forte
a CSG Company

Merchant Web Services

Integration Guide v1.2

Updated September 21, 2015

This document is only for the maintenance of existing integrations. To integrate to the Forte platform using web services, Forte recommends that new merchants use the latest version of Forte's REST API. For more information see DevDocs (http://www.forte.net/devdocs/api_resources/forte_api_v3.htm).

For the latest Information,
visit our website at www.forte.net

Revision History

Version	*Date*	*Changes*
1.0	06/10/2015	Initial
1.1	07/15/2015	Added the Account Updater flag, *SuppressAccountUpdater*, to the Paymethod object per WS-24.
1.2	09/21/2015	Added the AGI Transaction SOAP Endpoint.

Table of Contents

Introduction

Overview

Forte's merchant web services include the following:

- **Client:** Create and manage client and payment method tokens
- **Transactions:** Search and pull transaction information
- **Merchant:** Pull hierarchy data
- **AGI Transaction:** Create payment transactions using the Advanced Gateway Interface (AGI). See the AGI Guide for messaging and response details.

Locations

Use the following test and production locations with the following web services:

Client

- **Sandbox**: https://sandbox.paymentsgateway.net/WS/Client.svc
- **Live**: https://ws.paymentsgateway.net/Service/v1/Client.svc

Transaction

- **Sandbox**: https://sandbox.paymentsgateway.net/WS/Transaction.svc
- **Live**: https://ws.paymentsgateway.net/Service/v1/Transaction.svc

Merchant

- **Sandbox**: https://sandbox.paymentsgateway.net/WS/Merchant.svc
- **Live**: https://ws.paymentsgateway.net/Service/v1/Merchant.svc

AGI Transaction

- **Sandbox**: https://ws.paymentsgateway.net/pgtest/paymentsgateway.asmx
- **Live**: https://ws.paymentsgateway.net/pg/paymentsgateway.asmx

Codes for Field Requirements

The following table lists the requirement codes for the fields described in this document.

Code	*Requirement*	*Description*
M	Mandatory	Must appear when table's fields are used
O	Optional	May appear when table's fields are used
C	Conditional	See description for exact requirements
R	Response Only	Only appears in response messages

Enum Reference

Overview

The following table lists the available enums in the merchant web services.

Available Enums

Name	*Values*
ClientStatus	• Active • Deleted • Suspended
CompareOperators	• Equal • NotEqual • GreaterThan • LessThan ***Note*:** Equal behaves similar to Starts With.
EntryType	• VISA • MAST • DISC • AMER • DINE • JCB • PPD • CCD • POP • RCK • WEB • TEL • CTX • CIE • POS • ARC • BOC • None
FundingType	**eCheck** • Funded • Rejected • Previously_Funded_Reject **Credit Card** • Settled • Chargeback • UnChargeback
PaymentType	• eCheck • CreditCard
SearchOperators	• And • Or
SearchType	• Received • Originated
SortDirection	• ascending • descending

Continued

Enum Reference (cont'd)

Available Enums (cont'd)

Name	*Values*
TransactionColumn	• FirstName • LastName • CompanyName • Enteredby • Last4 • Amount • ConsumerID • Status

Type Reference

Overview

The following tables lists the different types available for merchant web services.

Folder

Name	*Type*	*Description*
Name	string	Folder Name
HID	int	Hierarchy ID
Folders	array	Array of Folder Objects
Merchants	array	Array of Merchant Objects (see the Object Reference section)

OriginationSummary

Name	*Type*	*Description*
MerchantID	string	Merchant ID
DateSum	datetime	Date Summary
TransactionType	int	Transaction Type
Status	string	Status
EntryType	string	Entry Type
Count	int	Count
ConvFeeAmount	double	Convenience Fee Amount (Service Fee)
TotalAmount	double	Total Amount

SettleSummary

Name	*Type*	*Description*
MerchantID	int	Merchant ID
SettleDate	datetime	Settle Date
ResponseCode	string	Response Code
FundingType	string	FundingType
Count	int	Count
ConvFeeAmount	double	Principal Convenience Fee Amount (Service Fee)
TotalAmount	double	Principal Total Amount
ConvFeeAmountFunded	double	Funded Convenience Fee Amount (Service Fee)
TotalAmountFunded	double	Funded Total Amount

Object Reference

Overview

The following tables lists the different objects available for merchant web services.

Authentication

The Authentication object contains the APILoginID and Secure Transaction Key used to identify and authenticate the requestor. Both fields may be maintained in the Virtual Terminal's **Gateway Settings** menu.

Name	*Type*	*Description*
`APILoginID`	string	Set in the Virtual Terminal
`TSHash`	string	`HMACMD5 (APILoginID + "|" + UTCTime, Securetransactionkey)`
`UTCTime`	string	UTC in ticks (since 01/01/0001 00:00:00)

Client

The Client object represents the information for a given client. Most fields are alphanumeric in nature. The only validation rules are the following:

A. The record must have either a First and Last Name or a Company Name
B. The Status must be one of the following enumerated values:
   - `ACTIVE`
   - `SUSPENDED`
   - `DELETED`

Note: Transactions for suspended clients will always fail.

