and should only be set in a transaction request if processing a Post-Authorization and the Post-Authorization is being used to add a Voice-Authorization to the batch or to "store" a Voice-Authorization. (For information on stored Voice-Authorizations, see page 91). The AuthCode property does not need to be set if the Post-Authorization completes a standard Pre-Authorization using the TroutD value of the Pre-Authorization. See the section **Follow On Transactions** for more information (see page 70).
TAX_EXEMPT	String	Tax Exempt Flag. This flag is used to indicate if the purchase is tax exempt. Used only for Commercial Card Transactions. **Valid Values:** 1 – Purchase is tax exempt; 0 – Purchase is not tax exempt.
DEST_ZIP_CODE	String	Destination Zip Code for American Express purchasing/commercial cards. This property must be set for American Express commercial card transactions when using American Express as the processor (or via split dial) in order to get the best discount rate. Additionally, the transaction's action code must indicate that the transaction is a commercial card transaction.
ITEM_ID	String	For Chase Paymentech Only – EID Number
BILLPAY	String	Only valid for Visa debit and credit transactions. It is used to indicate the transaction is being ran for payment of a bill (ultilty, monthly gym dues, etc.) **Valid values:** 0 – Non-Bill payment transaction 1 – Bill payment transaction

Tag	Data Type	Description - Credit Input File (.inx)
RESTRICTION_CODE	String	Only required for Voyager cards. This is used to determine the level of identification and which fields are required. Two digits. **Valid Values:** 00 - No ID Number or Odometer required. Fuel and Other allowed. 01 - No ID Number or Odometer required. Fuel only allowed. 10 - ID Number only required. Fuel and Other allowed. 11 - ID Number only required. Fuel only allowed. 20 - Odometer only required. Fuel and Other allowed. 21 - Odometer only required. Fuel only allowed. 30 - ID Number and Odometer required. Fuel and Other allowed. 31 - ID Number and Odometer required. Fuel only allowed. **Note:** Required for both manual and swiped transactions.
RFID	String	Set to 1 if card information was read from RFID (Radio Frequency Identification) device. If card was read from from RFID, track data must be populated and manual flag must be set to 1. Set to 0 otherwise.
VEHICLE_ID	String	Only required for Wright Express cards (5 digits) and Voyager cards (8 digits). **Note:** Required for both manual and swiped transactions.
CASHIER_NAME	String	Submit Cashier's login name.
PASSWORD	String	Submit Cashier's password.
CUSTOMER_FIRSTNAME [1]	String	Applicant's first name.
MIDDLE_INITIAL [1]	String	Applicant's middle initial.
CUSTOMER_LASTNAME [1]	String	Applicant's last name.
SUFFIX [1]	String	Applicant's suffix. (Ex. M.D., Ph.D, Jr.)
APT_SUITE [1]	String	Applicant's apartment or suite number.
CITY [1]	String	Applicant's city.
STATE [1]	String	Applicant state. (Ex: GA, FL, MA…)
EMAIL [1]	String	Applicant e-mail address. (Ex: XXX@XX.XXX)
PHONE_NUM [1]	String	Applicant's home phone number. Format xxxxxxxxxx 10-digits No Dashes
SS_NUM [1]	String	Applicant's Social Security Number. Format xxxxxxxxx 9-digits No Dashes
DOB [1]	String	Applicant's Date of Birth. Format is MMDDYYYY.
EMP_NAME [1]	String	Applicant's Employer's Company name.
WORK_PHONE_NUM [1]	String	Applicant data. Employer phone number. Format xxxxxxxxxx 10-digits.
HOME_OWNER [1]	String	Applicant data. Format: '**O**' = Own '**R**' = Rent '**B**' = Board '**P**' = Live w/ parents '**M**' = Military
ANNUAL_INCOME [1]	String	Applicant's annual income. Format: Up to 6 digits. Whole dollar amount with no decimals.
PHOTO_ID_STATE [1]	String	Applicant data. (Ex: GA, FL, MA…)
ID_TYPE [1]	String	Received as a response to a Credit App (P3)
STATE [1]	String	Received as a response to a Credit App (P3)
REPLY_FLAG [1]	Boolean	Reply notification. Format: **Y** = Accept **N** = Decline
SOURCE_CODE [1]	String	Default is Opt In = **"AP"** Format: Opt In - **"AP"** Opt Out - **"AO"**

Tag	Data Type	Description - Credit Input File (.inx)
ID_TYPE [1]	String	Applicant data. Type of photo identification being submitted. Format: **'D'** – Driver's License **'O'** – Other
FRAUD_FLAG [1]	Boolean	Code 10. Values are **1** or **0: 1** = True, **0** = False
AMOUNT_HEALTHCARE [#]	String	Total Healthcare Amount. This amount must be greater than or equal to the sum of the other amount categories. Max: **12 - digits** Format: **DDD.CC**
AMOUNT_PRESCRIPTION [#]	String	(Optional) Total amount of the prescription-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
AMOUNT_VISION [#]	String	(Optional) Total amount of the vision-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
AMOUNT_CLINIC [#]	String	(Optional) Total amount of the clinic-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
AMOUNT_DENTAL [#]	String	(Optional) Total amount of the dental-related healthcare expenses in this transaction. Max: **12 - digits** Format: **DDD.CC**
AMOUNT_TRANSIT [#]	String	(Optional) Reserved for future use. Max: **12 - digits** Format: **DDD.CC**
AMOUNT_COPAY [#]	String	(Optional) Reserved for future use. Max: **12 - digits** Format: **DDD.CC**
FSA [#]	Boolean	Indicates the transaction is FSA. This will bypass PCCharge BIN checking as PCCharge will assume that the POS has verified that this card has met the BIN requirements for an FSA transaction. **1** = True, **0** = False **Note:** If this is not passed it defaults to False.

° These properties are the minimum required to process a Sale or Pre-Authorization transaction.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

*** If the "Use Default Processor" option is enabled in the **PC**Charge preferences, and the PROCESSOR_ID and MERCH_NUM properties are omitted from the transaction request, **PC**Charge will process all transactions using the "Default Processor". The "Default Processor" is defined as the first merchant number that is set up **PC**Charge. Consult the **Multi-Merchant Support** section (see page 68) for more information on the "Use Default Processor" option. In addition, PROCESSOR_ID and MERCH_NUM should not be set when doing follow-on transactions. Refer to the section **Follow On Transactions** (see page 70) for more information.

[1] These properties are required to process **Citi applications**. For more information and examples see **Citi Credit Plan** on page 116.

[#] Additional tags used to process FSA/HRA transactions. For more information please see the section about **FSA on page 99**.

Charge Output File (.oux)

Tag	Data Type	Description - Charge Output File (.oux)
USER_ID	String	Returns the User name that is associated with the transaction. This value is echoed back from the original transaction. The User name will be in DOS file format, max 8 characters.
MERCH_NUM	String	Returns the merchant number that was specified in the MerchantNumber property.

Tag	Data Type	Description - Charge Output File (.oux)
TROUTD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
RESULT	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
AMOUNT_DUE	String	Returns the amount due, only for Elavon (NOVA) and **FSA (on page 99)** pre-paid functionality. **Note:** "Enable Pre-Paid Cards" option within PCCharge MUST be checked for this feature to work. The option is located under Setup>Credit Card Company>Extended Data and only with the Credit Card Company set to Elavon (NOVA). This option is only configurable in *PCCharge version 5.7.1 release I sp9a* and above.
AUTH_CODE	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
AUTH_AMOUNT	String	The authorized amount of the transaction. Only used for pre-paid cards with Elavon (NOVA) and **FSA (on page 99)**.
CC_AVAIL_BALANCE	String	Returns the PrePaid card balance. Only for pre-paid credit cards with Elavon (NOVA) and **FSA (on page 99)**.
DC_AVAIL_BALANCE	String	Returns the available balance on pre-paid debit cards. Only for pre-paid debit cards with Elavon (NOVA).
REFERENCE	String	Returns the reference number associated with the transaction. The reference number is assigned by the card associations. The reference number is used to help identify the transaction and is useful for the cardholder and merchant when doing research. This value is not returned with all transactions.
AVS_CODE	String	Returns the AVS response code from the issuing bank. If performing Address Verification on card-not-present transactions, this code indicates how well the AVS information passed in matches what the issuing bank has on file for the cardholder. Consult the section **DevKit Constants** for a description of values that may be returned (see page 143)
ADD_TEXT	String	Only supported on Fleet One, this field contains miscellaneous additional text returned from host. Currently PCCharge will support GetAddText1-GetAddText4.
RESTRICT_CODE	String	**Note:** Only supported on Fleet One. The product restriction code.
TRACE_NUMBER	String	The trace number returned from the processor. This value is not returned by all processing companies.
TRANS_DATE	String	Returns the date that the transaction was processed. This value is not returned by all processing companies.
TRANSACTION_REFERENCE_NUMBER	String	Returns the transaction reference number. This value is not returned by all processing companies.
TICKET	String	Returns the ticket number or invoice of the transaction. This value is echoed back from the original transaction or is generated by **PC**Charge if one is required to complete the transaction.
INTRN_SEQ_NUM	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
TRANS_ITEM_NUM	String	Returns the Transaction Item number or the number that is associated with the transaction in the settlement file. This value is not returned by all processing companies.
TRANSACTION_REFERENCE_NUMBER	String	Returns the transaction reference number from the processor. Only for pre-paid credit cards with Elavon (NOVA).
TBATCH	String	Returns the active batch number for the transaction. This value is not returned by all processing companies.

Tag	Data Type	Description - Charge Output File (.oux)
TRANS_ID	String	Returns the Transaction Identifier that is returned from the processor. This value is not returned by all processing companies.
TICODE	String	Returns the Transaction Indicator Code that is returned from the processor. The Transaction Indicator Code is a Validation code for VISA / MasterCard. This value is not returned by all processing companies.
IND	String	Returns the IND code. The IND code is a transaction description code and an Interchange compliance field. This value is not returned by all processing companies.
MSI	String	Returns the Market Specific Indicator. This value indicates the transaction's market segment. This value is assigned by the card associations and is not returned with all transactions.
RET	String	Returns the Retrieval reference number. This value is not returned by all processing companies
PEM	String	Returns the POS entry mode that is associated with the transaction. This value is not returned by all processing companies.
PS2000	String	Returns the PS2000 indicator from the processor. This value is not returned by all processing companies.
TIM	String	Returns the Time of the transaction. This value is not returned by all processing companies.
ACI	String	Returns the Authorization Characteristics Indicator is that is provided by the card associations. This value is stored for settlement.
PROC_RESP_CODE	String	Returns the response code that is provided by the processor. This value is not returned by all processing companies.
REC	String	Returns the record number of the transaction in the reversal file. Will return -1 if the processor doesn't support reversals. This value is not returned by all processing companies.
RECEIPT	String	Only used for the processor CITI private label cards. CITI private label cards will return the Receipt Disclaimer to be printed on the bottom of receipts based off of the Credit Plan Number.
CMRCL_TYPE	String	Returns the type of commercial card that was used for the transaction. This value is not returned by all processing companies.
CVV2_CODE	String	Returns the CVV2/CVC2/CID response code from the issuing bank. If performing CVV2/CVC2/CID validation on card-not-present transactions, this code indicates if the CVV2/CVC2/CID code passed in matches what the issuing bank has on file for the cardholder. Consult the section **DevKit Constants** for a description of values that may be returned (see page 143)
PURCH_CARD_TYPE	String	Returns a flag indicating whether the processor indicated whether the card was a Purchasing Card or not. This value is not returned by all processing companies. **Valid values:** 1 = Purchasing Card, 0 = Otherwise
GRATUITY_AMNT	String	Returns the gratuity amount if one is associated with the transaction. This value is not returned by all processing companies.
GRATUITY_AMNT_EST	String	Returns the estimated gratuity if one is associated with the transaction. This value is not returned by all processing companies.
RESULT_CODE	String	Returns a numerical representation of the result of the transaction. Currently, this field is only used for a batch transaction.
CMRCL_FLAG	String	Returns the Commercial Card Flag. This indicates what type of Commercial card was used for the transaction. This value is not returned by all processing companies.
NET_ID	String	Returns a one character identification code that identifies the network on which the transaction was approved. This value is not returned by all processing companies.
AUTH_SRC_CODE	String	Returns the Authorization Source Code. The authorization source code indicates to the processor who authorized the transaction. This value is not returned by all processing companies.
CARD_ID_CODE	String	Returns a code that is used to verify the identity of the cardholder.
ACCT_DATA_SRC	String	Returns the entry method of the transaction.
ECOMM_GOODS_IND	String	Returns a value indicating whether the goods sold were digital or physical in an e-commerce environment.

Tag	Data Type	Description - Charge Output File (.oux)
TERMINATION_STATUS	String	Returns a value of "6" when a transaction is successful. If anything other than a "6" is returned then the transaction may have resulted in an error. This applies to the following processors: **GSAR** and **NBS**
CORRELATION_UID [1]	String	Returned from Citi, needed for Counter Offer Submission.
PENDING_NUMBER [1]	String	Returned from Citi, needed for Counter Offer Submission.
REPLY_FLAG [1]	String	**'Y'** or **'N'** sent from Citi to notify message is a reply to an offer.
OPEN_TO_BUY [1]	String	Only present in the response if the "Display OTB" is turned ON. Decimal is included.
CREDIT_LIMIT [1]	String	Only present the in the response. Shows applicant's credit limit.

[1] These properties are required to process **Citi applications**. For more information and examples see **Citi Credit Plan** on page 116.

☛	**Note :** Integrators will use either the AMOUNT_DUE or the AUTH_AMOUNT tags to have the POS indicate the remaining purchase amount to the end user. *(Requires Split-Tender/Partial Auth Reversal enabled within PCCharge)*

☛	**Note :** When total cost of the transaction is covered by the initial purchase PCCharge will not return AMOUNT_DUE and AUTH_AMOUNT tags. *(Requires Split-Tender/Partial Auth Reversal enabled within PCCharge)*

Debit File Layouts

This section describes the tags required to process debit transactions.

When processing debit cards, a PINpad is required to allow the customer to enter their PIN. In addition, debit card information is always collected via a card swipe device, never via keyboard entry. Because of this, a card reader is also required.

When processing U.S. debit card transactions, merchants have the option of allowing the customer to receive cash back on a transaction. For instance, the customer purchases $50 of products and wants $25 cash back, set the Amount to 50.00 and CashBack to 25.00. This will withdraw a total of $75 from the debit card account, $50 for the products and $25 for cash to give to the customer.

