vindicia
Build Subscription Revenue

# CashBox®
# API Reference Guide

CashBox 5.0
February, 2014

March 1, 2014

# Table of Contents

 

 

 

 

# CashBox® API Guide Preface

CashBox is an on-demand solution for recurring and one-time billing, available for integration with your application through an object-oriented application programming interface (API), based on the Simple Object Application Protocol (SOAP). The CashBox solution is accessed through a public API to the CashBox application, which is hosted and maintained on the Vindicia network.

The CashBox API leverages a Service Oriented Architecture (SOA), meaning that CashBox users are not required to install application software on their network. Instead, use SOAP to communicate with the CashBox application, either through a thin client provided by Vindicia, or through a WSDL published by the Vindicia SOAP servers (e.g. http://soap.vindicia.com/1.0/Transaction.wsdl). (These SOAP servers comprise the first tier of Vindicia's network, and it is the only tier that is publicly accessible.)

This manual, the ***CashBox API Guide***, lists and describes the Objects available in the CashBox solution, and provides pseudo-code examples.

# CashBox API Overview

Each CashBox object consists of data members and methods that operate on those members. The data members fall into one of the following categories:

- Standard, built-in data types, such as integers or strings, that are common to programming languages.
- Enumerations, which are scalar types coded as standard data types, but which are restricted to a specific set of legal values.
- Data structures, which consist of multiple data members, each of which can be of different data types.
- Arrays, containing zero or more data elements, all of which must be the same data type.

A CashBox object's methods are functions that require one or more input arguments. Methods always return a code that indicates the success or failure of the function call. In the event of failure, the code value should provide clues on why the call failed.

The CashBox API is a structured language, and requires input parameters to be entered in the order shown. Parameters must be place-marked if not specified.

This guide presents Objects and their data members and methods alphabetically, for ease of reference. Variable parameters for the methods are presented in syntactical order.

## Input Parameters

The CashBox SOAP API requires input parameters to be entered in the order shown, and must be place-marked if not specified.

For example, if you wish to use the `Account.makePayment` method to enter a payment against an `Account`, and you wish to add a `note` without specifying the `invoiceId` or `overageDisposition`, you must enter `null` for those two parameters.

(See the `makePayment` method for details.)

To enter a payment of $37 against an `Account`, call

```
Account->makePayment($acct,
       $paymentMethod, 37, USD, null, null, "note")
```

Calling

```
Account->makePayment($acct, $paymentMethod, 37, USD, "note")
```

would result in a payment applied to `invoiceId` "note," with no note included, which is, most likely, an invalid call.

## The Return Object

All methods in the CashBox API return a `Return` object, which contains the return codes for the call.

The `Return` object contains three data members:

- `returnCode`: This data member contains a value that corresponds to a standard HTTP return code. For values of 400 or higher, assume that your call failed. The failure could be due to several reasons, such as an authentication failure or a CashBox failure to find any objects that match your input. See Table 1: Standard Return Codes for a list of the most common return codes.
- `returnString`: If `returnCode` indicates an error condition (a non-`200` return code), your application can check `returnString` for further information. Use the CashBox API to generate a log of `returnString`, to help you debug your application in the development and production phases.
- `soapId`: This ID is returned for certain calls to Vindicia, especially those made to submit a batch of data (for example, a batch of transactions or account activities) for ChargeGuard processing. This ID helps Vindicia track your batched data in Vindicia's system and, if the ID is available, you should log it in your application. If an incident arises that requires troubleshooting by Vindicia, a Vindicia representative might ask you for this ID to determine the status of your data.

Some return strings contain information specific to the call for which the return was generated. In some cases, these will take the format:

`Unable to load product by VID` ***input-VID:*** `No match.`

where ***input-VID*** specifies the object or call to which the return error applies.

In some cases, these will take the format:

`Unable to load product by VID` ***input-VID: error-description.***

where ***error-description*** more specifically explains the cause of the error. In both cases, variable text is displayed in bold-italic.

The following table lists and describes the most common return codes. If a method returns different return codes, they are listed with the method.