Name	*Type*	*Create*	*Update*	*Delete*	*Get*	*Description*
`MerchantID`	int	M	M	M	M	Merchant ID
`ClientID`	int	R	M	M	M	Client ID
`FirstName`	string	M	M	M	M	Optional if `CompanyName` is present.
`LastName`	string	M	O	—	R	Optional if `CompanyName` is present.
`CompanyName`	string	C	O	—	R	Optional if `FirstName` and `LastName` are present.
`Address1`	string	C	O	—	R	First line of Billing Address for customer
`Address2`	string	O	O	—	R	Second line of Billing Address for customer (e.g., Apt #)
`City`	string	O	O	—	R	Billing City for customer
`State`	string	O	O	—	R	Two-letter abbreviation for the Billing State of the customer
`PostalCode`	string	O	O	—	R	Billing Zip Code for the customer
`PhoneNumber`	string	O	O	—	R	Billing Phone Number for the customer

Continued

Object Reference (cont'd)

Client (cont'd)

Name	Type	Create	Update	Delete	Get	Description
CountryCode	string	O	O	—	R	Billing Country Code for the customer. United States is 01.
EmailAddress	string	O	O	—	R	Billing Email for customer.
FaxNumber	string	O	O	—	R	Billing Fax for customer.
ShiptoFirstName	string	O	O	—	R	First Name of the receiver of the goods.
ShiptoLastName	string	O	O	—	R	Last Name of the receiver of the goods.
ShiptoCompanyName	string	O	O	—	R	Company Name of the receiver of the goods, if applicable.
ShiptoAddress1	string	O	O	—	R	First line of the Shipping Address
ShiptoAddress2	string	O	O	—	R	Second line of the Shipping Address
ShiptoCity	string	O	O	—	R	Shipping City of the receiver of the goods.
ShiptoState	string	O	O	—	R	Two-letter abbreviation of the shipping address State.
ShiptoPostalCode	string	O	O	—	R	The shipping address Zip Code of the receiver of the goods.
ShiptoCountryCode	string	O	O	—	R	Country Code of the shipping address. United States is 01.
ShiptoPhoneNumber	string	O	O	—	R	Phone Number of the receiver of the goods.
ShiptoFaxNumber	string	O	O	—	R	Fax Number of the receiver of the goods.
ConsumerID	string	O	O	—	R	Consumer ID
Status	enum	—	O	—	R	• ACTIVE • SUSPENDED • DELETED

EntryType

Name	Type	Description
CC_Sale_10	string	Credit Card Sale (10)
CC_Auth_Only_11	string	Credit Card Authorization Only (11)
CC_Credit_13	string	Credit Card Credit (13)
CC_Ext_Authorization_15	string	Credit Card External Authorization (15)
eCheck_Sale_20	string	eCheck Sale (20)
eCheck_Auth_Only_21	string	eCheck Authorization Only (21)
eCheck_Credit_23	string	eCheck Credit (23)
eCheck_Force_25	string	eCheck Force (25)

Hierarchy

Name	Type	Description
Folder	Folder	Top Folder Object

Continued

Object Reference (cont'd)

LineItem

Name	Type	Description
Headers	string	Header data
Items	string	Item data

Merchant

Name	Type	Description
Name	string	Merchant Name
ID	int	Merchant ID

Page

Name	Type	Description
PageSize	int	Number of records returned. The maximum value is 0.
PageIndex	int	Page number (0 based)
SortBy	TransactionColumn	Sort by this field.
SortDirection	enum	Direction of the sorted field

PaymentMethod

This object represents payment method data. The ClientID may be omitted if it is being used without a Client record. Only the AcctHolderName, CcExpirationDate, CcProcurementCard, Note, and IsDefault fields may be updated. Other changes represent a new payment method and will have to be put into a new PaymentMethod record. CcCardNumber and EcAccountNumber fields are masked in response messages. Each record may hold either credit card or electronic check information but never both.

Name	Type	Create	Update	Delete	Get	Description
MerchantID	int	M	M	M	M	Merchant ID
ClientID	int	M	M	—	O	If not zero, this field must exist and be associated with a Merchant ID.
PaymentMethodID	int	R	M	M	O	Assigned by the service for Create requests. Updates must exist and be associated with a Merchant ID.
AcctHolderName	string	R	M	M	O	Full name of the credit card account holder as it appears on the credit card.
CcCardNumber	string	M	O	—	R	Must be a valid credit card number.
CcExpirationDate	string	C	—	—	R	Expiration Date of the credit card in YYYYMM format.