Debit Input File (.inx)

Tag	Data Type	Description - Debit Input File (.inx)
USER_ID° °° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62). **Note:** The value passed in USER_ID must match the name of the .inx file. For example: <USER_ID>User2</USER_ID> and User2.inx.
COMMAND°°	String	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
TROUTD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
PROCESSOR_ID° °° ***	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
MERCH_NUM° °° ***	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Debit Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
ACCT_NUM° °°	String	The Debit card number that will be used when processing the transaction. **Max Length:** 20 characters. **Example:** 5424180279791765
EXP_DATE° °°	String	The expiration date associated with the Debit card number that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208 Set this property if there is an expiration date associated with the Debit card.
MANUAL_FLAG° °°	String	Flag that indicates whether the transaction was swiped or manually entered. This property must be set to 1 (swiped) and the TRACK_DATA property must also be set.

Tag	Data Type	Description - Debit Input File (.inx)
TRANS_AMOUNT° °°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
TRACK_DATA° °°	String	The track II data captured from the magnetic strip of the card. The track II data is required. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in.
PRINT_RECEIPTS_FLAG	String	The number of receipts that **PC**Charge should print for the transaction. This value will override the corresponding value in the **PC**Charge GUI. **PC**Charge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.
TICKET_NUM	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.
CARDHOLDER	String	The cardholder's name. **Max Length:** 20 characters.
GRATUITY_AMNT°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. This is the Gratuity Amount of the transaction.
TXN_TIMEOUT	String	The number of seconds after which a timeout error will be returned from **PC**Charge. The count will start when the transaction is submitted to **PC**Charge. An error will be returned if the transaction has not finished processing when the time period expires. It is highly recommended that integrators review the section **Timeouts** (see page 59). **Note:** This tag only works when using the TCP Interface.
SHIFT_ID	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. The Shift ID. This value is optional. **Format:** Alphanumeric **Max Length:** 1 character.
LANGUAGE_CODE°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Set this to the language that is indicated by the Language Code that is encoded in the track II data on the customer's card. **Valid Values:** "English" or "French" (pass in the literal string)
KEY_SERIAL_NUM°°	String	If a Key Serial Number is returned from the PINpad, this property should be populated with that number. **If processing transactions with a PINpad using DUKPT encryption**, this value is sixteen or twenty characters long (depending on the processor's encryption). The **PC**Charge DevKit provides several tools for retrieving data from PINpads. If the **PC**Charge integration method chosen doesn't support these tools or the tools do not support the PINpad being used, a direct interface to the PINpad must be written by the integrator. **If processing transactions with a Verifone SC5000 PINpad**, set this property to the Chip Serial Number of the PINpad.
CASHBACK_AMNT	String	The amount of cash back that the customer will receive. This amount is in addition to value entered in TRANS_AMOUNT property. For example, if the total amount of the sale is $10 and the customer has requested $5 cash back, TRANS_AMOUNT should be set to $10 and CASHBACK_AMNT should be set to $5. The CASHBACK_AMNT property should be formatted the same the TRANS_AMOUNT property. **Max Length:** 9 characters. **Note:** Some debit processors do not support the cash back feature.

Tag	Data Type	Description - Debit Input File (.inx)
PIN_BLOCK° °°	String	The encrypted PIN block that is retrieved from the PINpad. The PIN is provided to the processor for verification. **Length:** 16 characters. The **PC**Charge DevKit provides several tools for retrieving data from PINpads. If the **PC**Charge integration method chosen doesn't support these tools or the tools do not support the PINpad being used, a direct interface to the PINpad must be written by the integrator.
MAC_BLOCK°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Set this to the MAC Block value returned by the PINpad.
DEBIT_TYPE°°	String	Only supported by Global Payments East (NDC) Canadian Debit and the Verifone SC5000 PINpad. Set this to the bank account type that the customer specified when entering transaction data into the PINpad. **Valid Values:** "Chequing" or "Savings"
ORIG_PURCH_DATA	String	The Original Purchase Data. Used when performing a Debit Return with the processors TSYS, Heartland, RBS WorldPay, and NPC. This is the original transaction date. **Format:** DDMMhhmm
BILLPAY	String	Only valid for Visa debit and credit transactions. It is used to indicate the transaction is being ran for payment of a bill (ultilty, monthly gym dues, etc.) **Valid values:** 0 – Non-Bill payment transaction 1 – Bill payment transaction
REFERENCE	String	NBS/ Fleet One cards require a Reference Number to be sent with each transaction. This is a minimum of 2 digits and a max of 15. This must be all numeric.
RFID	String	Set to 1 if card information was read from RFID (Radio Frequency Identification) device. If card was read from from RFID, track data must be populated and manual flag must be set to 1. Set to 0 otherwise.
CASHIER_NAME	String	Submit Cashier's login name.
PASSWORD	String	Submit Cashier's password.
PRODUCT_DETAIL_CODE_XX	String	**Note:** This is the number of items for RODUCT_DETAIL_PRODUCT_CODE_XX. PCCharge will support 1 – 10. • NBS supports using 1 – 10. • BAMS Tandem (NPC) supports using 1 – 3. **Example:** <PRODUCT_DETAIL_CODE_1>TESTLINE1</PRODUCT_DETAIL_CODE_1> <PRODUCT_DETAIL_CODE_2>TESTLINE2</PRODUCT_DETAIL_CODE_2> <PRODUCT_DETAIL_CODE_3>TESTLINE3</PRODUCT_DETAIL_CODE_3>

° These properties are required to process a Debit Sale transaction.

°° These properties are required to process a Canadian Debit Sale transaction using Global Payments East (NDC) and the SC5000 PINpad.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

*** If the "Use Default Processor" option is enabled in the **PC**Charge preferences, and the PROCESSOR_ID and MERCH_NUM properties are omitted from the transaction request, **PC**Charge will process all transactions using the "Default Processor". The "Default Processor" is defined as the first merchant number that is set up **PC**Charge. Consult the **Multi-Merchant Support** section (see page 68) for more information on the "Use Default Processor" option. In addition, PROCESSOR_ID and MERCH_NUM should not be set when doing follow-on transactions. Refer to the section **Follow On Transactions** (see page 70) for more information.

Debit Output File (.oux)

Tag	Data Type	Description - Debit Output File (.oux)
USER_ID	String	Returns the User name that is associated with the transaction. This value is echoed back from the original transaction. The User name will be in DOS file format, max 8 characters.

Tag	Data Type	Description - Debit Output File (.oux)
MERCH_NUM	String	Returns the merchant number that was specified in the MerchantNumber property.
TROUTD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PCC**harge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PCC**harge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
RESULT	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
AUTH_CODE	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
REFERENCE	String	Returns the reference number associated with the transaction. The reference number is used to help identify the transaction and is useful for the cardholder and merchant when doing research. This value is not returned with all transactions.
TRANS_DATE	String	Returns the date that the transaction was processed. This value is not returned by all processing companies.
TICKET	String	Returns the ticket number or invoice of the transaction. This value is echoed back from the original transaction or is generated by **PCC**harge if one is required to complete the transaction.
INTRN_SEQ_NUM	String	Returns the Internal Sequence Number, which is a **PCC**harge-assigned unique number for each transaction. This number is stored in the Number field in the **PCC**harge database (PCCW.MDB) for each transaction.
TRANS_ITEM_NUM	String	Returns the Transaction Item number or the number that is associated with the transaction in the settlement file. This value is not returned by all processing companies.
TBATCH	String	Returns the active batch number for the transaction. This value is not returned by all processing companies.
TRANS_ID	String	The TRANS_ID field returns the Working key (15 + 1 from TICode) that was provided by the processor. This field is only used for Master Session encryption, which is only supported by Elavon (NOVA).
TICODE	String	The TICODE field contains the last byte of the Working key that is provided by the processor.
TIM	String	Returns the Time of the transaction. This value is not returned by all processing companies.
NET_ID	String	Returns a one-character identification code that identifies the network on which the transaction was approved.
AUX_RESP_CODE	String	When using the SC5000 PINpad, returns the ISO response code

Check File Layouts

This section describes the tags required to process check transactions.

Check Input File (.inx)

Tag	Data Type	Description - Check Input File (.inx)
USER_ID° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62). **Note:** The value passed in USER_ID must match the name of the .inx file. For example: <USER_ID>User2</USER_ID> and User2.inx.
COMMAND°	String	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
TROUTD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
PROCESSOR_ID°	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
MERCH_NUM°	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Check Services Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
ACCT_NUM°°	String	For Check, MICR, or Double ID: The account number that will be used when processing the transaction. **Max Length:** 20 characters.
MANUAL_FLAG°	String	Flag that indicates whether the transaction was manually entered or swiped. **Valid values:** 0 = manual transaction, 1 = swiped transaction
TRANS_AMOUNT°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
PRINT_RECEIPTS_FLAG	String	The number of receipts that **PC**Charge should print for the transaction. This value will override the corresponding value in the **PC**Charge GUI. **PC**Charge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.
ZIP_CODE°°	String	The check writer's ZIP code. **Max Length:** 9 characters. **Format:** digits only. This value is required for COD transactions. **Note:** If submitting the 9-digit zip, do not include the dash.
TICKET_NUM	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.

Tag	Data Type	Description - Check Input File (.inx)
MULTI_FLAG	String	Flag that indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PC**Charge GUI. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid values:** 1 = TRUE, 0 = FALSE **Default value:** 0. See the section **Multi-trans Wait** for more information (see page 67). This Flag has no effect if processing will occur over IP or leased line.
TXN_TIMEOUT	String	The number of seconds after which a timeout error will be returned from **PC**Charge. The count will start when the transaction is submitted to **PC**Charge. An error will be returned if the transaction has not finished processing when the time period expires. It is highly recommended that integrators review the section **Timeouts** (see page 59). **Note:** This tag only works when using the TCP Interface.
CASHBACK_AMNT	String	The amount of cash back that the customer will receive. This amount is in addition to value entered in Amount property. For example, if the total amount of the sale is $10 and the customer has requested $5 cash back, Amount should be set to $10 and CashBack should be set to $5. The CashBack property should be formatted the same the Amount property. **Max Length:** 9 characters. **Note:** Some processors do not support the cash back feature.
CHECK_TYPE	String	**Valid Values:** 0 = Personal, 1 = Business **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
CUSTOMER_NAME	String	The first and last name of the customer. **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
CUSTOMER_CITY	String	The customer's city. **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
CHECK_READER_CODE	Enum	Passes the type of Check Reader that is being used. Currently only used by Telecheck and will only be set if TECK is the set processor. Cannot be configured in the PCCharge GUI. **Valid Values:** 001 - Magtek mini micr 002 - EnCheck 3000 003 - IVI 2500 004 - IVI 430 005 - IVI 431 006 - ICE 5700 007 - MagtekImager 008 - VeriFone CR1000i 009 - Epson - TMH6000 010 - Epson - TMH6000Imager 011 - WelchAllyn ScanTeam 8300 012 - VeriFone CR600 013 - Magtek Imager with Modem 014 - IBM 4610 reader/printer 015 - Ingenico EC2600 016 - RDM EC5000 017 - RDM EC6000 018 - NCR 7158 and 7167 019 - LS 100 020 = MagTek Excella 021 = MagTek Excella (DL Capture & F&B check images) 022 = VeriFone Model Quartet
CUSTOMER_STREET	String	The street address of the customer. **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
MICR	String	The raw MICR data from the bottom of the check. Used for conversion transactions.
MICR_READER_STATUS	String	**Valid Values:** 15 = Valid read by MICR reader, 15I = Valid read by MICR reader with imaging capability, 9 = Manual only **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
STATE°°	String	The state code of the state that issued the check writer's driver's license. The state code is required for DL (Driver's License). **Format:** 2 characters.
LICENSE°°	String	The driver's license number of the individual writing the check. **Max Length:** 20 characters. The driver's license is required for DL (Driver's License) transactions and when performing Double ID transactions.

Tag	Data Type	Description - Check Input File (.inx)
DL_TRACK_II	String	The parsed TrackII data from the driver's license. **Note**: Used only for processor TECK. Cannot be accessed in the PCCharge GUI.
ABA_NUM°°	String	The Transit Routing Number / ABA number that will be used when processing the transaction. This value indicates which bank issued the check. **Max Length:** 9 characters. This value is required for MICR transactions and when performing Double ID transactions.
PHONE_NUM°°	String	The phone number of the individual writing the check. **Max Length:** 7 digits. **Format:** digits only. The phone number is required for COD (Checks On Delivery).
DOB°°	String	The date of birth of the check writer. **Length:** Exactly six characters. **Format:** MMDDYY. The birth date is required for DL (Driver's License) check transactions.
CHECK_NUM°	String	The check number of the check that will be used when processing the transaction. **Max Length:** 10 characters.
MANAGER_NUM	String	Manager Number
CASHIER_NUM	String	The Cashier Number
CASHIER_NAME	String	Submit Cashier's login name.
PASSWORD	String	Submit Cashier's password.

Note: To perform Double ID, both the MICR and LICENSE fields must be populated.

° These properties are required, regardless of service type.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

°° COD -- required for Checks-On-Delivery
DL -- required for Driver's License
MICR -- required for MICR

Check Output File (.oux)

Tag	Data Type	Description - Check Output File (.oux)
USER_ID	String	Returns the User name that is associated with the transaction. This value is echoed back from the original transaction. The User name will be in DOS file format, max 8 characters.
MERCH_NUM	String	Returns the merchant number that was specified in the MerchantNumber property.
TROUTD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
TRANS_ID	String	**Only for TECK.** Returns the Trace ID associated with the transaction.
RESULT	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
RETURN_CHECK_FEE	String	Returns the response from the processor which indicates the fee for returned checks. **Note**: Only used for the processor TECK
RETURN_CHECK_NOTE	String	Returns the response from the processor which displays a note for returned checks. **Note**: Only used for the processor TECK
AUTH_CODE	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
TICKET	String	Returns the ticket number or invoice of the transaction. This value is echoed back from the original transaction or is generated by **PC**Charge if one is required to complete the transaction.

Tag	Data Type	Description - Check Output File (.oux)
INTRN_SEQ_NUM	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
PROC_RESP_CODE	String	Returns the response code that is provided by the processor. This value is not returned by all processing companies.

EBT File Layouts

This section describes the tags required to process EBT transactions.