Table 1 Standard Return Codes

| Return Code | Description |
|---|---|
| 200 | The call succeeded. |
| 400 | Your call failed, which could be due to an authentication failure, invalid user input, or a CashBox failure to find any objects that match your input. |
| 403 | The Vindicia server cannot authenticate your request. |
| 500 | The Vindicia server encountered an internal error. That error could occur for various reasons, the most common being an incorrectly populated input object, especially when you are making the call from a client library whose language does not support strict data-type checking. For resolution, especially during the development phase, contact Vindicia Technical Support. |
| 503 | A Vindicia back-end service, such as a database, is unavailable. Retry your call later. |

# 1 The Account Object

When a customer registers on your website, use the CashBox API to create an Account object. The Account object defines your customer's account, that is, it encapsulates the data members and methods that enable you to populate and maintain a customer's account information. Before someone can successfully order a product from you and be billed for it, an Account object that represents that person's account with you must exist in CashBox. You may create an Account object independently, or while creating an AutoBill object for a one-time transaction or recurring billing.

| | |
|---|---|
| **Note:** | If you create an AutoBill and specify an Account that does not yet exist, CashBox will create the Account, and attach it to the AutoBill. |

## 1.1 `Account` Object Hierarchies

The CashBox `Account` object supports two-level account hierarchies for payment and reporting; you may define parent and children accounts. A parent can have multiple children, but a child may have only one parent, and a child may not be a parent to another account.

The CashBox SOAP API allows you to:

- Link existing Accounts as parent and child.
- Unlink Accounts as parent and child. (Linking and unlinking an account is audited.)
- Transfer Credits from a parent to a child, or from one child to another. (An audit trail is kept of credit transfers.)
- Have a parent pay for a child's AutoBill by adding a `PaymentMethod` owned by the Parent to the AutoBill which includes the child's Account. (The child receives the entitlements; the parent pays.)
- Return all AutoBills that an Account pays.
- Return all children, all siblings, or the full family of any Account.
- Return the transaction history of a parent Account's `PaymentMethod`.

## 1.2 `Account` Data Members

The following table lists and describes the data members of the `Account` object.

| **Note** | Some CashBox objects' data members are CashBox data types that are data structures, which contain multiple data members themselves. These data structures are often listed as Subobjects in the documentation that follows. |
|---|---|

Table 1-1 `Account` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `company` | string | The customer's company name, if specified. |
| `credit` | `Credit` | A read-only data member that holds credit types (tokens, time, currency) available to this `Account`. CashBox populates this in the `Account` object returned to you in response to API calls.<br>Do *not* directly set the value of this attribute. To manipulate credit available to this `Account`, use methods such as `grantCredit()` or `revokeCredit()`.<br>See the `Credit` Object Data Members. |
| `emailAddress` | string | The email address for this `Account` object, if specified. |
| `emailTypePreference` | `EmailPreference` | The CashBox enumerated data type that specifies whether to send email to `Account` as plain text or HTML.<br>See the `EmailPreference` Subobject. |
| `entitlements` | `Entitlement` | An array of Entitlements associated with this Account. Note that Account entitlement modifications must be made using `Account` methods such as `grantEntitlement` or `revokeEntitlement`. Entitlement modifications made by other means (i.e. `update`), will be silently ignored.<br>See Section 8.1: Entitlement Data Members. |
| `merchantAccountId` | string | **Required**. Your unique identifier for this `Account` object, such as a database ID, a user name, or an email address. Once you have created the object with this ID, you may refer to the `Account` using the ID for future operations. |
| `name` | string | The customer's name, if specified. This name usually corresponds to the name on the credit card listed for the Account. |
| `nameValues` | `NameValuePair[]` | **Optional.** An array of name–value pairs associated with the customer for later reference.<br>See Section 10: The NameValuePair Object. |

Table 1-1 `Account` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `paymentMethods` | `PaymentMethod[]` | A list of default methods, one of which will be applied to a recurring transaction generated for this `Account` object if the customer has not explicitly specified a payment method for a subscription (`AutoBill`). Mark the payment methods active or inactive and sort them in order of preference. The first `paymentMethod` in the sort order will be used as the default.<br>See Section 11.1: PaymentMethod Data Members. |
| `preferredLanguage` | string | The customer's preferred language for communications. This preference is set in the customer account and must adhere to the W3C IANA Language Subtag Registry standard. Even though CashBox also supports the ISO-639.2 standard, the IANA Language Subtag Registry is the most recent and complete standard and is preferred.<br>If you use CashBox's email notification feature, and have uploaded an email template in the preferred language to CashBox, CashBox notifies the customer in this language. |
| `shippingAddress` | `Address` | The customer's shipping address. This field is optional if, for example, it is the same as `billingAddress`. CashBox looks up this address first when calculating a transaction's sales tax for this `Account` object.<br>See Section 3.1: Address Data Members. |
| `taxExemptions` | `TaxExemption[]` | An array of default exemptions for the sales tax on this Account's transactions. Multiple tax exemptions may be defined.<br>See the `TaxExemption` Subobject. |
| `tokenBalances` | `TokenAmount[]` | An array of `TokenAmount` objects that describes the account balance of various Token types. Each object in the array specifies the quantity of a specific type of Token. This is a read-only attribute, returned in the `Account` object in response to an `update()` call.<br>See the `TokenAmount` Subobject. |
| `VID` | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `Account` object, leave this field blank; it will be automatically populated by CashBox. |
| `warnBeforeAutoBilling` | `Boolean` | A Boolean flag that, if set to `true`, and if you are using CashBox's email notification feature, triggers an email notification to the customer before every recurring billing. |