Continued

Object Reference (cont'd)

PaymentMethod (cont'd)

Name	*Type*	*Create*	*Update*	*Delete*	*Get*	*Description*
`CcCardType`	enum	C	O	—	R	Supported values include the following: • `VISA` • `MAST` • `AMER` • `DISC` • `DINE` • `JCB`
`CcProcurementCard`	bool	C	—	—	R	A Boolean specifying whether the card is used for procurement.
`EcAccountNumber`	string	C	C	—	R	Must be a valid account number.
`EcAccountTRN`	string	C	—	—	R	Must be a valid routing number.
`EcAccountType`	enum	C	—	—	R	Supported values include the following: • `Savings` • `Checking`
`Note`	string	O	O	—	R	Merchant data
`IsDefault`	bool	C	C	—	R	Automatically sets to true if the merchant has only specified one payment method.
`SuppressAccountUpdater`	bool	C	—	—	—	A Boolean flag specifying whether Forte should run Account Updater* services.

* Forte's Account Updater service automatically updates cardholder account information by exchanging electronic credit card account updates with existing account databases that are maintained by the card associations. For more information on this service, contact Customer Service.

Range

Name	*Type*	*Description*
`MerchantIDs`	string	Merchant IDs
`StartDate`	datetime	Start Date of the range
`EndDate`	datetime	End Date of the range
`ChildMID`	Boolean	If set to true, the child merchant IDs will be included.
`EntryType`	EntryType	
`TransactionType`	TransactionType	This parameter has not been implemented

ReceivedSummary

Name	*Type*	*Description*
`MerchantID`	int	Merchant ID
`DateSum`	datetime	Date Summary
`TransactionType`	int	Transaction Type
`Status`	string	Status
`Code`	string	Code
`EntryType`	string	Entry Type
`Count`	int	Count
`ConvFeeAmount` (Legacy)	double	Convenience Fee Amount
`ServiceFeeAmount`	double	Service Fee Amount
`TotalAmount`	double	Total Amount

Continued

Object Reference (cont'd)

SearchFilter

Name	*Type*	*Description*
FieldName	enum	Field to Search
CompareOperator	enum	Method of Comparison
SearchOperator	enum	Method of Search
SearchText	enum	String to Search

Settlement

The settlement object holds information about transferring the funds from the card issuer's bank to the acquiring bank, which is your store's bank.

Name	*Type*	*Description*
MerchantID	string	Merchant ID
TransactionID	guid	Transaction ID
TransSettleID	guid	Transaction Settle ID
SettleDate	datetime	Settle Date
ResponseCode	string	Response Code
PaymentType	string	Credit Card or eCheck
FundingType	string	Funding Type
TotalAmountFunded	double	Total Amount Funded
ServiceFeeAmountFunded	double	Service Fee Amount Funded
Transaction	object	Transaction Object

Transaction

The transaction object holds the credit card/echeck, billing, and shipping information for an order.

Name	*Type*	*Description*
MerchantID	string	Merchant ID
Amount (*TotalAmount)	double	Amount
SalesTaxAmount	double	Sales Tax Amount
ConvenienceFee (*ServiceFeeAmount)	double	Convenience Fee Amount
ConvenienceFeePrincipal (*ServiceFeePrincipalAmount)	double	Convenience Fee Principal
DebitCredit	string	Debit or Credit Transaction
EnteredBy	string	Entered By
IPAddress	string	IP Address
TransactionID	string	Transaction or Trace ID
UpdatetoDate	datetime	Updated Date of Transaction
BillTo Fields		
BilltoFirstName	string	Bill to First Name
BilltoLastName	string	Bill to Last Name
BilltoCompanyName	string	Bill to Company Name
BilltoAddress	string	Bill to Address
BilltoAddress2	string	Bill to Address 2
BilltoCity	string	Bill to City
BilltoState	string	Bill to State
BilltoPostalCode	string	Bill to Postal Code
BilltoEmailAddress	string	Bill to Email Address
BilltoPhone	string	Bill to Phone
ShipTo Fields		
ShiptoName	string	Ship to Name
ShiptoCompanyName	string	Ship to Company Name
ShipttoAddress	string	Ship to Address

Continued

Object Reference (cont'd)

Transaction (cont'd)

Name	*Type*	*Description*
ShipTo Fields		
ShipttoAddress2	string	Ship to Address 2
ShiptoCity	string	Ship to City
ShiptoState	string	Ship to State
ShiptoPostalCode	string	Ship to Postal Code
ShiptoFreightAmount	string	Ship to Freight Amount
Defined Fields		
ConsumerOrderID	string	Consumer Order ID
ConsumerID	string	Consumer ID
WalletID	string	Wallet ID
MerchantData1	string	Merchant Data 1
MerchantData2	string	Merchant Data 2
MerchantData3	string	Merchant Data 3
MerchantData4	string	Merchant Data 4
MerchantData5	string	Merchant Data 5
MerchantData6	string	Merchant Data 6
MerchantData7	string	Merchant Data 7
MerchantData8	string	Merchant Data 8
MerchantData9	string	Merchant Data 9
LineItems	array	Array of the LineItems objects
User Defined	string	User-defined fields
eCheck Fields		
OriginationDate	datetime	Origination Date
UpdateDate	datetime	Updated Date
AttemptNumber	string	Attempt Number
AcctTRN	string	Account Transit Routing Number
AccountType	string	Account Type
CheckNo	string	Check Number
EntryClassCode	string	Entry Class Code
EntryDescription	string	Entry Description
ItemDescription	string	Item Description
Credit Card Fields		
CardType	string	Card Type
CardExpDate	string	Card Expiration Date
CardHolderName	string	Cardholder Name
TransactionResponse		