When processing EBT cards, a PINpad is required to allow the customer to enter their PIN. In addition, debit card information is always collected via a card swipe device, never via keyboard entry. Because of this, a card reader is also required. (Some EBT transactions can be manually entered).

When processing EBT card transactions, merchants have the option of allowing the customer to receive cash back on a transaction. For instance, the customer purchases $50 of products and wants $25 cash back, set the Amount to 50.00 and CashBack to 25.00. This will withdraw a total of $75 from the EBT card account, $50 for the products and $25 for cash to give to the customer.

EBT Input File (.inx)

Tag	Data Type	Description - EBT Input File (.inx)
USER_ID° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62). **Note:** The value passed in USER_ID must match the name of the .inx file. For example: <USER_ID>User2</USER_ID> and User2.inx.
COMMAND°	String	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
TROUTD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
PROCESSOR_ID°	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
MERCH_NUM°	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **EBT Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
ACCT_NUM°	String	The EBT card number that will be used when processing the transaction. **Max Length:** 20 characters. **Example:** 5424180279791765
EXP_DATE	String	The expiration date associated with the EBT card number that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208 Set this property if there is an expiration date associated with the EBT card.
MANUAL_FLAG°	String	Flag that indicates whether the transaction was swiped or manually entered. This property must be set to 1 (swiped) for swiped EBT transactions. If the transaction was swiped, the TRACK_DATA property must also be set. If performing a manually keyed EBT transaction, such as a Force or Voucher, set this property to 0 (manually entered).

Tag	Data Type	Description - EBT Input File (.inx)
TRANS_AMOUNT°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00".
TRACK_DATA°	String	The track II data captured from the magnetic strip of the card. The track II data is required for swiped EBT transactions. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in.
PRINT_RECEIPTS_FLAG	String	The number of receipts that **PC**Charge should print for the transaction. This value will override the corresponding value in the **PC**Charge GUI. **PC**Charge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.
ZIP_CODE	String	The ZIP_CODE stores the reference number from the original transactions. Some processor all a type of void and for these transactions a reference number must be provided. For Chase Paymentech EBT, the auth code goes in the zip field for voucher transactions.
TICKET_NUM	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all processors support alphanumeric characters. **Note:** When using NDC, lower case characters must not be used in the ticket number. **Note:** When using Elavon (NOVA), ticket numbers can only be alphanumeric, no hyphens.
CARDHOLDER	String	The cardholder's name. **Max Length:** 20 characters.
EBT_TYPE	String	Indicates what type of EBT transaction will be performed. **Valid Values:** F – Food stamp transaction; C – Cash benefits transaction
TXN_TIMEOUT	String	The number of seconds after which a timeout error will be returned from **PC**Charge. The count will start when the transaction is submitted to **PC**Charge. An error will be returned if the transaction has not finished processing when the time period expires. It is highly recommended that integrators review the section **Timeouts** (see page 59). **Note:** This tag only works when using the TCP Interface.
AUTH_CODE	String	For an EBT Post (Prior Auth Sale) or Force transaction: The Authorization code from the original voice authorization.
KEY_SERIAL_NUM	String	If a Key Serial Number is returned from the PINpad, this property should be populated with that number. This value is only applicable for PINpads using DUKPT encryption. This value is sixteen or twenty characters long (depending on the processor's encryption). The **PC**Charge DevKit provides several tools for retrieving data from PINpads. If the **PC**Charge integration method chosen doesn't support these tools or the tools do not support the PINpad being used, a direct interface to the PINpad must be written by the integrator.
CASHBACK_AMNT	String	The amount of cash back that the customer will receive. This amount is in addition to value entered in Amount property. For example, if the total amount of the sale is $10 and the customer has requested $5 cash back, Amount should be set to $10 and CashBack should be set to $5. The CashBack property should be formatted the same the Amount property. **Max Length:** 9 characters. **Note:** Some debit processors do not support the cash back feature.

Tag	Data Type	Description - EBT Input File (.inx)
PIN_BLOCK°	String	The encrypted PIN block that is retrieved from the PINpad. The PIN is provided to the processor for verification. **Length:** 16 characters. The **PC**Charge DevKit provides several tools for retrieving data from PINpads. If the **PC**Charge integration method chosen doesn't support these tools or the tools do not support the PINpad being used, a direct interface to the PINpad must be written by the integrator.
EBT_VOUCHER_NUM	String	The voucher number for an EBT force transaction. The voucher is provided by the processor at the time of authorization and must be supplied to clear the voucher.
RFID	String	Set to 1 if card information was read from RFID (Radio Frequency Identification) device. If card was read from from RFID, track data must be populated and manual flag must be set to 1. Set to 0 otherwise.
PRODUCT_DETAIL_CODE_XX	String	**Note:** This is the number of items for RODUCT_DETAIL_PRODUCT_CODE_XX. PCCharge will support 1 – 10. • BAMS Tandem (NPC) supports using 1 – 3. Example: <PRODUCT_DETAIL_CODE_1>TESTLINE1</PRODUCT_DETAIL_CODE_1> <PRODUCT_DETAIL_CODE_2>TESTLINE2</PRODUCT_DETAIL_CODE_2> <PRODUCT_DETAIL_CODE_3>TESTLINE3</PRODUCT_DETAIL_CODE_3>

° These fields are required to process a transaction.

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

EBT Output File (.oux)

Tag	Data Type	Description - EBT Output File (.oux)
USER_ID	String	Returns the User name that is associated with the transaction. This value is echoed back from the original transaction. The User name will be in DOS file format, max 8 characters.
MERCH_NUM	String	Returns the merchant number that was specified in the MerchantNumber property.
TROUTD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
RESULT	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
AUTH_CODE	String	For approved transactions, returns the authorization code from the issuing bank. For declined transactions, returns the reason why the transaction was declined (if the issuing bank provides one) or why the transaction was rejected.
REFERENCE	String	Returns the reference number associated with the transaction. The reference number is used to help identify the transaction and is useful for the cardholder and merchant when doing research. This value is not returned with all transactions.
TRANS_DATE	String	Returns the date that the transaction was processed. This value is not returned by all processing companies.
TICKET	String	Returns the ticket number or invoice of the transaction. This value is echoed back from the original transaction or is generated by **PC**Charge if one is required to complete the transaction.
INTRN_SEQ_NUM	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
TRANS_ITEM_NUM	String	Returns the Transaction Item number or the number that is associated with the transaction in the settlement file. This value is not returned by all processing companies.

Tag	Data Type	Description - EBT Output File (.oux)
TBATCH	String	Returns the active batch number for the transaction. This value is not returned by all processing companies.
TRANS_ID	String	The TRANS_ID field returns the Working key (15 + 1 from TICode) that was provided by the processor. This field is only used for Master Session encryption, which is only supported by Elavon (NOVA).
TICODE	String	The TICODE field contains the last byte of the Working key that is provided by the processor.
TIM	String	Returns the Time of the transaction. This value is not returned by all processing companies.
NET_ID	String	Returns a one character identification code that identifies the network on which the transaction was approved.
EBT_FOOD_BALANCE	String	Returns the remaining balance on a Food Stamp card. This value is not returned by all processing companies.
EBT_CASH_BALANCE	String	Returns the remaining balance on a Cash Benefits card. This value is not returned by all processing companies.
CASHIER_NAME	String	Submit Cashier's login name.
PASSWORD	String	Submit Cashier's password.

Gift File Layouts

This section describes the tags required to process Gift/Loyalty transactions.

Gift Input File (.inx)

Tag	Data Type	Description - Gift Input File (.inx)
USER_ID° **	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62). **Note:** The value passed in USER_ID must match the name of the .inx file. For example: <USER_ID>User2</USER_ID> and User2.inx.
COMMAND°	String	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
TROUTD	String	The TroutD (Transaction Routing ID) is used when performing "Follow On" transactions. The TroutD is a **PC**Charge-assigned unique identifier that will be associated with a transaction and any subsequent transactions related to it. This property must be set when performing Follow-on Transactions. Review the section **Follow On Transactions** (see page 70) for important information on implementing TroutD support.
TICKET	String	The ticket or invoice number for internal referencing by merchant. This value is stored by **PC**Charge and passed to the processor for referencing purposes. **Max Length:** 9 characters. The value can be alphanumeric. **Note:** Not all gift processors support ticket numbers.
PROCESSOR_ID°	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
MERCH_NUM°	String	The Merchant Number. This number is issued to the merchant by the Processor or the Merchant Services Provider. The value set in this property must match what is set up in the **Gift Card Setup** window of **PC**Charge. **Max Length**: 32 characters. This value can be alphanumeric.
ACCT_NUM°	String	The gift card number that will be used when processing the transaction. **Max Length:** 20 characters.
EXP_DATE	String	The expiration date associated with the gift card that will be processed. Must be exactly four characters long. **Format:** MMYY **Example:** 1208 **Note:** Most gift cards do not have an expiration date.
MANUAL_FLAG°	String	Flag that indicates whether the transaction was manually entered or swiped. If the transaction was swiped, the TRACK_DATA property must also be set. **Valid values:** 0 = manual transaction, 1 = swiped transaction
TRANS_AMOUNT°	String	The amount of the transaction. **Format:** DDDDDD.CC. **Max Length:** 9 characters, including the decimal. The value may not be negative. Do not use commas. **Note:** The amount MUST include the decimal point and the cents even if the amount is a whole dollar amount. **Example:** "3.00", not "3" or "3.". If sending less than one dollar, the zero place holder must be sent as well. **Example:** "0.50". If the amount is set to an incorrect format, the Error event will fire after calling the Send method. Consult the section **System Error Codes and Descriptions** for a list of valid errors that will be returned (see page 150). **Note:** Only add a zero to the front of the amount in the case of a cents-only transaction. An error will be returned if a dollar amount is padded with leading zeroes. **Example:** Incorrect format "0001.00". For Valuelink (VLNK) Balance Adjustment: **Format:** +/-DDDDD.CC.
TRACK_DATA	String	The track II data captured from the magnetic strip of the card.. **Max Length:** 40 characters. **Example:** 5424180279791765=08121011000001234567 **Note:** The characters that are appended to the beginning and ending of track II (usually ; and ?) should not be passed in.

Tag	Data Type	Description - Gift Input File (.inx)
PRINT_RECEIPTS_FLAG	String	The number of receipts that **PC**Charge should print for the transaction. This value will override the corresponding value in the **PC**Charge GUI. **PC**Charge will retain this value for subsequent transactions. **Valid values:** 0-9. Setting the property to 0 will disable receipt printing.
MULTI_FLAG	String	Flag that indicates whether **PC**Charge should leave the modem connection open in anticipation of other transactions that will follow shortly. If set, this value will override the corresponding value in the **PC**Charge GUI. Note that **PC**Charge can only keep the connection open as long as is allowed by the processing company. **Valid values:** 1 = TRUE, 0 = FALSE **Default value:** 0. See the section **Multi-trans Wait** for more information (see page 67). This Flag has no effect if processing will occur over IP or leased line.
GRATUITY_AMNT	String	The tip amount if it is a VTEC or VLNK restaurant transaction.
PROMO_CODE	String	Used for GVEX: A code defined by the merchant that affects the calculation from amount and units to points.
CASHIER_NUM	String	The numeric Cashier ID for VTEC and VLNK processors.
IND_TYPE°°	String	For VTEC: Flag indicating the industry. **Valid Values:** 1 = retail, 2 = restaurant
GIFT_PIN	String	Used for the pin for the processor **SVS**. Used for only for virtual gift card transactions. With **Paymentech Gift**, this tag is used to send the **CVD2** value.
GIFT_UNITS	String	The Units for points transactions.
GIFT_SEQ_NUM	String	Stores the card sequence number for GSAR transactions.
TOT_NUM_CARDS	String	the total number of cards for multiple issuance for Chase Paymentech. The refund amount for VTEC's deactivate transaction.
SOURCE_ACCT_NUM	String	For VTEC replace transaction, VLNK Balance Merge and Balance Transfer, the field should be set to the account number of the old card.
FORCE_FLAG	String	Flag indicating whether the transaction should be forced for Chase Paymentech. Valid Values: 1 – force, 0 – don't force
PARTIAL_REDEMPTION_FLAG	String	For GSAR: Flag indicating whether the transaction is a partial redemption transaction.
LOYALTY_FLAG	String	Flag indicating whether the transaction is a loyalty transaction. **Valid Values:** 1 – True 0 - False
REFUND_FLAG	String	Flag that indicates whether to provide the customer a refund when performing a VTEC Deactivate transaction. **Valid Values:** 1 – Provide refund 0 – Do not provide refund **Note:** This flag should only be set to 1 for a Valutec deactivate if the LOYALTY_FLAG is set to 0. Valutec does not support deactivate with refund for loyalty cards.
RFID	String	Set to 1 if card information was read from RFID (Radio Frequency Identification) device. If card was read from from RFID, track data must be populated and manual flag must be set to 1. Set to 0 otherwise.
VIRTUAL_GIFTCARD_FLAG	Boolean	Only used for the processor **SVS**. 0 - False, 1 - True – Only sent on an activation to determine if a pin should be returned.
CASHIER_NAME	String	Submit Cashier's login name.
PASSWORD	String	Submit Cashier's password.
SKU_LOYALTY	String	**GIVEX ONLY – Integration Only** Pass this tag in order to send product codes, quantities and amounts for loyalty transactions. **Format:** **Single Product Code:** <SKU_LOYALTY>Product Code,Amount,Quantity</SKU_LOYALTY> **Multi-Product Codes (Separated with a semi-colon):** <SKU_LOYALTY>Product Code,Amount,Quantity; Product Code,Amount,Quantity</SKU_LOYALTY>

° These properties are required to process a gift card redemption or sale transaction.