# 1.3 Account Subobjects

The Account object has several subobjects:

- Credit Subobject
- CurrencyAmount Subobject
- EmailPreference Subobject
- TaxExemption Subobject
- TimeInterval Subobject
- TokenAmount Subobject

## Credit Subobject

An array of Credit amounts. Credit may be currency, time, or Tokens.

Table 1-2 Credit Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| currency-Amounts | CurrencyAmount | An array of CurrencyAmount objects. |
| timeIntervals | TimeInterval | An array of TimeInterval objects, each of which specifies a unit of time (day, week, month, year) and its amount. |
| tokenAmounts | TokenAmount | An array of TokenAmount objects. Each TokenAm-ount object specifies a Token Type, and the number of tokens of that type to be credited. A Token object must exist before being used in a Credit object. |

## CurrencyAmount Subobject

Defines the Currency Credit.

Table 1-3 CurrencyAmount Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| amount | decimal | The amount of currency granted. Must be a positive value. |
| currency | string | The ISO 4217 currency code used for the currency amount. Default is USD. |
| description | string | A description of the currency grant. |
| nameValues | NameValuePair | An optional array of name-value pairs to associate with this currency credit.<br>See Section 10: The NameValuePair Object. |
| reason | string | The reason for the currency credit. |
| sortValue | integer | Used to determine the order in which Credit is redeemed. |
| VID | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new CurrencyAmount object, leave this field blank; it will be automatically populated by CashBox. |

## EmailPreference Subobject

Allows you to set whether the Account prefers to receive HTML or plain text emails.

Table 1-4 EmailPreference Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| html | string | The customer prefers to receive email in HTML format. |
| multipart | string | The customer prefers to receive email messages in mixed media format.<br>**Note:** CashBox does not yet support this value; it is a placeholder for future implementation. |
| plaintext | string | The customer prefers to receive email in plain text format. |

## TaxExemption Subobject

Describes an Account-specific tax exemption.

An Account may have several Tax Exemptions. If the country specified in the TaxRegion data member of the TaxExemption object matches the country in which a Transaction occurs, the Transaction is exempted, and no tax is applied. This exemption will override any otherwise applicable taxes for the Transaction.

Table 1-5 TaxExemption Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| active | Boolean | If set to true, specifies that the exemption is active and serves as a criterion for calculation of sales tax. |
| exemptionId | string | Specifies the type of exemption, such as the U.S. Tax ID or value-added tax (VAT) ID. |
| region | TaxRegion | Specifies the geographical region for the tax exemption. TaxRegion is the ISO-3166-1 two-letter code for the country (for example, US, GB, or FR), for which CashBox computes sales tax. |

## TimeInterval Subobject

Defines the Time Interval Credit.

Table 1-6 TimeInterval Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| amount | integer | Amount of time to be credited. |
| description | string | A description of the time interval grant. |
| nameValues | NameValuePair | An optional array of name-value pairs to associate with this time-interval credit.<br>See Section 10: The NameValuePair Object. |
| reason | string | The reason for the time grant. |
| sortValue | integer | Used to determine the order in which Credits are redeemed. |
| type | TimeInterval-Type | Unit of time in which this time duration is specified. Possible values for TimeIntervalType are:<br>• Day<br>• Week<br>• Month<br>• Year |
| VID | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new TimeInterval object, leave this field blank; it will be automatically populated by CashBox. |

## `TokenAmount` Subobject

Defines the Token Credit.

Table 1-7 `TokenAmount` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `amount` | integer | The number of Tokens to be credited. |
| `token` | `Token` | The Type of Token for this Credit.<br>See Section 17.1: Token Data Members. |

## 1.4 Account Methods

The following table summarizes the methods for the Account object.