TransactionResponse

After the transaction has been submitted, this object holds the transaction response.

Name	*Type*	*Description*
TransactionID	string	Transaction or Trace ID
Last4	string	Last four digits of the credit card or echeck account number
ResponseType	string	Response Type
ResponseCode	string	Response Code
ResponseDescription	string	Response Description
Status	string	Status
AuthCode	string	Authorization Code
PreauthResult	string	Pre-Authorization Result
PreauthCode	string	Pre-Authorization Code
PreauthDescription	string	Pre-Authorization Description
PreauthNegReport	string	Pre-Authorization Negative Items
AVSResult	string	Address Verification Service Result
MerchantClientID	int	Merchant Client ID, if created

Continued

Object Reference (cont'd)

TransactionSummary

The TransactionSummary object summarizes the transaction object, highlighting some of the more important fields.

Name	*Type*	*Description*
MerchantID	string	Merchant ID
MerchantClientID	string	Merchant Client ID
TransactionID	guid	Transaction or Trace ID
Status	string	Status
TsCreated	datetime	Date of Transaction
Last4	string	Last four digits of the echeck account or credit card number
BilltoFirstName	string	Bill to First Name
BilltoLastName	string	Bill to Last Name
BilltoCompanyName	string	Bill to Company Name
CardType	string	Credit Card Type
DebitCredit	string	Debit or Credit Transaction
EnteredBy	string	Entered By
ConsumerOrderID	string	Consumer Order ID
ConsumerID	string	Consumer ID
WalletID	string	Wallet ID
ResponseCode	string	Response Code
AuthCode	string	Authorization Code
Amount	double	Amount of the transaction
ConvenienceFeePrincipal	double	Amount of the Convenience Fee Principal, if applicable
TransactionType	int	Numeric Transaction Code

Methods Reference

getTransactionAuthTicket

This method allows the user to authenticate to the Transaction Service so the user can work with the Transaction API.

Name	*Type*	*Description*
Request		
strAPILoginID	string	API Login ID for Authentication
strKey	string	Secret Key for Authentication
Return	**Authentication**	**An authentication object.**

Sample Code

```
public static TransactionService.Authentication GetTransactionAuthTicket(string
strAPILoginID, string strKey)
        {
            TransactionService.Authentication ticket = new
TransactionService.Authentication();
            ticket.APILoginID = strAPILoginID;
            ticket.UTCTime = DateTime.UtcNow.Ticks.ToString();
            ticket.TSHash = CalculateHMACMD5(ticket.APILoginID + "|" +
ticket.UTCTime, strKey.Trim());

            return ticket;
        }
```

getClientAuthTicket

This method allows the user to authenticate to the Transaction Service so the user can work with the Transaction API.

Name	*Type*	*Description*
Request		
strAPILoginID	string	API Login ID for Authentication
strKey	string	Secret Key for Authentication
Return	**Authentication**	**An authentication object.**

Sample Code

```
public static ClientService.Authentication GetClientAuthTicket(string strAPILoginID,
string strKey)
{
    ClientService.Authentication ticket = new ClientService.Authentication();
    ticket.APILoginID = strAPILoginID;
    ticket.UTCTime = DateTime.UtcNow.Ticks.ToString();
    ticket.TSHash = CalculateHMACMD5(ticket.APILoginID + "|" + ticket.UTCTime,
strKey.Trim());

    return ticket;
}
```

Continued

Methods Reference (cont'd)

createClient

This method creates a client and sets the client's status to active (`A`).

Name	*Type*	*Description*
Request		
`Ticket`	Authentication	Authentication Object
`Client`	object	Client Object
`Return`	**int**	**Client ID**

Sample Code

```
private void CreateClient(int mid)
{
    ClientRecord client = new ClientRecord();
    client.MerchantID = MerchantID;
    client.FirstName = "Bob";
    client.LastName = "Smith";
    //other code describing client omitted

    try
    {
        using (ClientServiceClient proxy = new ClientServiceClient())
        {
            int id =
proxy.createClient(Authenticate.GetClientAuthTicket(txtID.Text.Trim(),
txtKey.Text.Trim()), client);
            Response.Write("Created Client ID = " + id.ToString());
        }
    }
    catch (Exception e)
    {
      Response.Write(e.Message);
    }
}
```

updateClient

This method updates a client. ***Note:*** Deleting a client or updating the client's status to `Suspended` will likewise affect the client's scheduled transactions. Updating the client status from `Suspended` to `Active` will not affect the client's scheduled transactions.