°° Required for VTEC gift card transactions

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

Gift Output File (.oux)

Tag	Data Type	Description - Gift Output File (.oux)
USER_ID	String	Returns the User name that is associated with the transaction. This value is echoed back from the original transaction. The User name will be in DOS file format, max 8 characters.
MERCH_NUM	String	Returns the merchant number that was specified in the MerchantNumber property.
TROUTD	String	Returns the TroutD (Transaction Routing ID) for the transaction. The TroutD is a **PC**Charge-assigned unique identifier that is associated with the transaction throughout its "lifespan". This number is stored in the TroutD field in the **PC**Charge database (PCCW.MDB) for each transaction. See the section **Follow On Transactions** (see page 70) for more information.
RESULT	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
AUTH_CODE	String	The Auth Code field serves many purposes. For a GVEX **Balance** transaction returns the balance remaining on a gift card. For all other GVEX transactions, returns the transaction reference/Error message. For VTEC, returns the Auth Code. **For a VTEC Batch function:** returns the number of sales done that day and the total amounts of sales in the following format <# of transaction>, <amount>. For a VLNK transaction returns the authorization code or transaction response description.
REFERENCE	String	The Reference field serves many purposes. For a GVEX **Register** transaction returns the first eleven digits of an account number. For a VTEC batch function, returns the number of activations done that day and the total amounts of activations in the following format <# of transaction>, <amount>.>. For all other VTEC transactions, returns the account's remaining balance. For a VLNK transaction returns the previous balance (balance before transaction was applied). For a BPS **Redemption** transaction, returns the retrieval reference number.
TICKET	String	For a VTEC batch function, returns the account's **Deactivates**. Data is returned in the format <# of transactions>, <amount>. For all other VTEC transactions, returns the account's remaining loyalty points. For a GSAR transaction returns the trace number. For a VLNK **Activation** or **Reload** transaction, returns the redemption amount. For a VLNK **Redemption**, **Redemption Unlock**, or **Cash-out** transaction, returns the cashback amount. **For a VTEC batch function: returns** the number of gift card that has been de-activated that day and the total amounts of de-activations in the following format <# of transaction>, <amount>.>.
INTRN_SEQ_NUM	String	Returns the Internal Sequence Number, which is a **PC**Charge-assigned unique number for each transaction. This number is stored in the Number field in the **PC**Charge database (PCCW.MDB) for each transaction.
TRANS_ID	String	For a GVEX **Register** transaction, the remaining digits of a gift card number. For a GVEX **Redemption**, **Increment**, and **Cancel**, the balance remaining on a gift card. For a VTEC batch function, the number Add Value Transactions done that day and the total amounts of Add Value in the following format <# of transaction>, <amount>.>. For a VLNK or GSAR transaction, returns the previous balance (balance after transaction was applied).
RET	String	For VLNK, returns trace number (generated by VLNK host). For GVEX, returns the point balance. **For a VTEC batch function:** returns the number of Gift Transactions Voids performed that day.
ACTIVATION_COUNT	String	Returns the number of activations in the current batch
ACTIVATION_TOTAL_AMOUNT	String	Returns the total dollar amount of activations in the current batch
ADDPOINTS_COUNT	String	Returns the number of AddPoints Transactions in the current batch
ADDPOINTS_TOTAL_AMOUNT	String	Returns the total dollar amount of AddPoints transactions in the current batch
ADDVALUE_COUNT	String	Returns the number of AddValue transactions in the current batch
ADDVALUE_TOTAL_AMOUNT	String	Returns the total dollar amount of AddValue transactions in the current batch

Tag	Data Type	Description - Gift Output File (.oux)
AMOUNT_DUE	String	Used in partial redemption transactions where only part of the amount was authorized. Returns the remainder amount that is owed to the merchant.
AUTH_AMOUNT	String	Used in partial redemption transactions where only part of the amount was authorized. Returns the actual authorized amount.
BALANCE_TRANSFER_COUNT	String	Returns the number of Balance Transfers in the current batch
BALANCE_TRANSFER_TOTAL_AMOUNT	String	Returns the total dollar amount of Balance Transfers in the current batch
CASHBACK	String	Used in redemption for remaining balance transactions where the transaction amount is so close to the balance of the card that the entire balance is authorized. Returns the remainder that is owed to the customer.
CREDIT_COUNT	String	Returns the number of credits in the current batch
CREDIT_TOTAL_AMOUNT	String	Returns the total dollar amount of credits in the current batch
GIFT_CARD_BALANCE	String	Returns the gift card balance.
GIFT_PIN	String	Returns the gift pin. Used only for virtual gift cards.
POINTS_COUNT	String	Returns the number of points transactions in the current batch
POINTS_TOTAL_AMOUNT	String	Returns the total dollar amount of points transactions in the current batch
PROC_RESP_CODE	String	Returns the processor response code
SALE_COUNT	String	Returns the number of redemptions in the current batch
SALE_TOTAL_AMOUNT	String	Returns the total dollar amount of redemptions in the current batch
TIP_COUNT	String	Returns the number of Tip transactions in the current batch
TIP_TOTAL_AMOUNT	String	Returns the total dollar amount of Tip transactions in the current batch
TRANS_DATE_TIME	String	Returns the transaction date and time when passed back by a processor.
VOID_BALANCE	String	Returns the Void Balance
VOID_COUNT	String	Returns the number of voids in the current batch
VOID_TOTAL_AMOUNT	String	Returns the total dollar amount of Voids in the current batch
LEVEL	String	Returns the customer's loyalty level. Only used for VTEC loyalty gift cards.
<PRE_AUTH_COUNT>	String	Only for GAPI, this returns the total number of gift card pre-auth transactions processed that day.
<PRE_AUTH_TOTAL_AMOUNT>	String	Only for GAPI , this returns the total amount of gift card pre-auth transactions processed that day
<POST_AUTH_COUNT>	String	Only for GAPI, this returns the total number of gift card post-auth transactions processed that day.
<POST_AUTH_TOTAL_AMOUNT>	String	Only for GAPI, this returns the total amount of the post-auth transactions processed that day.
<ISSUANCE_COUNT>	String	Only for GAPI, this returns the total number of gift cards issued that day.
<ISSUANCE_TOTAL_AMOUNT>	String	Only for GAPI, returns the total amount of the gift cards issued that day.
<DEACTIVATE_COUNT>	String	Only for GAPI, this returns how many gift cards where deactivated that day.
<DEACTIVATE_TOTAL_AMOUNT>	String	Only for GAPI, this returns the total amount of gift card deactivations that day.
<BALANCE_ADJUST_COUNT>	String	Only for GAPI, this returns the number of gift cards that were balance adjusted that day.
<BALANCE_ADJUST_TOTAL_AMOUNT>	String	Only for GAPI, this returns the total amount of balance adjustments on gift cards that day.
<BALANCE_MERGE_COUNT>	String	Only for GAPI, this returns the total number of gift cards that were balance merged that day.
<BALANCE_MERGE_TOTAL_AMOUNT>	String	Only for GAPI, this returns the total amount of gift card balance merges that day.
<REPORT_LOST_STOLEN_COUNT>	String	Only for GAPI, returns the total reported stolen or lost gift cards that day.
<REPORT_LOST_STOLEN_TOTAL_AMOUNT>	String	Only for GAPI, returns the total amount of all stolen or reported lost gift cards that day.
<CASHOUT_TOTAL_AMOUNT>	String	Only for GAPI, returns the total amount of all cashout transactions processed that day.
<CASHOUT_COUNT>	String	Only for GAPI, returns the total number of the cashout transactions processed that day.

Tag	Data Type	Description - Gift Output File (.oux)
<REACTIVATE_COUNT>	String	Only for GAPI, returns the total number of gift cards that have been reactivated that day.
<REACTIVATE_TOTAL_AMOUNT>	String	Only for GAPI, the total amount of all gift cards that have been reactivated that day.

VeriFone Stored Value API (GAPI)

The VeriFone Stored Value API (GAPI) is a proprietary specification that allows for stored value card processors to add themselves to **PC**Charge. Applications using GAPI can also integrate with **PC**Charge using the various integration methods. For more information on adding a stored value card processor to **PC**Charge, and how to obtain the VeriFone Stored Value API, please contact VeriFone sales at 1-800-725-9264.

Batch File Layouts

This section describes the tags required to perform batch/settlement functions.

Batch Input File (.inx)

Tag	Data Type	Description - Batch Input File (.inx)
USER_ID**	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62). **Note:** The value passed in USER_ID must match the name of the .inx file. For example: <USER_ID>User2</USER_ID> and User2.inx.
COMMAND	String	The action code that identifies what type of transaction will be performed. Consult the section **DevKit Constants** for a list of valid values (see page 143).
PROCESSOR_ID	String	The code for the processing company that will be used to process the transaction. This value can be no more than four characters and must be capitalized. The processor specified in this property must be set up with a valid merchant number in **PC**Charge. A list of valid processor codes are listed in the **Processing Company Codes** section (see page 152).
MERCH_NUM	String	Sets the Merchant Number issued by the processor that identifies the account. The merchant number must be setup in the Credit Card Setup section of **PC**Charge.
SPLIT_PROCESSOR_FLAG	String	Only used when settling the processor CITI for private label transactions. Set this property to the main credit card processor ID code being used.
TXN_TIMEOUT	String	The number of seconds after which a timeout error will be returned from **PC**Charge. The count will start when the settlement is submitted to **PC**Charge. An error will be returned if the settlement has not finished processing when the time period expires. It is highly recommended that integrators review the section **Timeouts** (see page 59). **Note:** This tag only works when using the TCP Interface.
BATCH_CLOSE_TYPE	String	Flag that determines what type of batch close will occur. This flag only supported by FDMS Atlanta and Fifth-Third when using action code 30 or 31 **Valid values:** 1 – Standard End of Day Batch Close (Default) 2 – Shift Close 3 – Fifth-Third Terminal Based Batch Close of Debit, EBT, or Gift
CASHIER_NAME	String	Submit Cashier's login name.
PASSWORD	String	Submit Cashier's password.
BATCH_NUMBER	String	Submit batch number that was returned on the AMEX split dial settlement, needs to be passed when performing the following Action Codes: 35 – Reverse Batch 36 – Resubmit Batch 37 – Get Results 38 – Finalize Batch
CARD_TYPE	String	Submit with a Batch Inquiry to receive summary results based on Card Type. Valid Values: MC – MasterCard VISA – Visa AMEX – American Express DISC – Discover Card DEBIT – Debit Card EBT – Electronic Benefits Card PL – Private Label JCB – JCB Card

** The user name is used to keep the transaction associated with the correct terminal. It is highly recommended that integrators review the **Multi-User Support** section (see page 62). This section contains detailed information about user names and how they should be implemented.

Batch Output File (.oux)

Tag	Data Type	Description - Batch Output File (.oux)
USER_ID	String	Returns the User name that is associated with the transaction. This value is echoed back from the original transaction. The User name will be in DOS file format, max 8 characters.
MERCH_NUM	String	Returns the merchant number that was specified in the MerchantNumber property.
RESULT	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.
AUTH_CODE	String	The AUTH_CODE field returns the status of the batch. If the batch was successfully closed this status will indicate that, if the batch was not closed and there was a problem, it will return the response from the processor. For example, if TSYS sends back a rejected batch response this method will return the RB# number that TSYS returns.
DEBIT_COUNT	String	The number of debit transactions in the batch. This value is not returned by all processing companies.
DEBIT_TOTAL_AMOUNT	String	The total amount of the debit transactions in the batch. This value is not returned by all processing companies.
REFERENCE	String	The reference field returns the response from the processor if the response is a declined. This field is not returned for all processors.
TRANS_DATE	String	The Transaction Date field returns the error code from the processor if one is provided for a declined transaction. All processors do not support this field.
INTRN_SEQ_NUM	String	The INTRN_SEQ_NUM field returns the dollar amount that was settled in the batch or is waiting to be settled in the open batch.
TRANS_ID	String	The Trans Identifier field returns the Settlement number that is stored in association with the transaction in **PC**Charge's database. This number is not returned from the processor but is **PC**Charge's internal sequencing number scheme.
TICODE	String	The TICode field returns the number of transactions that are in the batch for which the function was performed.
RET	String	The RET field is only used for Inquiries, Action 30, and returns the number of batches that will be settled for a particular merchant number. For Example, if a merchant account is setup as TSYS using TCP/IP, there is a limitation on how many transactions can be sent across on a single batch, so **PC**Charge breaks the batch up into smaller batches. The RET field returns the number of smaller batches that would be created for that merchant account.
PROC_RESP_CODE	String	Returns the response code that is provided by the processor. This response code is not provided for all credit card processors.
RESULT_CODE	String	Returns a numerical representation of the result of the transaction.
BATCH_NUMBER	String	After a terminal-based batch settles, returns the batch number.
RECORD_COUNT	String	
TERMINATION_STATUS	String	Termination Status returns a "6" upon successful settlement. If anything other than a "6" is returned there may be an issue with the settlement file and should be investigated. This applies to : **GSAR**, **Elavon (NOVA)**, and **NBS** **Response:** **6 = Batch Settled and file has been deleted.** **8 = Batch Settled, but the file is locked and cannot be deleted.**
EBT_COUNT	String	Number of EBT Transactions
EBT_AMOUNT	String	Total Dollar Amount of EBT Transactions
AMEX_COUNT	String	Number of AMEX Transactions
AMEX_AMOUNT	String	Total Dollar Amount of AMEX Transactions
TOTAL_COUNT	String	Number of Transactions
TOTAL_BALANCE	String	Total Dollar Amount of Transactions in the batch
VISA_COUNT	String	Number of VISA Transactions
VISA_AMOUNT	String	Total Dollar Amount of VISA Transactions
MC_COUNT	String	Number of MC Transactions

Tag	Data Type	Description - Batch Output File (.oux)
MC_AMOUNT	String	Total Dollar Amount of MC Transactions
DISC_COUNT	String	Number of DISC Transactions
DISC_AMOUNT	String	Total Dollar Amount of DISC Transactions
PL_COUNT	String	Number of PL Transactions
PL_AMOUNT	String	Total Dollar Amount of PL Transactions
JCB_COUNT	String	Number of JCB Transactions
JCB_AMOUNT	String	Total Dollar Amount of JCB Transactions

Note: In the event there are multiple batches waiting to be settled in one settlement, the integrated application will need to be designed to loop through the settlement response to retrieve the response for each batch.

Report File Layouts

This section describes the tags required to submit report requests.