Table 1-8 Account Object Methods

| Method | Description |
|---|---|
| addChildren | Creates a parent-child relationship. |
| decrementTokens | Deducts from this Account object the specified number of tokens of various token types. This method is equivalent to tokenTransaction with negative values for the token amounts. |
| extendEntitlementByInterval | Extends an Account entitlement by the interval specified.<br>(The entitlement must already exist and be on the Account when this method is called.) |
| extendEntitlementToDate | Extends an account entitlement to the date specified.<br>(The entitlement must already exist and be on the Account when this method is called.) |
| fetchAllCreditHistory | Returns all credit grants and decrements for all Accounts. |
| fetchByEmail | Returns the Account objects with the specified email address. |
| fetchByMerchantAccountId | Returns the Account with the specified ID (merchantAccountId). |
| fetchByPaymentMethod | Returns all Account objects with the specified payment method. Identify the payment method with the VID, your payment method ID, or a unique identifier for the payment method type, such as a credit-card account number if the payment method type is credit card. |
| fetchByVid | Returns the Account object with the specified VID. |
| fetchByWebSessionVid | Returns the Account object with the specified WebSession VID. |
| fetchCreditHistory | Returns an audit log of credit-related events for an Account, or for all Accounts. |
| fetchFamily | Returns the family of the given Account. |
| grantCredit | Adds a specified amount of credit to an Account. |
| grantEntitlement | Grants entitlement to an Account. |
| incrementTokens | Adds the specified number of tokens to this Account. This method is equivalent to tokenTransaction with positive values for the token amounts. |
| isEntitled | Determines whether or not an Account has an entitlement. This checks Account entitlements, as well as entitlements associated with the Account's AutoBills. |
| makePayment | Enters a payment against the Account. |
| redeemGiftCard | Redeems a specified gift card and adds the corresponding credit to an Account. |

Table 1-8 `Account` Object Methods *(Continued)*

| Method | Description |
|---|---|
| `removeChildren` | Removes a child or multiple children from a parent. |
| `reversePayment` | Reverses an `Account` payment made using `makePayment`. This method may *only* be used with payments using `MerchantAcceptedPayment` payment methods. |
| `revokeCredit` | Deducts a specified amount of credit from the `Account`. |
| `revokeEntitlement` | Revokes an entitlement from the `Account`.<br>**Note:** This method will revoke only those `Entitlements` granted using the `grantEntitlement` method; it will not revoke entitlements acquired through an `AutoBill`. |
| `stopAutoBilling` | Cancels one or more `AutoBill` objects (subscriptions) associated with this `Account` object. |
| `tokenBalance` | Returns the balance of tokens of the specified type for this `Account`. If no type is specified, returns the balances for all the token types in the object. |
| `tokenTransaction` | Performs one or more token transactions, which can be on multiple token types, on this `Account`. The transactions may be positive, increasing the token balance; or negative, reducing the token balance. |
| `transfer` | Merges the target `Account` with a given (source) `Account`, and returns the target `Account` with the merged content. |
| `transferCredit` | Transfers credits from one `Account` to another. |
| `update` | Creates or updates an `Account` object. |
| `updatePaymentMethod` | Updates a payment method for this `Account` object. Call this method to update the payment methods on the active subscriptions (`AutoBill` objects) associated with this `Account`. |

## addChildren

This method adds one or more child `Accounts` to a parent `Account` using an input array of child `Accounts`.

### Input

***parent:*** the `Account` that will be parent to these children.

***child:*** an array of the child or children `Accounts` to attach to this parent `Account`.

***force:*** a Boolean flag that, if set to `true`, replaces any parents that these children may already have.

***payerReplacementBehavior:*** an action to take on methods that might, as a side effect, change who pays for an `Account`, for example: `Account.addChildren`.

`payerReplacementBehavior` may be one of the two following strings:

| | |
|---|---|
| `ReplaceOnAllAutoBills` | This option specifies that any AutoBills that the child has, or will have, are to be paid by the parent Account. |
| `ReplaceOnlyFutureAuto-Bills` | This option specifies that all future AutoBills for the child Account are to be paid by the parent Account. Existing AutoBills will be left as is. |

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***childAdded:*** the array of `Accounts` added.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
// to add children to an existing account

$parentAcct = new Account();

// account id for an existing account that will be the parent
$parentAcct->setMerchantAccountId('dad-101);

// existing accounts that will be the children
$childAcct1 = new Account();
$childAcct1->setMerchantAccountId('son-101);
$childAcct2 = new Account();
$childAcct2->setMerchantAccountId('son-102);

// want to replace existing parent of children, if any
$force = true;

// Future autobills for the children will be paid using
// parent's payment method
$payerReplacementBehavior = 'ReplaceOnlyFutureAutoBills';