Name	*Type*	*Description*
Request		
`Ticket`	Authentication	Authentication object
`Client`	object	Client object
`Return`	**int**	**Client ID**

Continued

Methods Reference (cont'd)

updateClient (cont'd)

Sample Code

```
private void UpdateClient(int mid)
{
    ClientRecord client = new ClientRecord();
    client.MerchantID = MerchantID;
    client.FirstName = "Bob";
    client.LastName = "Smith";


    try
    {
        using (ClientServiceClient proxy = new ClientServiceClient())
        {
            int id =
proxy.updateClient(Authenticate.GetClientAuthTicket(txtID.Text.Trim(),
txtKey.Text.Trim()), client);
            Response.Write("Updated Client ID = " + id.ToString());
        }
    }
    catch (Exception e)
    {
      Response.Write(e.Message);
    }
}
```

deleteClient

This method deletes a client.

Name	*Type*	*Description*
Request		
Ticket	Authentication	Authentication Object
MerchantID	int	Merchant ID
ClientID	int	Client ID
Return	**int**	**Client ID**

Sample Code

```
private void DeleteClient(int ClientID, int mid)
        {
            try
            {
                using (ClientServiceClient proxy = new ClientServiceClient())
                {
                    int id =
proxy.deleteClient(Authenticate.GetClientAuthTicket(txtID.Text.Trim(),
txtKey.Text.Trim()), mid, ClientID);
                    Response.Write("Deleted Client ID: " + id.ToString());
                }
            }
            catch (Exception e)
            {
               Response.Write(e.Message);
            }
        }
```

Continued

Methods Reference (cont'd)

getClient

This method gets a client.

Name	*Type*	*Description*
Request		
Ticket	Authentication	Authentication Object
MerchantID	int	Merchant ID
ClientID	int	Client ID
Return	**array**	**Client**

Sample Code

```
public void GetClients(int mid)
{
    try
    {
        using (ClientServiceClient proxy = new ClientServiceClient())
        {
            int ClientID = txtClientID.Text.Trim().Length > 0 ?
Convert.ToInt32(txtClientID.Text) : 0;
            grid.DataSource =
proxy.getClient(Authenticate.GetClientAuthTicket(txtID.Text.Trim(),
txtKey.Text.Trim()), mid, ClientID);
            grid.DataBind();
        }
    }
    catch (Exception e)
    {
        Response.Write(e.Message);
    }
}
```

createPaymentMethod

This method creates a new `PaymentMethod` record. Either the credit card or echeck fields must be present, but not both. If the method is created for a valid `ClientID` and no other payment methods exists for that client, then the `isDefault` flag will be set automatically. If successful, the new `PaymentMethodID` is returned and a fault is returned on failure.

Name	*Type*	*Description*
Request		
Ticket	Authentication	Authentication object
PaymentMethod	object	Payment Method object
Return	**int**	**Payment Method ID**

Continued

Methods Reference (cont'd)

createPaymentMethod (cont'd)

Sample Code

```
private void CreateCCPayment(int ClientID, int mid)
{
    PaymentMethod payment = new PaymentMethod();
    payment.AcctHolderName = "Bob Smith";
    payment.CcCardNumber = "4111111111111111";
    payment.CcExpirationDate = "201506";
    payment.CcCardType = CcCardType.VISA;
    payment.Note = "Insert Note";
    payment.ClientID = ClientID;
    payment.MerchantID = mid;

    try
    {
        using (ClientServiceClient proxy = new ClientServiceClient())
        {
            int id =
proxy.createPaymentMethod(Authenticate.GetClientAuthTicket(txtID.Text.Trim(),
txtKey.Text.Trim()), payment);
            Response.Write("Created CC Payment Method ID: " + id.ToString());
        }
    }
    catch (Exception e)
    {
        Response.Write(e.Message);
    }
}
```

updatePaymentMethod

This method updates an existing `PaymentMethod` object. Setting the `isDefault` flag for a payment method will unset the flag for the previous default (if one existed). A fault is returned upon failure. ***Note:*** The `ClientID` field may not be updated.