Report Input File (.inx)

Tag	Data Type	Description - Report Input File (.inx)
USER_ID°	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62). **Note:** The value passed in USER_ID must match the name of the .inx file. For example: <USER_ID>User2</USER_ID> and User2.inx.
COMMAND°	String	The action code that identifies what type of report will be requested. **Valid Values:** 81-84. **Example:** If running a credit card detail report, set the action code to "81". Consult the section **DevKit Constants** for a list of valid values (see page 143).
MERCH_NUM	String	**Merchant Number filter**. Set this property to filter the report by the merchant number specified. Setting this property will generate a report consisting of only those transactions processed via the merchant number specified. To generate a report that includes all merchant numbers in **PC**Charge, set this property to "ALL" or leave blank. **Example:** "99999999911"
ACCT_NUM	String	**User name filter**. If a valid user name is set in the ACCT_NUM property, the report will be filtered by that user name. The report returned will consist of only those transactions processed by the user name specified. **Example:** "User1". If this property is left blank, the report will show transactions processed by all users.
MANUAL_FLAG	String	**Result filter**. Use this filter to create a report consisting of only those transactions with the result specified. **Valid Values:** 0 = all (default), 1 = approved, 2 = declined **Example:** 1
TRACK_DATA	String	**Destination Directory for Report File**. Specifies the destination directory where the report file will be generated by **PC**Charge (if PERIODIC_PAYMENT_FLAG is set to "1"). **Example:** "C:\My Documents\PCCReports\" **Path Formats:** UNC, MS-DOS(8 Characters) and Long. **Max Length:** 40 characters (if the Destination Directory is longer than 40 characters, use CUSTOMER_CODE for the additional characters. Must end with a "\" unless the directory name will be continued in the CUSTOMER_CODE property. **Note:** If running in a Client/Server environment, this property is the path from the server running **PC**Charge, not the client. For example, if a client submitted a report request that specified "C:\" as the destination directory, the report would be written to the local hard drive of the server running **PC**Charge, not to the client's hard drive.
CUSTOMER_CODE	String	**Destination Directory for Report File (continued)**. Continuation of the destination directory (if the directory name is greater than 40 characters). **Max Length:** 25 characters. Must end with a "\"
PERIODIC_PAYMENT_FLAG	String	**Report Output setting**. Determines if the report will be printed by **PC**Charge or written to a file. **Valid Values:** "0" = print to default printer specified in **PC**Charge (default). "1" = print to file using filename specified in TRANS_ID and path specified in TRACK_DATA.
STREET	String	**Starting Date/Time Filter** (Optional) Specifies the start date and start time of the report. **Format**: Date: MM/DD/YY Time: HH:MM:SS PM. Use to create a report consisting of only those transactions processed on or after the date specified. If a start date is not specified, today's date is assumed. If a start time is not specified, 12:00:00 AM is assumed. The start date can be passed without the start time. However, the start time cannot be passed without the start date. **Examples:** "03/04/05 09:00:00 AM" or "03/04/05"

Tag	Data Type	Description - Report Input File (.inx)
CARDHOLDER	String	**Ending Date/Time filter**. Specifies the end date and end time of the report. **Format**: Date: MM/DD/YY Time: HH:MM:SS PM. When used in conjunction with `Street`; will create a report consisting of only those transactions processed between the start and end date/time specified (inclusive). If an end date is not specified, today's date is assumed. If an end time is not specified, 11:59:59 PM is assumed. The end date can be passed without the end time. However, the end time cannot be passed without the end date. **Examples:** `"07/06/05 06:00:00 PM" or "07/06/05"`
TRANS_ID	String	**Report File name/Report File Type**. Specifies the filename and extension of the report file generated by **PC**Charge (if `PERIODIC_PAYMENT_FLAG` is set to `"1"`). Also determines what file type will be used when **PC**Charge generates the report. To specify the file type, set the extension to one of the following: **`.pdf`** – Create the report file in the Portable Document Format. **Ex.** `Report.pdf` **`.rtf`** – Create the report file in Rich Text. **Ex.** `Report.rtf` **`.txt`** – Create a report file in flat text. **Ex.** `Report.txt` **Default: .txt** (If an extension other than the ones listed is passed, the report will be returned as flat text and a .txt extension will be added to the filename)

° These properties are required to submit a report request.

Report Output File (.oux)

Tag	Data Type	Description - Report Output File (.oux)
USER_ID	String	Returns the User name that is associated with the transaction. This value is echoed back from the original transaction. The User name will be in DOS file format, max 8 characters.
RESULT	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section (see page 156) for a list of valid values and descriptions.

Configuration File Layouts

This section describes the tags required to configure various settings. Currently, only configuration of the Transaction Archive (pccw.mdb database backup) is supported.

Configuration Input File (.inx)

Tag	Data Type	Description - Configuration Input File (.inx)
COMMAND°	String	The action code that identifies what type of configuration command will be performed. First byte is always a Z. Consult the section **DevKit Constants** for a list of valid values (see page 143).
CFG_TYPE°	Long	Configuration type. **Valid value:** • 0 = CFG_TXN_ARCHIVE = Configure Transaction Archive. Use action code ZC.
CFG_ENABLED	String	Enable or disable current configuration (1 = Enable, 0 = Disable).
CFG_PATH	String	Specify path for saved output files (**Example:** backed up transaction database). Must end with a backslash "\".
CFG_SIZE_LIMIT	String	Transaction archive size limit for GUI archive prompting and validation. Specified in megabytes.
CFG_KEEP_DAYS	String	Transaction archive preservation range. All transactions within the past number of "keep days" will remain in the pccw.mdb database following a transaction archive command.

° These properties are required to submit a configuration command.

Configuration Output File (.oux)

Tag	Data Type	Description - Configuration Output File (.oux)
RESULT	String	Result of configuration command. **Responses:** • CONFIG SUCCESS = Configuration updated successfully. • CONFIG FAILURE = Could not update configuration. • INVALID TYPE = Unsupported configuration type.

Various Utility File Layouts

This section describes the tags required to perform various functions.

Transaction Archive Input File (.inx)

Tag	Data Type	Description - Transaction Archive Input File (.inx)
USER_ID°	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62). **Note:** The value passed in USER_ID must match the name of the .inx file. For example: <USER_ID>User2</USER_ID> and User2.inx.
COMMAND°	String	Action code for system configuration command. Transaction Archive: ZA
MANUAL_FLAG	String	This field is optional when performing a size limit check, but is required to be set for a full archive. (1 = perform full database archive, 0 = check database size limit).

° These properties are required to submit a transaction archive command.

Transaction Archive Output File (.oux)

Tag	Data Type	Description - Transaction Archive Output File (.oux)
USER_ID	String	Returns the User name that is associated with the transaction. This value is echoed back from the original transaction. The User name will be in DOS file format, max 8 characters.
RESULT	String	Result of the command. **Responses:** **If** MANUAL **set to** 1 **for full archive:** • ARCHIVE SUCCESS = transactions successfully archived to pccwhist.mdb. • ARCHIVE FAILURE = transactions not archived due to: feature not enabled, database under the configured size limit, or an unexpected database error occurred. **If** MANUAL **set to** 0 **for size limit check :** • EXCEEDS LIMIT = Transaction database exceeds the configured archive limit. • UNDER LIMIT = Transaction database is under the configured archive limit.
CURRENT_SIZE	String	Current transaction database size in bytes.
CONFIG_SIZE	String	Current configured size limit for transaction archive in bytes.
INTRN_SEQ_NUM	String	Unique internal sequence number present for all transactions.

Transaction Inquiry Input File (.inx)

Refer to the section **Transaction Inquiry** (see page 128) for more information on the Transaction Inquiry request.

Tag	Data Type	Description - Transaction Inquiry Input File (.inx)
USER_ID°	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section (see page 62). **Note:** The value passed in USER_ID must match the name of the .inx file. For example: <USER_ID>User2</USER_ID> and User2.inx.
COMMAND°	String	Action code for Transaction Inquiry: ZI
TROUTD°	String	Either the TROUTD tag OR the ACCT_NUM tag must be supplied.

Tag	Data Type	Description - Transaction Inquiry Input File (.inx)
ACCT_NUM°	String	Either the TROUTD tag OR the ACCT_NUM tag must be supplied.

° These properties are required to submit a transaction inquiry command.

Transaction Inquiry Output File (.oux)

Tag	Data Type	Description - Transaction Inquiry Output File (.oux)
RECORD_COUNT	String	The number of records matching the inquiry
TRANS_RECORD	String	Contains nested XML tags providing information on transaction(s) pulled from Trans table in the **PC**Charge database (pccw.mdb).

Cashier Configuration File Layout

This section describes the tags required to perform various Cashier Configuration functions.

NOTE: See **Cashier Permissions** on page 75 for an example.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

Add Cashier Configuration Input File (.inx)

Tag	Data Type	Description - Add Cashier Configuration Input File (.inx)
USER_ID *	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section of the manual. **Note:** The value passed in USER_ID must match the name of the .inx file. For example: <USER_ID>User2</USER_ID> and User2.inx.
COMMAND *	String	Action Code for cashier configuration command. **Add Cashier: ZU**
CASHIER_NAME *	String	Submit Cashier's login name.
PASSWORD	String	Submit Cashier's password.
NEW_CASHIER_PASSWORD	String	Submit a new password for the Target Cashier. **Note:** case-sensitive Must be complex: • minimum 7 characters • must have at least one upper case character • one numeric character one special character (e.g., @, $, %, etc.)
NEW_CASHIER_CONFIRM_PASSWORD	String	Resubmit the password for verification. **Note**: Must match the original NewCashierPassword
ADD_CASHIER_1	String	Allows the integrator to submit new cashiers up to 5 at a time. Requires 3 Name Value pairs to be entered for each AddCashier tag submitted. **CASHIER_NAME <Name>** **PASSWORD <Password>** **PERMISSIONS <Permissions>** Example: <ADD_CASHIER_1>CASHIER_NAME CreditR PASSWORD CreditR123% PERMISSIONS 10000|000|000|000|000000000000|0000|000000|00 </ADD_CASHIER_1> See **Chapter 4** Section **Cashier Permissions** For further examples.
ADD_CASHIER_2	String	
ADD_CASHIER_3	String	
ADD_CASHIER_4	String	
ADD_CASHIER_5	String	
TARGET_CASHIER_NAME	String	Submitted when performing a function to edit a cashier's settings. This specifies which cashier's information to alter.

* These properties are required to submit a configuration command.

Add Cashier Configuration Output File (.oux)

Tag	Data Type	Description - Add Cashier Configuration Output File (.oux)
USER_ID	String	Sets the **PC**Charge user name associated with the transaction. The user name must be in DOS file format, no spaces. **Max Length:** 8 characters. For more information on user names, consult the **Multi-User Support** section of the manual. **Note:** The value passed in USER_ID must match the name of the .inx file. For example: <USER_ID>User2</USER_ID> and User2.inx.
RESULT	String	**Responses**: **PROCESSED:** All 5 cashiers have been added. **CAPTURED:** All cashiers with correctly formatted data have been added if less than 5. **NOT CAPTURED:** At least one to four of the cashiers was not added.
AUTH_CODE	String	Success if RESULT = PROCESSED 5 digit code otherwise
INTRN_SEQ_NUM	String	

TCP Interface

Integration via the TCP Interface is very similar to File Method integration. Refer to the section **File Layout Specifications** (see page 483) for information on creating and reading transaction data. When using the TCP Interface, this data is sent and received via sockets.

To process a transaction using the TCP Interface:

1. Create a string that contains the transaction request. Refer to the section **File Layout Specifications** (see page 483) for information on creating the transaction request string.
2. Open a socket connection to **PC**Charge.
3. Send the request string.
4. Once **PC**Charge processes the transaction, a stream of data will be returned with the results
5. Parse this data to retrieve the results of the transaction. Refer to the section **File Layout Specifications** (see page 483) for information on the response data.

The only difference between building the message for the TCP interface and the File Method is that the <XML_FILE> tag can omitted from the request when submitting transactions via the TCP Interface, and the response will not include the <XML_FILE> tags.

Note: Before attempting to send TCP transaction requests to **PC**Charge, the **PC**Charge TCP Interface must be activated. Merchants must check the "Use TCP/IP Connection" option in the Setup | Configure System | Advanced menu of **PC**Charge Payment Server or Pro.

Note: Merchants should make sure the "Enable TCP/IP Client Reversals" option is un-checked in the Setup | Configure System | Advanced menu of **PC**Charge Pro or **PC**Charge Payment Server if using a processor other than FDMS Atlanta. This option is disabled by default.

Note: The default port of 31419 should be changed to maximize security when processing transactions in a live environment. The default port of 31419 may be changed if a device is already using that port on the computer running **PC**Charge.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

PAYware SIM - SIM.DLL (ActiveX) Method

PAYware SIM

PAYware SIM is a comprehensive ActiveX control used to enable electronic payment processing and device control and third party applications. The SIM.dll is designed to allow developers to control the functions of VeriFone pin pads as well as process Credit, Debit, Check, EBT and Settlement transactions to VeriFone's four payment engine services: PAYware PC, PCCharge, PAYware Transact, and IPCharge.

PAYware SIM is similar to the Device.ocx and the PSCharge.dll included in the PCCharge DevKit. The PAYware SIM combines the functionality of both controls, while also giving access to VeriFone's other payment engines.

For more information on the SIM.DLL, please refer to the **PAYware SIM Development Guide** addendum included with the PCCharge DevKit.

CHAPTER 7 -- Code Sample Information

Code Samples

The various sections in this chapter describe several of the code samples that are included in the **PC**Charge DevKit. In many cases, the code samples themselves contain comments, therefore no information will be provided about those code samples in this chapter. Please refer to the comments in the code samples or contact the **PC**Charge Developer Support department for more information.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

Java Client

The File Method Java Client demonstrates the File Method integration using Java.

Note: This program is supplied for demonstration purposes only.

Purpose

This program demonstrates the use of the **PC**Charge File Method interface using a Java client application. It employs classes that encode/decode fields into/from INX and OUX records.

Site/User Specific Code Modifications

The values assigned to the following variables in the `PCCGUI` class code must be customized to what is appropriate to a specific user and site as follows:

```
String Processor = "VISA"; // Processing company

String TID = "999999999911"; // Merchant ID

String User = "User1"; // ignored for unlimited user license, otherwise must be licensed user

String Path = "C:\\Program Files\\active-charge\\"; // path to PCCharge directory (use two "\"s: "\" is a reserved character in Java)
```

Compilation

- Use the following command to compile the code with JavaSoft JDK1.0.2 and JDK1.1.X: `javac Pccfile.java`
- The code also compiles and executes under Microsoft J++.

Compilation produces seven class files, which must be found in the same directory when the main class is executed.

Initiating Execution

As an application, with the Java Virtual Machine:

```
java Pccfile
```

Operation

1. Execution produces a form with six entry fields.
2. Only **Card Number**, **Expiration Date**, and **Amount** fields are mandatory for a valid transaction.
3. Initiate processing by clicking on the **Send** button.
4. A file named `User1.inx` will be written to the directory specified in **Path**.
5. The program will block input until it reads the transaction result from the file `User1.oux`.