$response = $parentAcct->addChildren(
   array($childAcct1, $childAcct2),
   $force,
   $payerReplace);

if ($response['returnCode'] == 200) {
   // children successfully added to the parent
}
else {
   // Error while adding the children
   print $response['returnString'] . "\n";
}
```

## decrementTokens

The `decrementTokens` method deducts the specified number of tokens, of named token types, from the `Account` object. Before calling `decrementTokens`, call `tokenBalance()` to verify that there are enough tokens of the specified type to fulfill the call. Use `decrementTokens` to deduct tokens from an `Account` object without conducting a formal CashBox transaction.

### *Input*

***account:*** the `Account` object from which to deduct tokens. Use the `merchantAccountId` or `VID` to identify the object.

***tokenAmounts:*** an array of one or more `TokenAmount` objects, each of which specifies the type of token to deduct and its quantity. The quantity must be a positive number. Before calling `decrementTokens`, you must have created the token types.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***tokenAmounts:*** an array of one or more `TokenAmount` objects, each of which specifies a type of token, and its balance (quantity) in the `Account` object, if the call succeeds.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

## *Example*

```
// to deduct tokens from an existing account

$acct = new Account();

// Reference an existing account from which tokens are to be deducted
$acct = new Account();
$acct->setMerchantAccountId('9876-5432');

// Refer to an existing token type using its id
$tok = new Token();
$tok->setMerchantTokenId("US_FREQ_BOOK_BUYER_PT");

// create a TokenAmount object and populate it with token type and
// quantity
$tokAmt = new TokenAmount();
$tokAmt->setToken($tok);
$tokAmt->setAmount(2);
// Refer to another existing token type using its id
$tok2 = new Token();
$tok2->setMerchantTokenId("US_FREQ_DVD_BUYER_PT");

// create a TokenAmount object and populate it with token type and
// quantity
$tokAmt2 = new TokenAmount();
$tokAmt2->setToken($tok2);
$tokAmt2->setAmount(2);

$tokAmounts = array($tokAmt, $tokAmt2);

// make the SOAP call to decrement tokens

$response = $acct->decrementTokens($tokAmounts);

if($response['returnCode']==200) {
    // the call returns new token balances on the account
    // print those out
    $newTokBalances = $response['tokenAmounts'];
    foreach ($newTokBalances as $newTokBal) {

        print "Token type" . $newTokBal->token->merchantTokenId . "\n";
        print "Token amount available" . $newTokBal->amount . "\n";
    }

}
```

## extendEntitlementByInterval

The `extendEntitlementByInterval` method extends an `Account` entitlement by the interval provided.

(The entitlement must already exist and be on the `Account` when this method is called.)

### Input

***account:*** the `Account` to which this extension applies.

***entitlement:*** an object of type `Entitlement` for the given `Account`.

***merchantEntitlementId:*** the merchant's unique ID for this entitlement. This may be specified in lieu of the full `Entitlement` object. Note that either the `Entitlement` or the `merchantEntitlementId` must be specified.

***interval:*** the extension interval to be applied to entitlement.

***note:*** an optional memo regarding the entitlement extension.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***account:*** the `Account` object with modified entitlements.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Entitlement not specified.`<br>• `Base Account not specified.`<br>• `Extension interval not specified.`<br>• `Account not found.`<br>• `Entitlement not found.`<br>• `Entitlement extension failed:` ***error-description.***<br>• `Failed to save Account after entitlement extension:` ***error-description.***<br>• `Failed to reload account after entitlement extension:` ***error-description.*** |

### *Example*

```
// to extend entitlements by 2 days

$acct = new Account();
$acct->setMerchantAccountId('xyz123');

$interval = new TimeInterval();
$interval->setType('Day');
$interval->setAmount(2);

$entitle = new Entitlement();
$entitle->setDescription('For playing Scrabble');
$entitle->setStartTimestamp($today);
$entitle->setEndTimestamp($tomorrow);
$entitle->setMerchantEntitlementId('bac');

$acct->grantEntitlement($entitle);

$response = $acct->extendEntitlementByInterval(
   $entitle,
   null,
   $interval,
   'Extended by 2 days'
);

// check $response ...
```

## extendEntitlementToDate

The `extendEntitlementToDate` method extends an `Account` entitlement to the date provided.

(The entitlement must already exist and be on the `Account` when this method is called.)

### Input

***account:*** the `Account` to which this extension applies.

***entitlement:*** an object of type `Entitlement` for the given `Account`.

***merchantEntitlementId:*** the merchant's unique ID for this entitlement. This may be specified in lieu of the full `Entitlement` object. Note that either the `Entitlement` or the `merchantEntitlementId` must be specified.

***extensionDate:*** the new end time for entitlement.

***note:*** an optional memo regarding the entitlement extension.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***account:*** the `Account` object with modified entitlements.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Entitlement not specified.`<br>• `Base Account not specified.`<br>• `Extension date not specified.`<br>• `Account not found.`<br>• `Entitlement not found.`<br>• `Failed to convert extension date:` ***error-description.***<br>• `Entitlement extension failed:` ***error-description.***<br>• `Failed to save Account after entitlement extension:` ***error-description.***<br>• `Failed to reload account after entitlement extension:` ***error-description.*** |

### *Example*