Name	*Type*	*Description*
Request		
`Ticket`	Authentication	Authentication object
`PaymentMethod`	object	Payment Method object
`Return`	**int**	**Payment Method ID**

Continued

Methods Reference (cont'd)

updatePaymentMethod (cont'd)

Sample Code

```
private void UpdateCCPayment(int ClientID, int mid)
{
    PaymentMethod payment = new PaymentMethod();
    payment.AcctHolderName = "Bob Sanders";
    payment.CcExpirationDate = "201111";
    payment.IsDefault = true;
    payment.CcProcurementCard = true;
    payment.Note = "Insert Note";
    payment.ClientID = ClientID;
    payment.MerchantID = mid;
    payment.PaymentMethodID = Convert.ToInt32(ViewState["PaymentMethodID"]);

    try
    {
        using (ClientServiceClient proxy = new ClientServiceClient())
        {
            int id =
proxy.updatePaymentMethod(Authenticate.GetClientAuthTicket(txtID.Text.Trim(),
txtKey.Text.Trim()), payment);
            Response.Write("Updated CC Payment Method ID: " + id.ToString());
        }
    }
    catch (Exception e)
    {
        Response.Write(e.Message);
    }
}
```

deletePaymentMethod

This method deletes a payment method. ***Note:*** Payment Methods used by Active or Suspended scheduled transactions cannot be deleted.

Name	*Type*	*Description*
Request		
Ticket	Authentication	Authentication object
MerchantID	int	Payment Method object
PaymentMethodID	int	Payment Method ID
Return	**int**	**Payment Method ID**

Continued

Methods Reference (cont'd)

deletePaymentMethod (cont'd)

Sample Code

```
private void DeletePayment(int PaymentMethodID, int mid)
{
    try
    {
        using (ClientServiceClient proxy = new ClientServiceClient())
        {
            int id =
proxy.deletePaymentMethod(Authenticate.GetClientAuthTicket(txtID.Text.Trim(),
txtKey.Text.Trim()), mid, PaymentMethodID);
            Response.Write("Deleted Payment Method ID: " + id.ToString());
        }
    }
    catch (Exception e)
    {
        Response.Write(e.Message);
    }
}
```

getPaymentMethod

This method gets a payment method.

Name	*Type*	*Description*
Request		
Ticket	Authentication	Authentication object
MerchantID	int	Payment Method object
ClientID	int	Client ID
PaymentMethodID	int	Payment Method ID
Return	**Array**	**Payment Method array**

Sample Code

```
public void GetPaymentMethods(int mid)
{
    try
    {
        using (ClientServiceClient proxy = new ClientServiceClient())
        {
            grid.DataSource =
proxy.getPaymentMethod(Authenticate.GetClientAuthTicket(txtID.Text.Trim(),
txtKey.Text.Trim()), mid, ClientID, PaymentMethodID);
            grid.DataBind();
        }
    }
    catch (Exception e)
    {
        Response.Write(e.Message);
    }
}
```

Continued

Methods Reference (cont'd)

getHierarchyTree

This method gets a hierarchy tree.

Name	*Type*	*Description*
Request		
Ticket	Authentication	Authentication object
MerchantID	int	Merchant ID
Return	**Hierarchy**	**Hierarchy object**

searchTransactions

This method searches received or originated transactions.

Name	*Type*	*Description*
Request		
Ticket	Authentication	Authentication object
Range	Range	Range object
Page	Page	Paging object
SearchType	SearchType	Received or originated results
PaymentType	PaymentType	Credit Card or eCheck
sFilter	SearchFileter	Array of transaction search filter objects
TotalCount	(out) int	Total record count
Return	**Array**	**Array of TransactionSummary objects**

Sample Code

```
 protected void btnGetTransDetail_Click(object sender, EventArgs e)
        {
            try
            {
                using (TransactionServiceClient proxy = new
TransactionServiceClient())
                {
                    int TotalRecords;
                    Range range = new Range();
                    range.StartDate = new DateTime(2000, 1, 1);
                    range.EndDate = new DateTime(2010, 1, 12);
                    TransactionService.Page page = new
ServiceTestClient.TransactionService.Page();
                    page.PageIndex = 0;
                    page.PageSize = 0;
                    page.SortBy = TransactionColumn.Amount;
                    page.SortDirection =
ServiceTestClient.TransactionService.SortDirection.ascending;
                    Authentication ticket =
Authenticate.GetTransactionAuthTicket(txtID.Text.Trim(), txtKey.Text.Trim());

                    TransactionSummary[] tran = proxy.searchTransactions(ticket,
range, page, SearchType.Originated, null, null, out TotalRecords);
                    foreach (TransactionSummary t in tran)
                    {
                        Response.Write(t.Amount);
                    }
}
}
```

Continued

Methods Reference (cont'd)

searchSettleActivity

The `searchSettleActivity` method offers a settlement history of a transaction including chargebacks (`C00`) and un-chargebacks (`U00`). The transaction object is not available within Settle objects for this method. Use the `getSettleDetails` or `getTransaction` methods to get the transaction information.