A message box will report the result of the transaction.

Web-based Integration Samples

The **PC**Charge DevKit includes several samples designed to demonstrate the coding required to create a Web-based integration to **PC**Charge. The samples included in the DevKit do not actually perform credit card processing. They are purely an interface or "front-end" to the **PC**Charge processing engine.

The intended audience for this chapter includes webmasters and TCP/IP application developers. It assumes basic knowledge of Web, Internet, and TCP/IP terminology, as well as familiarity with the specific web server (if any) to be used.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

System Requirements

PCCharge Virtual Terminal Sample

- Windows NT Server 4.0 or higher with Service Pack 5 installed
- Internet Information Server (IIS) v3.0 or higher, running the Web Server Service Active Server Pages (ASP)
- For Internet-based use, host must provide a Secure Socket Layer (SSL) to ensure secure transactions.
- Accessing and using a **PC**Charge Virtual Terminal site requires either Internet Explorer v6.0 or higher or Netscape v6.0 or higher.
- **PC**Charge with Unlimited Users activated
- Read/write access to the **PC**Charge program directory

ASP Sample

- Windows NT Server v4.0 or higher
- Microsoft Internet Information Server v4.0 or higher
- **PC**Charge with Unlimited Users activated
- Read/write access to the **PC**Charge program directory

Cold Fusion Sample

- Windows NT Server v4.0, or higher
- Cold Fusion v4.0 or higher
- Web Server cable of executing Cold Fusion Tags (Microsoft Internet Information Server, v4.0 or higher recommended)
- **PC**Charge with Unlimited Users activated.
- Read/write access to the **PC**Charge program directory

Java Applet Sample

- Java-enabled Web Server
- Java v1.0.2 or higher or Microsoft J++
- Any operating system supporting Java, including the Java socket functions
- **PC**Charge with Unlimited Users activated
- TCP/IP connectivity to **PC**Charge

PCCharge Virtual Terminal Sample

The **PC**Charge Virtual Terminal is a code sample based on Active Server Page (ASP) technology that demonstrates how merchants can view, charge, and settle credit card transactions using a browser. Typically, applications such as the **PC**Charge Virtual Terminal would be used in the following scenarios:

1. For ISPs that host **PC**Charge and set up eCommerce sites for multiple merchants. The ISP would provide a "Virtual Terminal" interface for their merchants to provide a way to view reports, process voids and post-Authorizations, and settle batches.
2. For Windows-based Intranets. The Virtual terminal can be used as a "Virtual point-of-sale", meaning merchants would simply use their browsers to process transactions across the network, eliminating the need to install extra software on each client machine.

Installation

When installing the DevKit, the **PC**Charge Virtual Terminal will be installed into the directory `C:\Program Files\Active-Charge SDK\Virtual Terminal`.

Choose a folder that will made into a virtual directory in IIS, or create a new sub-folder under an existing website directory. Copy the files and folders from the **PC**Charge Virtual Terminal folder into this virtual directory. If the Virtual Terminal's files and folders are copied to a machine on which the DevKit has not been installed, the file `PSCharge.dll` should also be copied from the directory `\WINNT\System` of the first machine to the directory `\WINNT\System` of the new machine.

`PSCharge.dll` must also be registered for the **PC**Charge Virtual Terminal to work properly (use the command `regsvr32 PSCharge.dll` to register this file).

Note: If problems occur while running **PC**Charge on a separate machine than the Web Server (on the same network), have the Network Administrator set up a Global User as follows:

1. Add a Global User on a trusted domain server.
2. Set the IUSR (found in the IIS Default Web Site) to this Global User.
3. Give the Global User full permissions to the **PC**Charge directory.

In the **Internet Information Services (IIS) Manager** click on the **Web Service Extensions** folder. Here you will see that **Active Server Pages are Prohibited** (this is the default configuration of IIS 6) Highlight **Active Server Pages** and click the **Allow** button ASP is now active.

Once installed, it is important not to change the directory structure of the **PC**Charge Virtual Terminal server files. Several sub-folders are created in the folder, and the files in these folders are referenced in the installed ASP/HTML files. A listing of all ASP/HTML files installed, and a brief description of the purpose of these files, is included in the following ASP/HTML Files Table:

ASP/HTML Files Table

File	Description - ASP/HTML Files Table
`81.asp`	ASP code to display the Credit Card Detail report
`82.asp`	ASP code to display the Batch Pre-Settle report
`83.asp`	ASP code to display the Batch Post-Settle report
`global.asa`	Sets session variables to create connection to charge-processing-engine database and path information on start up.

File	Description - ASP/HTML Files Table
PCCWBatch.asp	Displays options (buttons) for doing batch Inquiry and Settlement
PCCWBatch_post.asp	Displays results of batch Inquiry or Settlement
PCCWBusy.asp	Initializes variables and displays busy message to user during processing of transaction
PCCWDefault.htm	Sets up frames for the **PC**Charge Virtual Terminal
PCCWDisplayRpt.asp	Displays Report
PCCWInquire.asp	Submits requests and receives responses when user clicks **Inquire** to do batch Inquiry
PCCWInquireBusy.asp	Displays busy messages when user is performing batch Inquiry
PCCWMenu.asp	Provides a menu for choosing transactions, settlement, reports, and help options
PCCWMerch.asp	Screen to allow merchant to enter and save merchant number and processing company abbreviation
PCCWMerch_post.asp	Checks validity of entered merchant numbers and processing companies and informs user appropriately. Consult the section **Virtual Terminal Security** (see page 532).
PCCWOfflinePurge.asp	Deletes all offline transactions by deleting the offline transaction file.
PCCWOfflinePurge_post.asp	Displays a "wait" message with a spinning credit card icon during an Offline Batch Purge.
PCCWOfflineRpt.asp	Displays merchant's Offline Transaction Report if merchant has any offline transactions. Offline transactions are transactions that have been submitted to the payment-processing engine, but have not yet been authorized. These transactions are stored until the merchant decides to actually authorize them.
PCCWOfflineRptPurge_post.asp	Displays the results of an Offline Batch Purge.
PCCWOfflineRptUpdate.asp	Updates offline transaction file stored by payment-processing engine when merchant chooses to change status of any transaction in offline file.
PCCWPost.asp	Displays table for allowing user to enter Post Authorization transaction.
PCCWPreReport.asp	Displays different Report types and options from which user can choose.
PCCWReports.asp	Validates user selections on Report types and options screen, and sets errors or calls other reports pages as necessary.
PCCWRunOffline.asp	Submits stored offline transaction file for authorization. Each transaction in file not marked to be dropped is processed (authorized) by payment-processing engine. Those marked to be dropped, if any, are skipped.
PCCWRunOffline_post.asp	Displays results of processing offline transaction file.
PCCWRunOfflineBusy.asp	Displays busy message to user (merchant) during processing of offline transaction file.
PCCWRunTrans.asp	ASP code to process transactions.
PCCWSale.asp	Displays table for allowing user to enter Sale or Pre-Authorization transaction.
PCCWSettle.asp	Submits batch settlement and checks response when user performs batch settlement.
PCCWSettleBusy.asp	Displays busy messages to user when processing batch Settlement.
PCCWTitle.asp	Displays the VeriFone, Inc. logo and the **PC**Charge Virtual Terminal title.
PCCWTrans.asp	Determines what transaction type user has selected from menu and redirects user to correct page for running that transaction.
PCCWTrans_Err.asp	Shows user any error that may have occurred while processing credit card transaction. Displayed in place of PCCWTrans_post.asp if error occurred.
PCCWTrans_post.asp	Shows user results of credit card transaction
PCCWVoid.asp	Displays table allowing user to enter Void transaction type

In addition to the ASP/HTML files installed, there are several sub-directories installed as well:

Virtual Terminal Sub-Directories

Sub-Directory	Description - Virtual Terminal Sub-Directories
help	Contains the online Help HTML files used in **PC**Charge Virtual Terminal
images	Contains all images used in **PC**Charge Virtual Terminal
js	Contains JavaScript helper functions that make users confirm their actions when purging off-line transaction files
css	Contains cascading style sheets used in **PC**Charge Virtual Terminal

Virtual Terminal Installation

The system's "Internet User" *must* be given full access to the directory `C:\Program Files\active-charge` **OR** `C:\Program Files\pccw` for the **PC**Charge Virtual Terminal to work properly. To find out which user is designated as the system's "Internet User":

1. Open the Internet Service Manager
2. Right click on "Default Web Site"
3. Choose Properties
4. Click on Directory Security
5. Click the Edit button next to Anonymous Access and Authentication control
6. Click the Edit button next to Allow Anonymous Access.

The user name that appears next to Username (ex. `IUSR_COMPUTER`) is the "Internet User". Simply add this Username to the **PC**Charge folder's permissions list and assign the user "Full Control" permissions.

Once the server files have been installed, start the Internet Service Manager and make the directory where the files are installed a virtual directory, if it is not already.

Virtual Terminal Security

Caution: Please read this section carefully.

PCCharge Virtual Terminal requires Secure Socket Layer (SSL) encryption to be in place on the site before **PC**Charge Virtual Terminal can be hosted securely for merchants. Once an SSL certificate has been acquired for the site, verify that **PC**Charge Virtual Terminal is setup to use this security. To verify the security setup, follow the procedure below:

1. Open the ASP file named `PCCWMerch_post.asp`.
2. Find the line in this file that reads `Response.Cookies("MrChant").Secure = FALSE`.
3. Change the word `FALSE` to `TRUE`.
4. **Save** this file, and close it.

This ensures that a merchant's merchant account information is encrypted when stored on the machine from which the merchant is browsing.

FOR TESTING PURPOSES ONLY: With test merchant accounts on non-SSL sites, set the above line = FALSE until ready to go live with a **PC**Charge Virtual Terminal site. Setting the line = TRUE without using SSL will cause **PC**Charge Virtual Terminal to not save merchant account information properly.

Setting up Merchant Information

Click **Setup** from the menu frame to display the Merchant Setup Information screen. The information on this screen must be entered correctly before using **PC**Charge Virtual Terminal. Once this information is entered correctly, it will be stored in a cookie. Cookies must be enabled in the local Web browser for the **PC**Charge Virtual Terminal to work correctly.

There are two required fields on this screen: **Processing Company** and **Merchant Number**. These values should be set to a valid Processing Company and Merchant Number that is set up in **PC**Charge. See the **DevKit Constants** section for a list a valid processor codes (see page 143).

Using Virtual Terminal

Use a web browser to open the file: **`PCCWDefault.htm`**

Note: In order to use the Virtual Terminal, the Unlimited User License MUST be activated in PCCharge. See the section **Multi-User Support** (see page 62) for more information.

Processing Transactions

To process a transaction, follow the procedure below:

1. Select the type of transaction to be performed from the drop-down list under **Transactions** in the Menu frame.

2. Click the **Submit** button. This will display one of several transaction screens, depending on the type of transaction selected.

3. Enter data for the available fields and click the **Process** button.

4. **ONLY CLICK THIS BUTTON ONCE** per transaction and **DO NOT** click the **Stop** button on the browser after clicking **Process**. The transaction will be processed even if the **Stop** button is clicked.

A busy animation will appear while processing is underway and the results of the transaction will appear when processing is complete.

PCCharge Virtual Terminal can also process "offline" transactions. When the offline transaction processing is turned on, a `.bch` file is created that holds all transaction information entered into Virtual Terminal.

To process an offline transaction, follow the procedure below:

1. Click **Setup** in the Menu frame.

2. Check the box next to **Perform Offline Transactions**. Click **Update**.

3. Process transactions as shown in the steps above.

Settling Batches

Selecting **Settlement** from the menu frame will display two settlement options: **Inquire** and **Settle**.

If the merchant's processor is Terminal based or set up for manual open/close, batch settlement *must* be performed every day that transactions have been processed to ensure that funds are deposited into the merchant's bank account. The two options on this screen allow merchants to view and settle their daily batches.

Once one of these options have been selected, *do not* click the **Stop** button.

To settle offline transactions, click the **Settlement** button in the Menu frame. Click **Process Offline Transactions**. To remove the transactions saved in the .bch file, click the **Purge Offline Batch** button.

Inquire

Use this to retrieve the current status of the pending batch (unsettled transactions). Clicking this option will return the number of items in the current batch, the current batch number, and the total dollar amount of the transactions in the batch. This option is *only* for viewing the batch status and does not change the status of the batch. It does not settle or close the batch.

Settle

This option settles or closes all of the transactions in the pending batch.

Reports

Because of updated encryption and changes in the database structure, reports are no longer available through the Virtual Terminal starting with PCCharge version 5.7.

ASP Sample

The ASP Sample uses Active Server Pages to communicate transaction and settlement information from a web page to **PC**Charge via the PSCharge.dll. This sample creates an instance of the PSCharge charge or batch class, sets a few properties, calls the send method, and then waits for the response from **PC**Charge. This solution is ideal for Windows-based servers running Internet Information Server.

If the Web Tools were installed during the installation of the DevKit, the ASP sample will be installed in the directory C:\Program Files\active-charge SDK\Web Tools\Asp. Refer to the code and comments in the files in this directory when beginning the integration.

Note: The ASP sample should not be used as-is on a public merchant website because, among other things, it allows the users to specify the transaction amount. The HTML forms of the sample may be modified for this type of website or could be used as the basis of an Intranet-based credit card processing application.

The HTML samples can be installed anywhere in the web server document path. If they are to be used securely, they must be installed in a secure path area as previously configured in the web server software.

PCCharge can run on the same computer as the ASP sample, or they can be separated in various ways. The required connections for the ASP Sample are as follows:

- The ASP pages must be installed into an IIS virtual directory so that they can be properly executed by IIS. Normally this means that they will be installed on the same computer with the web server, although this is not a strict requirement.

- The provided ASP sample communicates transaction requests to **PC**Charge via the PSCharge.dll. Thus, both the ASP pages and **PC**Charge may be running on the same computer or they may reside on different computers. If using separate computers, the servers must be in the same domain (or in domains that trust one another), the **PC**Charge directory must be shared, and proper permissions for the web server's Internet User must be assigned to the **PC**Charge directory. This means that the Internet User that invokes a transaction request on COMPUTER1 (IUSR_COMPUTER1, for example) must have read/write access to the **PC**Charge directory on COMPUTER2 (**Example:** \\COMPUTER2\active-charge). Also, PSCharge.dll must reside and be registered properly on the machine that hosts the ASP pages (COMPUTER1 in this case).