```
// to extend entitlements to a given date

$acct = new Account();
$acct->setMerchantAccountId('xyz123');

$entitle = new Entitlement();
$entitle->setDescription('For playing Scrabble');
$entitle->setStartTimestamp($today);
$entitle->setEndTimestamp($tomorrow);
$entitle->setMerchantEntitlementId('bac');

$acct->grantEntitlement($entitle);

$next_friday = '2011-08-12T23:59:59Z';

$response = $acct->extendEntitlementToDate(
   $entitle,
   null,
   $next_friday,
   'Extended until next friday'
);

// check $response ...
```

## fetchAllCreditHistory

The `fetchAllCreditHistory` method returns all Credit events that match the input ***timestamp*** parameters, for all Accounts.

CashBox maintains a log of credit-related events for each account. This log keeps track of events such as credit granted, revoked, consumed, or earned from a gift card redemption. Retrieve the audit log by calling the `fetchAllCreditHistory` or `fetchCreditHistory` methods for the `Account` or `AutoBill` objects.

The following table describes data members of the `CreditEventLog` object.

Table 1-9 `CreditEventLog` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `credit` | `Credit` | The `Credit` object used during a credit-related action or event.<br>See the `Credit` Subobject. |
| `note` | `string` | A memo regarding the Credit event. |
| `timeStamp` | `dateTime` | Time when this credit related action or event took place. |
| `type` | `CreditEventType` | Type of this credit related action or event. Use this to decide whether this action or event incremented or decremented credit.<br>See Table 1-9: `CreditEventLog` Object Data Members. |

Table 1-10 `CreditEventType` Object Enumeration Values

| Value | Description |
|---|---|
| `Consumption` | Credit decremented due to use in a recurring or one time transaction |
| `GiftCardRedemption` | Credit added due to a redemption of a gift card. |
| `GiftCardReversal` | Credit decremented due to a reversal of a gift card that was previously redeemed. |
| `GiftCardStatusInquiry` | No change in credit. |
| `Grant` | Credit added due to a credit grant you made. |
| `Refund` | Credit added due to refund of a credit based transaction. |
| `Revocation` | Credit decremented due to a credit revocation you made. |

(For more information, see the `fetchCreditHistory` method below.)

## Input

***timestamp:*** the starting timestamp (lower limit) for the range of credit event logs you wish to retrieve.

***endTimestamp:*** the ending timestamp (upper limit) for the range of credit event logs you wish to retrieve.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***creditEventLogs:*** the array of Credit events (grants and deductions) with a timestamp and event type.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load account.`<br>• `No matching credit events found.`<br>• `Invalid value or values of timestamp, and/or page, and/or page size.` |

### *Example*

```
// to fetch all credit history for an account

$acct = new Account();

// account id for an existing customer whose
// credit history you want to retrieve

$acct->setMerchantAccountId('jdoe101');

$page = 0; // paging begins at 0
$pageSize = 5; // five records

do {
    $ret =
        $acct->fetchAllCreditHistory($page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedLogs = $ret['creditEventLogs'];
        $count = sizeof($fetchedLogs);
        foreach ($fetchedLogs as $log) {
            $credit = $log->getCredit();
            $ts = $log->getTimeStamp();
            $eventType = $log->getType();
            // process retrieved credit event log
            // details here.
        }
        $page++;
    }
} while ($count > 0);
```

## fetchByEmail

The `fetchByEmail` method returns an `Account` object whose email address matches the input. If you use an email address as an identifier for your customers, you may call this method to retrieve an `Account` object.

### Input

***emailAddress:*** the `Account` object's email address, which serves as the search criterion.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***accounts:*** the most recently modified `Account` object whose email address matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | `Must specify email address to load by!` |
| `404` | One of the following:<br>• `Unable to load account by emailAddress` ***input-emailAddress:*** `No match.`<br>• `No AutoBills found for email address` ***input-emailAddress*****:** `No match.`<br>• `Unable to load account by email address` ***input-emailAddress*****:** `No match.` |

### Example

```
// Create an account object to make the SOAP call
$account = new Account();