Name	*Type*	*Description*
Request		
`Ticket`	Authentication	Authentication object
`Range`	Range	Range object
`PaymentType`	PaymentType	Payment Type object
`FundingType`	FundingType	Funding Type object
`PageIndex`	int	Zero-based page index
`PageSize`	int	Number of records per page (max 200)
`Return`	**array**	**Array of Settle objects**
	int	**Number of total records returned if not paged**

Continued

Methods Reference (cont'd)

searchSettleActivity (cont'd)

Sample Code

```
private void DoSearch(int pageIndex, int pageSize)
{
    try
    {
        using (TransactionServiceClient proxy = new TransactionServiceClient())
        {
            int TotalRecords = 0;
            Range range = new Range();
            range.StartDate = Convert.ToDateTime(txtStart.Text);
            range.EndDate = Convert.ToDateTime(txtEnd.Text);

            range.ChildMID = chkChild.Checked;
            range.MerchantIDs = txtMID.Text;

            Nullable<PaymentType> p = null;

            switch (ddlPayment.SelectedValue)
            {
                case "CreditCard":
                    p = PaymentType.CreditCard;
                    break;
                case "eCheck":
                    p = PaymentType.eCheck;
                    break;
            }

            Nullable<FundingType> f = null;

            switch (ddlFunding.SelectedValue)
            {
                case "Chargeback":
                    f = FundingType.Chargeback;
                    break;
                case "Funded":
                    f = FundingType.Funded;
                    break;
                case "Previously Funded Reject":
                    f = FundingType.Previously_Funded_Reject;
                    break;
                case "Rejected":
                    f = FundingType.Rejected;
                    break;
                case "Settled":
                    f = FundingType.Settled;
                    break;
                case "UnChargeback":
                    f = FundingType.UnChargeback;
                    break;

            }

            grid.DataSource =
proxy.searchSettleActivity(Authenticate.GetTransactionAuthTicket(txtID.Text.Trim(),
txtKey.Text.Trim()), range, p, f, pageIndex, pageSize, out TotalRecords);

            grid.DataBind();
           }
      }
}
```

Continued

Methods Reference (cont'd)

getReceivedSummary

This method gets received transactions by date.

Name	*Type*	*Description*
Request		
Ticket	Authentication	Authentication object
Range	Range	Range object
PageIndex	int	Zero-based page index
PageSize	int	Number of records per page (max 200)
Return	**ReceivedSummary**	**Array of received summary objects**
	int	**Number of total records returned if not paged**

Sample Code

```
        private object CallSummaries(int pageIndex, int pageSize, ref int TotalRecords)
        {
            try
            {
                using (TransactionServiceClient proxy = new
TransactionServiceClient())
                {
                    Range range = new Range();
                    range.StartDate = Convert.ToDateTime(txtStart.Text);
                    range.EndDate = Convert.ToDateTime(txtEnd.Text);

                    range.ChildMID = chkChild.Checked;
                    range.MerchantIDs = txtMID.Text;

                    object summaries = null;

                    Authentication ticket =
Authenticate.GetTransactionAuthTicket(txtID.Text.Trim(), txtKey.Text.Trim());

                    if ((string)ViewState["type"] == "getSettleSummary")
                    {
                        range.EntryType = (EntryType)Enum.Parse(typeof(EntryType),
ddlEntryType.SelectedValue);
                        range.TransactionType =
(TransactionType)Enum.Parse(typeof(TransactionType), ddlTransType.SelectedValue);
                        summaries = proxy.getSettleSummary(ticket, range,
pageIndex, pageSize, out TotalRecords);
                    }
                    else if ((string)ViewState["type"] == "getReceivedSummary")
                        summaries = proxy.getReceivedSummary(ticket, range,
pageIndex, pageSize, out TotalRecords);
                    else if ((string)ViewState["type"] == "getOriginationSummary")
                        summaries = proxy.getOriginationSummary(ticket, range,
pageIndex, pageSize, out TotalRecords);

//Other code omitted
                }

            }
            catch (Exception ex)
            {
                Response.Write(e.Message);
        }
          }
```

Continued

Methods Reference (cont'd)

getOriginationSummary

This method gets received transactions by date.

Name	*Type*	*Description*
Request		
Ticket	Authentication	Authentication object
Range	Range	Range object
PageIndex	int	Zero-based page index
PageSize	int	Number of records per page (max 200)
Return	**OriginationSummary**	**Array of origination summary objects**
	int	**Number of total records returned if not paged**

getSettleSummary

This method gets settled transactions by date.

Name	*Type*	*Description*
Request		
Ticket	Authentication	Authentication object
Range	Range	Range object
PageIndex	int	Zero-based page index
PageSize	int	Number of records per page (max 200)
Return	**SettletSummary**	**Array of settle summary objects**
	int	**Number of total records returned if not paged**

getSettleDetail

This method gets settled transaction by settle date.