Creating an Object

The user settable properties and methods for each of to DLL's classes are listed in the **DLL (Active X) Method** section of the API (See page 264) or in PSCharge.html. PSCharge.html is located in the DevKit in the C:\Program Files\active-charge SDK\Web Tools\Asp directory. Before any properties or methods can accessed, an instance of a DLL class must be made. This is done from Active Server Pages with code similar to the following:

```
Set GO = Server.CreateObject("PSCharge.Charge")
```

Cold Fusion

The Cold Fusion Sample uses tags to communicate transaction and settlement information from a web page to **PC**Charge via the PSCharge.dll. This sample creates an instance of the PSCharge charge or batch class, sets a few properties, calls the send method, and then waits for the response from **PC**Charge. This solution is ideal for Windows-based servers running a Cold Fusion compatible web server.

If the Web Tools were installed during the installation of the DevKit, the Cold Fusion sample will be installed in the directory C:\Program Files\active-charge SDK\Web Tools\Cold Fusion. Refer to the code and comments in the files in this directory when beginning the integration.

Note: The Cold Fusion sample should not be used as-is on a public merchant website because, among other things, it allows the users to specify the transaction amount. The HTML forms of the sample may be modified for this type of website or could be used as the basis of an Intranet-based credit card processing application.

The HTML samples can be installed anywhere in the web server document path. If they are to be used securely, they must be installed in a secure path area as previously configured in the web server software.

The required connections for the Cold Fusion Sample are as follows:

- The Cold Fusion tags must be installed so that they can be executed as Cold Fusion script by the web server. Normally this means that they will be installed on the same computer with the web server, although this is not a strict requirement.

- The provided Cold Fusion sample communicates transaction requests to **PC**Charge via the PSCharge.dll. Both the Cold Fusion sample and **PC**Charge must be running on the same computer. Also, PSCharge.dll must reside and be registered properly on the machine that hosts the Cold Fusion pages (COMPUTER1 in this case).

Creating an Object

The user settable properties and methods for each of to DLL's classes are listed in the **DLL (Active X) Method** section of the API (See page 264) or in PSCharge.html. PSCharge.html is located in the DevKit in the C:\Program Files\active-charge SDK\Web Tools\Asp directory. Before any properties or methods can accessed, an instance of a DLL class must be made. This is done from Cold Fusion tags with code similar to the following:

```
<CFOBJECT ACTION="Create" NAME="Charge1" CLASS="PSCharge.Charge">
```

Java

Java Applet -- This client-server based sample uses several Java classes to communicate transaction information from a Java Applet to one of our payment-processing engines via sockets. This solution is also ideal for multi-platform sites, or non-Windows-based servers.

Required Connections

PCCharge can be installed and run on the same computer as any of the samples, or they can be separated in various ways. The required connections for each configuration are as follows:

Java Applet

- The Java Applet, the web server, and **PC**Charge must all reside on the same server.
- The Java Applet communicates with **PC**Charge via TCP/IP using sockets. By default, the Java Applet communicates with **PC**Charge via port 31419. The default port of 31419 should be changed to maximize security when processing transactions in a live environment.

TCP Interface Web Samples

The Java Applet sample communicates transaction information to **PC**Charge via sockets. Before attempting to use these samples, you should make sure that the **PC**Charge TCP Interface has been activated and that the proper user configuration has been set in **PC**Charge. Please refer to the **TCP Interface** chapter before proceeding (see page 522).

Java Applet

The folder containing the Java Applet sample contains two main files: `sample.html` and `PCCClient.java`. Before attempting to use the Java Applet, you should edit the `PCCClient.java` file and modify the IP Address, Port, Merchant Number, and Processor code values, if necessary. Once these values match the **PC**Charge configuration, you must then compile `PCCClient.java` using a Java compiler. The command to compile with JavaSoft is `javac PCCClient.java`. Alternatively, `PCCClient.java` may be compiled with Microsoft J++.

Once compiled, seven `.class` files will be created. Below are the classes and their descriptions.

- `HDR.class` -- Format of the header data sent to **PC**Charge
- `INX.class` -- Format of transaction data sent to **PC**Charge
- `OUX.class` -- Format of transaction results returned from **PC**Charge
- `PCCClient.class` -- The main class
- `PCCGUI.class` -- Creates the data entry form and handles command events
- `Records.class` -- Super class for HDR, INP, & OUT records
- `ReportDialog.class` -- Reports transaction results

To run the Java Applet, simply load the `sample.html` form into a Java-capable web browser. The Java Applet will appear.

For more information on the Java Applet, see the `README.txt` file that resides in the directory `C:\Program Files\Web Tools\Java`.

General Troubleshooting

Invalid merchant account?

If the response page indicates an invalid merchant account, you are probably sending the wrong merchant number or processor code to **PC**Charge.

Trouble with multiple transactions?

If your application operates properly with single transactions, but has difficulty with multiple transactions, you may not have the unlimited user version of **PC**Charge installed. Otherwise, there is a high likelihood that your application is not handling the socket connections or data flow properly. The **PC**Charge TCP Interface has been tested successfully with high serial and concurrent transaction volume.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

Permissions

The most common problem integrators experience while attempting to implement an ASP or Cold Fusion integration involves NT permissions. For a web site visitor to successfully submit an ASP or Cold Fusion transaction request via the `PSCharge.dll`, that visitor *must* have full access to the directory `C:\Program Files\active-charge` or `c:\Program Files\PCCW`. To set full access permissions for the visitor, follow these steps:

In Microsoft IIS, go to the Internet Service Manager

Right click on "Default Web Site"

Choose **Properties**

Click on **Directory Security**

Click the **Edit** button next to **Anonymous Access and Authentication control**

Click the **Edit** button next to **Allow Anonymous Access**.

Note the user name that appears next to **Username** (ex. `IUSR_COMPUTER`). This is your "Internet User". Simply add this user to the permissions list of the **PC**Charge directory and assign the user "Full Control" permissions.

If you experience problems using your Web Server and PCCharge on separate machines on the same network, you should have your Network Administrator set up a Global User as follows:

1. Add a Global User on a trusted domain server.
2. Set the IUSR (found in the IIS Default Web Site) to this Global User.
3. Give the Global User full permissions to the **PC**Charge directory.

User In Use

The Integration may receive a response “Not Captured: User In Use”. There are a couple scenarios where this may happen.

- The integration has multiple workstations and all of them send transactions with the same <USER_ID> tag. The User In Use response is letting the integrator/merchant know that the User ID is being used by another system at that time.
- The other time this response is seen is using TCP/IP integration and the connection between the POS and PCCharge is severed as soon as the transaction is sent. The next transaction will return a “Not Captured: User In Use” response. This alerts the integrator that the previous transaction may have been processed. PCCharge does not want to process any further transactions until the integrator/merchant checks the transactions to make sure it was either Captured or Declined. Then, PCCharge must be restarted to clear up the “User In Use” response. Once that is done, transactions will begin to process as usual.

Appendix

Test Credit Cards and Merchant Accounts

The **PC**Charge DevKit includes test merchant accounts and test credit card numbers that can be used while developing and testing integrated applications. Currently, VeriFone, Inc. includes only test merchant accounts for credit card and check processing companies. Debit, EBT, and Gift test merchant accounts can be acquired by contacting the various processing companies directly. Contact information for several processors can be found within this section.

Test Credit Card Numbers

Several credit card numbers are provided for testing purposes. The expiration date for each test card should be 12xx, where 'xx' is the last two digits of the current year or later. Most processors require that the amount of $1.00 be used for test transactions. Some processors will decline amounts greater than $1.00. Integrators should experiment with different amounts during integration and testing. Also included is generic Address Verification Data and Card Verification Values. The processor may return a positive result if these values are used.

Credit Cards

`371449635398431` (American Express)
`4012000033330026` (Visa)
`4387755555555550` (Visa)
`4055011111111111` (Visa Commercial Card)
`5424180279791765` (MasterCard)
`5439750001500206` (MasterCard)
`6011000998980019` (Discover)
`5014861541104200412 Driver ID: 0041` (Fleet One)
`70768599874125027 Driver PIN: 11411` (Fuelman)
`7088869008250004273` (Voyager)

Address Verification Data

Most processors that support AVS will accept the following information as valid:

- Zip Code: `85284`
- Street: `8320 Main Street`

Card Verification Values

Most processors that support CVV2, CVC2, and CID, will accept the following information as valid:

- VISA: `999`
- MasterCard: `998`
- Discover: `996`
- AMEX: `9997`

Test Track Data

This section includes track I and II data for four different credit cards. Integrators that have access to a magnetic stripe encoding device may use this information to create their own test credit cards.

Test Card 1 (Generic MasterCard):

```
%B5424180279791765^VERIFONE TEST 1^08121011000 1111A123456789012?
;5424180279791765=081210110000001234567?
```

Test Card 2 (Generic MasterCard):

```
%B5439750001500206^VERIFONE TEST 2^08121011000 1111A123456789012?
;5439750001500206=081210110000001234567?
```

Test Card 3 (Generic Visa):

```
%B4012000033330026^VERIFONE TEST 3^08121011000 1111A123456789012?
;4012000033330026=081210110000001234567?
```

Test Card 4 (First Data Omaha Test Card):

```
%B0227271714569^FDMS OMAHA TEST CARD^08121005432100000000000000150A?
;0227271714569=081210110000012345678?
```

Test Merchant Account Information

Information about each test merchant account can be found below. The setup files for the test merchant accounts are included in the DevKit in the `\active-charge SDK\Test Merchant Info\` directory. Values such as URLs and phone numbers are different for some processors when using the test accounts. These values have been set up properly in the test merchant account files already. This section should be used as a reference for the test merchant accounts and to acquire contact information for processors that do not provide test accounts in the DevKit.

Note: Some test merchant accounts allow for or even require a real credit card number to be used while testing. If using a real credit card, no charges will be placed against a real card number, but the limit-to-buy will be reduced by the amount of the transaction. Therefore, use small amounts less than $10 when testing.

Note: Some test merchant accounts use Datawire to process transactions via a tcp/ip connection (over the internet). VeriFone is unable to provide the necessary Datawire ID (DID) for a test account. Integrators will be required to setup a test account directly with First Data and request a DID when the test account is created.

!	**WARNING:** New password requirements in **PC**Charge version 5.8 are necessary to meet PA-DSS compliance standards. This also requires that all passwords will **expire** after a maximum of **90 days**.

FDMS North / Cardnet (CES)

Test Credit Card Company Number: Please contact First Data to request an account for testing.

Valid Credit Card Numbers:
`5499840000000329`
`5424180279791765`
`5439750001500206`
`4012000033330026`

To test address verification, use the following:
Zip code: `105232417`
Street Address: `4 NOB HILL DRIVE`

FDMS Omaha / FDR (FDC)

Test Credit Card Company Number: Please contact First Data to request an account for testing.

Valid Credit Card Numbers:
`0227271714569`

FDMS South / NaBanco (NB)

Test Credit Card Company Number: Please contact First Data to request an account for testing.

Valid Credit Card Numbers:
`5442981111111114`
`5424180279791765`

The following phone number may be used for testing: (800) 884-8379

Global Payments-East (NDC)

Test Credit Card Company Number: Please contact Global to request an account for testing.

Valid Credit Card Numbers:
4003010123456780
5424180279791765
5439750001500206
4012000033330026

Chase Paymentech (GSAR)

Test Credit Card Company Number: Please contact Chase Paymentech to request an account for testing.

Valid Credit Card Numbers:
5424180279791765
5439750001500206
4012000033330026

The primary authorization address: https://netconnectvar.paymentech.net/NetConnect/controller

Note: In order to test Canadian Debit with Chase Paymentech (GSAR), the integrator will need to obtain a test merchant account from VeriFone Developer Support, and a VeriFone SC5000 PINpad from Chase Paymentech (GSAR) that is configured properly for use with Canadian Debit. Application version 2.0H must be loaded onto the PINpad. MAC data is specific to the PINpad and merchant number. If EBT transactions will be supported, a separate PINpad device is required. Please contact VeriFone developer support at 1-877-659-8983 or devsupport@verifone.com for more details.

American Express (AMEX)

Integrators may contact the following individual to set up a test account:
Scott Arndt (800) 451-8413

TSYS (Formerly Vital) (VISA)

Test Credit Card Company Number: 999999999911
Test URL: ssl2.vitalps.net
Port: 5003

Valid Credit Card Numbers:
5424180279791765
5439750001500206
4012000033330026

FDMS Nashville / Envoy (FDCN)

Test Credit Card Company Number: Please contact First Data to request an account for testing.

Valid Credit Card Numbers:
5424180279791765
5439750001500206
4012000033330026

National Bankcard Systems (NBS)

Test Credit Card Company Number: Please contact NBS to request an account for testing.

Valid Credit Card Numbers:
5014861541104200412 Driver ID: 0041
70768599874125027 Driver PIN: 11411
6900460430001234566 Driver ID: 410483
7088869008250004273

BAMS FACTS (NPC)

Test Credit Card Company Number: Please contact Bank of America to request an account for testing.

Valid Credit Card Numbers:
5424180279791765
5439750001500206
4012000033330026

ECHO (ECHO)

Test Credit Card Company Number: Please contact Echo to request an account for testing.

Valid Credit Card Numbers:
5439750001500206

Elavon (NOVA)

Test Credit Card Company Number: Please contact Elavon to request an account for testing.

Valid Credit Card Numbers:
5424180279791765
5439750001500206
4012000033330026

RBS WorldPay (LYNK)

RBS WorldPay has not provided VeriFone with contact information to allow integrators to request a test account. Integrators will need to contact RBS WorldPay directly to obtain a test account.

Alliance Data Systems, Inc. (ADSI)

Test Credit Card Company Number: Please contact ADSI to request an account for testing.

Valid Credit Card Numbers:
5424180279791765
5439750001500206
4012000033330026

FDMS Atlanta (BPAS)

FDMS Atlanta has not provided VeriFone with a test account. Integrators will need to contact FDMS Atlanta directly to obtain a test account.

Note: FDMS Atlanta requires a "mini-cert" in order to allow merchants to use a POS / integration with **PC**Charge and their network. For more information, speak with a FDMS Atlanta representative.

Fifth-Third Bank-St Pete (BPS)

Fifth-Third Bank-St Pete has not provided VeriFone with a test account. Integrators will need to contact Fifth-Third Bank-St Pete directly to obtain a test account.