// now load a customer account into the account object
$response = $account->fetchByEmail('somebody@yahoo.com');
if($response['returnCode'] == 200) {
    $fetchedAccount = $response['data']->account;

    foreach $fetchedAcct ($fetchedAccount) {
        // process a fetched account
    }
}
else {
    // The call was unsuccessful
    print "Return code: " . $response['returnCode'] . "\n";
    print "Return string: " . $response['returnString'] . "\n";
}
```

## fetchByMerchantAccountId

The `fetchByMerchantAccountId` method returns an `Account` object whose ID (the `merchantAccountId` assigned by you) matches the input. When you first create an `Account` object in the Vindicia database with the `update` method, specify a unique value for the `merchantAccountId` field of that object. Best practice suggests that the `merchantAccountId` value map directly to the customer's unique ID in your own database.

### *Input*

***merchantAccountId:*** your Account ID (`merchantAccountId`), which serves as the search criterion.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***account:*** the `Account` object whose ID assigned by you (`merchantAccountId`) matches the input.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load account by VID` ***input-merchantAccountId:*** `No match.`<br>• `Unable to load account by VID` ***input-merchantAccountId:*** `Vindicia internal error.`<br>• `Must specify merchantAccountId to load by!` |

### *Example*

```
// Create a SOAP caller object
$account = new Account();
$accountId = "34583";

// now load an account into the Account object
// by (unique) Account ID
$response = $account->fetchByMerchantAccountId($accountId);
if($response['returnCode'] == 200) {
    $fetchedAccount = $response['data']->account;
}
else {
    // The call was unsuccessful
    print "Return code: " . $response['returnCode'] . "\n";
    print "Return string " . $response['returnString'] . "\n";
}
```

## fetchByPaymentMethod

The `fetchByPaymentMethod` method returns all `Account` objects with a payment method that matches the input. Use this method to conduct global searches, such as “all the accounts that use a certain credit card as the payment method.”

This method supports paging to limit the number of records returned per call. Occasionally, returning a large number of records in one call swamps buffers and might cause a failure. Vindicia recommends that you call this method in a loop, incrementing the page for each loop iteration with an optimal page size (number of records returned in one call) until the page contains a number of records that is less than the given page size.

### *Input*

***paymentMethod:*** an object of type `PaymentMethod`, which serves as the search criterion. Identify the payment method with its VID, your payment method ID (`merchantPaymentMethodId`), or one of the following, depending on the payment method type:

- The account number for a credit card.
- The account number-bank routing number combination for ACH and ECP.
- The fiscal number for a Boleto.
- The `PaypalEmail` for PayPal.

**Note:** If you use SOAP releases prior to 3.5, you will not be able to search accounts using the PayPal payment method. If you use SOAP 3.6.0 or later, you can search accounts and transactions using `PaypalEmail`.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***accounts:*** one or more `Account` objects whose payment method matches the input.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 404 | No matching accounts. |
| 400 | One of the following:<br>• Payment method type is credit card, but credit card information is incomplete.<br>• Payment method type is ECP, but ECP account and routing information is incomplete.<br>• Payment method type is Boleto, but Boleto payment information is incomplete.<br>• Payment method type is currently not supported.<br>• Must specify a PaymentMethod object, a non-negative page number, and a page size greater than 0. |

## Example

```
$pm = new PaymentMethod();
$pm->setType('CreditCard');
$cc = new CreditCard();
$cc->setAccount('4111111111111111');
// this is the card number we want to search by
$cc->setExpiration('201108');
$pm->setCreditCard($cc);

$acct = new Account();
$page = 0;
$pageSize = 10; // max 10 records per page

do {
    $response = $acct->fetchByPaymentMethod($pm, $page, $pageSize);

    if($response['returnCode']==200) {
        $accounts = $response['data']->accounts;

        foreach ($accounts as $account) {
        // process each account found here
        print "Found account with id: "
            . $account->getMerchantAccountId() . "\n";
        }
    }
    $page++
} while (count($accounts) == $pageSize);
```

## fetchByVid

The `fetchByVid` method returns an `Account` object whose VID matches the input. When you first create an `Account` object with the `update` method, leave the `VID` field empty; CashBox automatically assigns the object a VID. For convenience, store the VID in your application so that you can retrieve or refer to that object with its VID later. If you do not assign unique account IDs (`merchantAccountId`) yourself, you may identify `Account` objects with their VIDs.

### Input

***vid:*** the `Account` object's Vindicia identifier, which serves as the search criterion.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***account:*** the `Account` object whose VID matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load account by VID` ***input-vid:*** `No match.`<br>• `Unable to load account by VID` ***input-vid:*** `Vindicia internal error.`<br>• `Must specify VID to load by!` |

### Example

```
$accountVid = 'MyVindiciaAccountVid';

// Create a SOAP caller object
$account = new Account();
$accountVID = "36c8de2cb74b2c2b08b259cf231ac8d90d1bb3b8";

