PC Charge DevKit

Technical White Paper

Notice

© VeriFone, Inc.
Chatham Business Center
8001 Chatham Center Drive, Suite 500
Savannah, Georgia 31405
Development Support: (877) 659-8983
Technical Support: (877) 659-8981
Fax: (912) 527-4596

Table of Contents

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.

Introduction

Thank you for your interest in the **PC**Charge DevKit with its associated suite of products.

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.

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.

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, Access, Java, Delphi 7, ASP, ASP.NET and Cold Fusion is included to demonstrate the OCX (ActiveX), DLL (ActiveX), OLE (COM), TCP Interface, and File Methods of integration.

PCCharge DevKit Development Support

The **PC**Charge DevKit includes three hours of telephone developer support to be used within twelve months. After that, a DevKit upgrade may be purchased in order to receive three additional hours of support.

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.

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 http://www.verifone.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.

Check Summary Report – This report summarizes information for check transactions that have been performed.

Credit Card Detail Report – This report summarizes information for credit card transactions that have been performed. 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 PIN-pad. 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.

Integration Methods

There are five **PC**Charge integration methods available. The five integration methods are:

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

Note: Integration methods 1 and 2 (OCX, DLL) can either create a text file that is sent to and processed by **PC**Charge, or the TCP/IP functionality of the OCX or DLL can be used. Thus, all three of these integration methods are essentially sending transaction information to **PC**Charge via the File Method or TCP/IP. 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.

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 an XML 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.

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.

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 an XML 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.

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.

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.

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 XML 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 and **IP**Charge. RiTA Server, designed by VeriFone, Inc., is an enterprise level payment processing product. IPCharge, designed by VeriFone, Inc. is a web based Payment Gateway. If the integrator feels that an integration to the RiTA Server or **IP**Charge products might occur in the future, migration to RiTA or **IP**Charge products 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 and **IP**Charge. RiTA Server, designed by VeriFone, Inc., is an enterprise level payment processing product. IPCharge, designed by VeriFone, Inc. is a web based Payment Gateway. If the integrator feels that an integration to the RiTA Server or **IP**Charge products might occur in the future, migration to RiTA or IPCharge products 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.

PCCharge DevKit Development Support

The **PC**Charge DevKit includes three hours of telephone developer support to be used within twelve months. After that, a DevKit upgrade may be purchased in order to receive three additional hours of support.

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.

Developer Support Contact Information	
Toll-free support hotline	(877) 659-8983
E-mail address	devsupport@verifone.com

Developer Support Hours
9:00 AM to 7:00 PM ET Weekdays During normal working hours, requests for callbacks and e-mail responses are fulfilled within 24 hours. Weekend, holiday and off-shift support can be pre-arranged during normal business hours.

API Documentation Example

The following is an example of the API documentation that is available within the **PC**Charge DevKit manual.

PSCharge.dll Charge Class

The Charge class of PSCharge.dll 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.

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.
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".
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.

Property	Data Type	Description - Charge Class Properties
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. 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.
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 (utility, 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
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. **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. **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.

Property	Data Type	Description - Charge Class Properties
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.
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** 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 not yet available with PCCharge. Leave this set to false.
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** 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
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 00010002000300040005, it stores 5 item IDs (0001, 0002, 0003, 0004, and 0005). These numbers must be obtained from Chase Paymentech.

Property	Data Type	Description - Charge Class Properties
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
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** for more information. **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. 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.

° 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. 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 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** for more information.

Charge Class 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.
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.
GetAmountDue	String	Returns the amount due. Only used for the processor NOVA.
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	Returns the authorization amount for the transaction. Only used for the processor NOVA.
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.
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.

Method Name	Returned Value	Description - PccCharge Methods
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. (GetCreditCardType is the same as GetCardIssuer).
GetCCAvailBalance	String	Returns the PrePaid card balance. Only for pre-paid credit cards with 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.
GetDCAvailBalance	String	Returns the available balance on pre-paid debit cards. Only for pre-paid debit cards with 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.
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.
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** 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), NOVA (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.
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 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 - PccCharge Methods
GetResult	String	Returns the result, which indicates the transaction's status upon completion. Refer to the **Transaction Result Constants** section 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.
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 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 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** 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** 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. 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. 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.
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.
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

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.
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.

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.