NOTE: Please do not use any of the included test credit card data for testing with Fifth-Third Bank-St Pete. A relationship manager from Fifth-Third Bank-St Pete will provide this information once the test account is created.

Heartland Payment Systems (HPTS)

Heartland Payment Systems has not provided VeriFone with a test account. Integrators will need to contact Heartland directly to obtain a test account.

Check Services Test Information

TeleCheck International, Inc. (TECK)

Test System Phone Number: 1-800-366-8950
Site ID: Please contact TeleCheck International to request a test account.

CrossCheck (CRCK)

Test System Phone Number: 1-800-654-7346
Please contact CrossCheck to request a test account.

Certegy (EFAX)

Test System Phone Number: 1-800-237-2626
SiteID: Please contact Certegy to request a test account.

Integration Troubleshooting

The first step in troubleshooting an integration issue is to rule out the possibility that the issue might be occurring in the **PC**Charge transaction engine. To determine if **PC**Charge is causing the problem, run the transaction directly from the **PC**Charge GUI. If an error occurs or the transaction is unsuccessful when processed directly from the **PC**Charge GUI, the issue is most likely a merchant setup or communication issue. These types of problems must be resolved before any integration troubleshooting can occur. If a merchant is using a licensed copy of **PC**Charge with a live merchant account, they should contact VeriFone, Inc.'s technical support department to resolve the issue. If an integrator is using one of test merchant accounts that are included in the DevKit, the integrator should contact VeriFone, Inc.'s development support department to resolve the issue.

Once it has been determined that **PC**Charge is not returning the error or is cause of the problem, several tools are provided that can be used to troubleshoot the integration issue. In general, most integration issues are related to message formatting and are caused by missing or invalid properties or the invalid formatting of values.

To narrow down which properties or values are causing the problem, a comparison should be made of the transaction request that is being submitted by the integrated application and the transaction request that is submitted by the **PC**Charge GUI.

To make this comparison, first activate IO Logging in **PC**Charge (see below), and then run the transaction from the **PC**Charge GUI. After successfully running the transaction from the **PC**Charge GUI, run the same transaction from the integrated application. The IO Log will capture the details from both transactions, thus allowing the integrator to compare the transaction requests and determine which properties or values are causing the problem.

IODebug.log

The `IODebug.log` is a text file created by the **PC**Charge that can be used to view the transaction requests and responses that are processed by **PC**Charge. Logging to the `IODebug.log` file is disabled by default. To activate IO Logging in **PC**Charge, go to setup, configure system, system log. Place a checkmark under "Create IODebug.Log File" or, from the **PC**Charge GUI, hold down the `Shift` key and then hit the `F11` key.

Once logging has been enabled, the text "**IOLog Enabled**" will appear in the title bar of **PC**Charge. While logging is enabled, every transaction request and response (whether submitted from the **PC**Charge GUI or via integration) will be placed in the file `IODebug.log`. This file will be located in the **PC**Charge directory. The following is an example of the type of information that will appear in the `IODebug.log` file:

```
"==================================="
"PCCharge Pro for Windows v5.7.1"
"-----------------------------------"
"INX : 06/03/04 10:40:21 >> ┐C:\Program Files\PCCW\User1 <XML_FILE>
<XML_REQUEST><USER_ID>User1</USER_ID>...
"OUX : 06/03/04 10:40:22 >> ┐<XML_FILE><XML_REQUEST><USER_ID>User1 </USER_ID>
<RESULT>Error</RESULT>...
```

Note: In this example, text has been truncated and replaced with "…".

The data following "INX" represents the details of request message and the data following "OUX" represents the details of the response message. The actual XML message used to process the transaction starts with <XML_FILE> tag on the INX line.

Note: In order to view the contents of the IODebug.log in a more readable fashion, copy just the XML message (<XML_FILE>[contents]</XML_FILE>) to a new text file. Use the extension of .xml when saving the text file. Open this new file using Internet Explorer. Internet Explorer will automatically show the XML message in a more readable format.

PCCharge "Internal Use" Tags

When viewing the contents of the IODebug.log, integrators will notice that there are several tags that appear in the request message that are not documented in the DevKit manual. These tags are added to each transaction request by **PC**Charge automatically and are reserved for "Internal Use". With the exception of "TROUTD", integrators should ignore these tags when determining the cause of integration issues.

The tags that are reserved for internal use include (but are not limited to) the following:

RESP_TYPE
INTRN_SEQ_NUM
INP_TYPE
ENHANCED_TRANS_FLAG
IMPORT_TRANS_FLAG
TXN_METHOD
IS_PURCHASE_CARD
TROUTD (This property should be passed by integrators when performing follow-on transactions, but is automatically populated for transactions such a Sales, Credits, Pre-Authorizations, etc.)

Transaction Request Duplication

PCCharge has the ability to create a duplicate of each transaction request that is submitted to **PC**Charge. Unlike the IODebug.log file, the contents of the files created by this feature are encrypted. The contents of these files can only be viewed by VeriFone, Inc. support representatives. These files may be requested by VeriFone, Inc. support representatives to assist in troubleshooting.

Transaction Request Duplication is disabled by default. To activate this feature, from the **PC**Charge GUI, hold down the Shift key and then hit the F10 key. Once this feature has been enabled, the text "**TransDup Enabled**" will appear in the title bar of **PC**Charge. While this feature is enabled, every transaction request will be duplicated and placed in an individual file located in the **PC**Charge directory. The filename will include the user name, the date and time of the transaction request, and will have an extension of .dup.

Communication Log

Another tool that can be used for troubleshooting purposes is the **PC**Charge Communication log. This log records the information that is passed from **PC**Charge to the processor. This feature is used to log communication for both modem-based and Internet-based processors.

Communication logging is disabled by default. To activate communication logging, access the modem settings screen from the **Setup** menu in **PC**Charge. (If the "Simple Modem Setup" screen appears, click

the **Use Manual Modem Setup** button.) Click the **Advanced** button and then check the **Create Log File** option. If the modem log will be sent to VeriFone, Inc. for assistance in troubleshooting a problem, the **Encrypt Log File** option must be checked and a 16-digit password must be supplied in order to encrypt the file. If the file will be used by the integrator or merchant to troubleshoot the issue locally, the **Encrypt Log File** option must be unchecked so that the contents of the log file are readable.

Once the communication log has been activated, a file called `Logfile.PCC` will be created in the **PC**Charge directory.

Error Log

PCCharge supports error logging. This feature is always enabled. Any time **PC**Charge encounters an internal error, the date, time, transaction number, and a description of the error will be written into a file named `ErrLog.PCC`. This file appears in the **PC**Charge directory. The information in this file can be used to troubleshoot integration issues. The most common errors that would be logged would be permission errors, database errors, and runtime errors.

Note: Occasionally, information will be written to this file that is intended for informational purposes only. The existence of data in this file does not necessary indicate that an error has occurred while **PC**Charge is processing transactions.

Troubleshooting a Live Installation

If designed properly, integrated applications should always check for the existence of the file `SYS.PCC` in the **PC**Charge directory prior to submitting transactions. This check provides a method that can be used when troubleshooting a live customer site.

When troubleshooting, it is often beneficial to see the exact contents of the `.INX` file that the integrated application is submitting to **PC**Charge. Of course, when **PC**Charge is running and an integrated application submits a transaction, **PC**Charge begins processing the transaction immediately, thus deleting the `.INX` file before it can be retrieved. If **PC**Charge is not running, the integrated application will indicate an error rather than submitting the transaction. To get around this, simply close **PC**Charge and then delete the `SYS.PCC` file prior to submitting the transaction. This procedure will circumvent the `SYS.PCC` check that is built into the integrated application. Once the transaction is submitted by the integrated application, the `.INX` file will be placed into the **PC**Charge directory. This file can now be moved or copied from the **PC**Charge directory. And, using the other tools mentioned in the section, the file can be compared to other `.INX` files or forwarded to VeriFone, Inc. support representatives.

Contacting Support

Although the tools noted in this section are provided to assist integrators in resolving integration issues, it may still be necessary to contact VeriFone, Inc.'s development support department for assistance. Prior to contacting support, it is suggested that all logs are activated while processing the transactions that are causing the issue(s). The logs should be reviewed by integrator and the following may be requested by VeriFone, Inc.'s developer support department:

- `IODebug.log`
- `ErrLog.PCC`
- `Logfile.PCC`
- Any `.INX` or `.OUX` files related to the transaction(s)
- Any `.DUP` files related to the transaction(s)

Contact information for the developer support department can be found in the section **Support Policy** (see page 556).

Distribution and Deployment

VeriFone, Inc. offers a variety of distribution and deployment methods to fit the reseller's needs and resources. They are designed to reduce errors, streamline logistics, reduce shipping, and maximize deployment. VeriFone products require activation for live use. Activation requires an activation code derived from the product's serial number and the individual merchant's account number.

Distribution Methods

Full Distribution

VeriFone's full distribution with packaged CD, user manual, and serial number is the traditional method of deploying our software CD products directly to the merchants for loading, activation, setup, and configuration. Activation information can be pre-loaded prior to shipment or loaded at the time of installation. Setup installation includes:

- Load Software
- Activate Serial Number
- Merchant Account Number
- Activation Code
- Configure Systems
- Password and Preferences
- Queuing and Communications
- Company Information
- Address Verification
- Printers
- Modem
- Advanced Configuration
- User Setup
- Credit Setup
- Debit Setup
- Check Setup
- Other Devices

Fast Distribution

VeriFone provides streamline activation and registration of products shipped through third party deployment to merchants. Through VeriFone fast activation, serial numbers and information is exchanged by e-mail. Registration is completed through an Internet connection to VeriFone's Registration Server. Setup installation includes:

- Load Software
- Activate with Pre-Configured Setup File with Serial Number, Merchant Account Number, and Activation Code
- Configure Systems As Above

Master Disk Distribution

Master Disk distribution deploys products embedded into the POS or eCommerce solutions. This Master Disk copy provides better version control and allows easier distribution with developers' applications. With Master Disk, no physical packaged media is required, only the purchase of licensed serial numbers. Activation only requires the pre-purchased serial number. Master Disk activation is authorized for new installs only, not for upgrades of existing or previously installed licenses. Contact your VeriFone Sales representative for more details on embedding and the Master Disk deployment program.

Embedding

As part of the Master Disk distribution program, participating developers are authorized to embed and automatically install one of VeriFone's payment processing engines (**PC**Charge Payment Server or **PC**Charge Pro) onto each merchant's computer as long as the payment processing engine is configured to run in Demo Mode. To configure either the payment processing engines to run in Demo mode, modify the **PC**Charge shortcut icon to include the `/d` command line switch. For example:

- **PC**Charge Pro: `"C:\Program Files\PCCW\Pccw.exe" /D`
- **PC**Charge Payment Server: `"C:\Program Files\Active-Charge" /D`

Important Note: Prior to activation, the Demo mode command line switch must be deleted to enable LIVE mode.

Demo Mode

PCCharge Payment Server or **PC**Charge Pro can be demonstrated in Demo Mode. In Demo Mode, pseudo-transactions will simulate transactions with sample data without actually communicating to the processor. **Important Note**: Prior to activation, the Demo mode command line switch must be deleted to enable LIVE mode.

Evaluation Mode

VeriFone products can be set up in Evaluation Mode with the test merchant accounts. In Evaluation Mode, **PC**Charge can communicate to various payment processor test sites using the modem or an Internet connection. An actual authorization takes place, but no funds are transferred. Testing under Evaluation Mode is reserved for developers doing interface testing and proof-of-concept testing. For more information on setting up the test merchant accounts to enable Evaluation mode, refer to the section **Install the Test Merchant Accounts** (see page 24).

Warehousing/Block Inventory

Resellers may purchase blocks of inventory to be stocked at VeriFone for later deployment. VeriFone warehousing prevents reseller inventory from becoming obsolete. CDs are not produced until the reseller is ready to install, thereby; the reseller always receives the most current version of our products. With Master Disk distribution, no CDs are produced, only serial numbers that are sent to resellers at the time of the original purchase order. At the time of installation, resellers will supply their customers or submit activation orders to VeriFone for activation. Upon activation, one license will be deducted from the reseller's block of inventory.

Activation

VeriFone products require activation, setup, and configuration to properly function. Without precise setup and configuration, there will be a high probability that the electronic payment-processing software will not process transactions properly. Activation cannot occur without the proper merchant and processor information from the Merchant Services Provider. Therefore, it is very important during the initial setup to have the information available. Accuracy and completeness in setup will yield many dividends in reducing technical support and higher customer satisfaction.

Activation requires three components:

1. **Serial Number**

   Example: `0002 0000 000001 51`

   - First 4 numbers -- Specific to company buying licenses from VeriFone.
   - Second 4 numbers -- reserved for VeriFone.
   - Last 8 numbers -- unique to merchant.
   - Test Serial Number: `1234 1234 123456 54`

2. **Merchant Account Number**

   From the Merchant Services Provider (MSP):

   The merchant account number can only come from the organization providing transaction services. Normally, the organization sending the merchant its monthly statement will provide the merchant account number and the processor being used.

3. **Activation Code**

   From the combination of the Serial Number and Merchant Account Number:

   At the time of installation, the activation code can be obtained from online setup options located on VeriFone's website (www.pccharge.com), through the Reseller Den, or by calling the **PC**Charge Technical Support department (877-659.8981). If you do not have access to the Reseller Den, contact salesinfo@verifone.com to request your login and password for this valuable resource.

Support Policy

Philosophy

VeriFone prides itself on its exceptional customer service, and we recognize that our success in the market is directly tied to the quality of our support. Our staff undergoes continual training not only on our own product offerings, but also on Windows, modems, and other hardware.

Contact

Toll-Free Technical Support Line: (877) 659-8981

Toll-Free Developer Support Line: (877) 659-8983

Technical Support E-mail: `support@verifone.com`

Developer Support E-mail: `DevSupport@verifone.com`

More Information

Support Policy

The most up-to-date information about VeriFone, Inc.'s support resources can be found on the web. The following is a link to the Support Policy:

http://www.pccharge.com/downloads/files/SupportPolicy.pdf

Reseller Web Site

VeriFone has created a Reseller's Den. You may access the reseller's den at www.verifonezone.com . Then, click the **Reseller's Den** link to the right of the middle of the page. To obtain a Login and Password, you'll need to be a certified VeriFone reseller. Contact our Sales Department for more information.