Name	*Type*	*Description*
Request		
Ticket	Authentication	Authentication object
MerchantID	int	Merchant ID
Day	datetime	Day
PageIndex	int	Zero-based page index
PageSize	int	Number of records per page (max 200)
Return	**OriginationSummary**	**Array of Settle objects**
	int	**Number of total records returned if not paged**

Continued

Methods Reference (cont'd)

getSettleDetail (cont'd)

Sample Code

```
private void DoSearch(int pageIndex, int pageSize)
 {
     try
     {
         using (TransactionServiceClient proxy = new TransactionServiceClient())
         {
             int TotalRecords = 0;
             int MerchantID = Convert.ToInt32(txtMID.Text);
             DateTime day = Convert.ToDateTime(txtDay.Text);

             grid.PageSize = Convert.ToInt32(txtPageSize.Text);

             Settle[] settles =
proxy.getSettleDetail(Authenticate.GetTransactionAuthTicket(txtID.Text.Trim(),
txtKey.Text.Trim()), MerchantID, day, pageIndex, pageSize, out TotalRecords);

             grid.DataSource = settles;
             grid.DataBind();

            //Other code omitted
          }

     }
     catch (Exception ex)
     {
         Response.Write(e.Message);
     }
 }
```

getReceivedDetail

This method gets received transactions by the received date.

Name	*Type*	*Description*
Request		
Ticket	Authentication	Authentication object
MerchantID	int	Merchant ID
Day	datetime	Day
PageIndex	int	Zero-based page index
PageSize	int	Number of records per page (max 200)
Return	**Transaction**	**Transaction detail object (no line items)**
	int	**Number of total records returned if not paged**

Continued

Methods Reference (cont'd)

getReceivedDetail (cont'd)

Sample Code

```
private void DoSearch(int pageIndex, int pageSize)
{
    try
    {
        using (TransactionServiceClient proxy = new TransactionServiceClient())
        {
            int TotalRecords = 0;
            DateTime day = Convert.ToDateTime(txtDay.Text);

            int MerchantID = Convert.ToInt32(txtMID.Text);

            grid.DataSource =
proxy.getReceivedDetail(Authenticate.GetTransactionAuthTicket(txtID.Text.Trim(),
txtKey.Text.Trim()), MerchantID, day, pageIndex, pageSize, out TotalRecords);
            grid.DataBind();


            lblMsg.Text = string.Empty;
            lblInfo.Text = "Total records count: " + TotalRecords.ToString();
            if (TotalRecords > 0)
                Response.Write("<br>Showing record #: " + (pageIndex * pageSize +
1).ToString() + " to " + ((TotalRecords < (pageIndex + 1) * pageSize) ?
TotalRecords.ToString() : ((pageIndex + 1) * pageSize).ToString()));

            //show pager if TotalRecords > PageSize
            if (TotalRecords > Convert.ToInt32(txtPageSize.Text))
            {
                int totalpages = TotalRecords / Convert.ToInt32(txtPageSize.Text) +
1;
                divPager.Visible = true;
                lnkPrev.Enabled = (pageIndex > 0);
                lnkNext.Enabled = (pageIndex < totalpages - 1);
                //load avaliable page number
                lstPage.Items.Clear();
                for (int i = 0; i < totalpages; i++)
                {
                    lstPage.Items.Add(i.ToString());
                }
                lstPage.SelectedValue = pageIndex.ToString();

            }
            else
                divPager.Visible = false;
        }
    }
    catch (Exception ex)
    {
        Response.Write(e.Message);
      }
}
```

Continued

Methods Reference (cont'd)

getTransaction

This method gets details of a transaction.

Name	*Type*	*Description*
Request		
Ticket	Authentication	Authentication object
MerchantID	Range	Merchant ID
TransactionID	int	Transaction ID
Return	**TransactionSummary**	**Array of transaction summary objects**

Sample Code

```
protected void btnGetTransDetail_Click(object sender, EventArgs e)
{
    try
    {
        using (TransactionServiceClient proxy = new TransactionServiceClient())
        {
            Transaction tran =
proxy.getTransaction(Authenticate.GetTransactionAuthTicket(txtID.Text.Trim(),
txtKey.Text.Trim()), Convert.ToInt32(txtMID.Text), txtTransID.Text);

            //show the transaction in a grid
            grid.DataSource = new List<Transaction> { tran };
            grid.DataBind();

            //show the transaction response in a grid
            gridResponse.DataSource = new List<TransactionResponse> { tran.Response
};
            gridResponse.DataBind();

            //show the lineitems in a grid
            gridLineItem.DataSource = tran.LineItems;
            gridLineItem.DataBind();
            lblMsg.Text = string.Empty;
        }
    }
    catch (Exception ex)
    {
Response.Write(e.Message);
    }
}
```

Error Messages

Error Messages

Name	*Description*
`U55`	See `pg_response_description`
`Invalid Client Token`	Either the specified token was not found or the token does not belong to the merchant making the request.
`Client Token Suspended`	The client's Status has been set to `Suspended` and no transactions may be processed for it at this time.
`No Default Payment Token`	The client token specified has no default payment method token.
`Invalid Payment Token`	Either the specified payment token was not found, the payment token does not belong to the merchant making the request, or the payment token associated with the client token is also present in the request.