// now load a customer account into the account object by VID
$response = $account->fetchByVid($accountVid);
if($response['returnCode'] == 200) {
    $fetchedAccount = $response['data']->account;
}
else {
    // The call was unsuccessful
    print "Return code: " . $response['returnCode'] . "\n";
    print "Return string: " . $response['returnString'] . "\n";
}
```

## fetchByWebSessionVid

Use Vindicia's Hosted Order Automation (HOA) feature to create CashBox objects that contain sensitive payment information, such as credit-card account numbers. Using HOA, you may have your customers submit their data through a specially designed Web order form, accessed from your server, which allows you to store credit card numbers directly on Vindicia's servers. Because HOA completely bypasses your server at form submission, your PCI compliance efforts may be mitigated. See Chapter 13: Hosted Order Automation in the ***CashBox Programming Guide*** for details on HOA.

Within your HOA implementation, call the `fetchByWebSessionVid` method to retrieve the `Account` object created by HOA on Vindicia's servers when a customer submits an order form that results in a one-time or recurring bill. You must also create a `WebSession` object on Vindicia's servers before serving the form to your customer to track the form's submission to Vindicia. For details, see Section 19: The WebSession Object.

The `WebSession` object's VID serves as the tracking ID for various activities, from serving the order form to a customer, to returning a success or failure page to that same customer. The success page to which HOA redirects the customer's browser after successfully processing the data is the order form. On that page, the `WebSession` object's VID is available to you because HOA passes it during the redirection. In turn, you can pass that VID as the input parameter to this call and retrieve the `Account` object created by HOA. Finally, you can extract the contents of the `Account` object and include them, as appropriate, in the success page to be returned to the customer.

### *Input*

***vid:*** the `WebSession` object's Vindicia unique identifier for tracking the submission of the order form.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***account:*** an `Account` object created by HOA as a result of an order form submitted by a customer.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Missing required parameter 'vid'. |
| 404 | Unable to find requested Account: No matches. |

## *Example*

```
// To call the fetchByWebSessionVid on a success web page:

$webSessionVid = …; //passed in by redirected page
$soap = new WebSession();
$response = $soap->fetchByVID($webSessionVid);

if ($response['returnCode'] == 200) {
   $fetchedWs = $response['data']->session;
   // check if the CashBox API call made by HOA was successful
   $retCode = $fetchedWs->apiReturn->returnCode;

   if ($retCode == 200) {

      // Assuming HOA created an Account object, let's fetch it

      $soapAcct = new Account($soapLogin, $soapPwd);
      $resp = $soapAcct->fetchByWebSessionVid($webSessionVid);

      if ($resp['returnCode'] == 200) {
         $createdAccount = $resp['data']->account;

         // Get Account contents here to be included in
         // HTML returned to the customer.
      }
      else {
         // Return error message to customer
      }
   }
   else {
      // return failure page to customer
   }

}
else {
   // Return error message to the customer
}
```

## fetchCreditHistory

The `fetchCreditHistory` method returns `creditEventLogs` for the Account.

For more information, please see the `Account` object's `fetchAllCreditHistory` method.

### Input

***account:*** the (optional) `Account` object for which you wish to retrieve credit event history. You may populate only the `merchantAccountId` or `VID` in this object so that CashBox can locate it in its database. Leave this variable blank if you wish to fetch credit history across all Accounts.

***timestamp:*** the starting timestamp (lower limit) for the range of credit event logs you wish to retrieve.

***endTimestamp:*** the ending timestamp (upper limit) for the range of credit event logs you wish to retrieve.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to return per call. This value must be greater than 0.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***creditEventLogs:*** an array of `CreditEventLog` objects. Each of these objects describes a specific credit-related event or action associated with the input `Account`. For more information, see Table 1-9: `CreditEventLog` Object Data Members.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load account.`<br>• `Invalid value or values of timestamp, and/or page, and/or page size.` |
| `404` | `No matching credit events found.` |

## *Example*

```
// to fetch credit history for an account

$acct = new Account();

// account id for an existing customer whose
// credit history you want to retrieve

$acct->setMerchantAccountId('jdoe101');

$page = 0; // paging begins at 0
$pageSize = 5; // five records
$startTime = '2010-01-01T22:34:32.265Z';
$endTime = '2010-01-30T22:34:32.265Z';

do {
    $ret =
        $acct->fetchCreditHistory($startTime, $endTime, $page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedLogs = $ret['creditEventLogs'];
        $count = sizeof($fetchedLogs);
        foreach ($fetchedLogs as $log) {
            $credit = $log->getCredit();
            $ts = $log->getTimeStamp();
            $eventType = $log->getType();
            // process retrieved credit event log
            // details here.
        }
        $page++;
    }
} while ($count > 0);
```

## fetchFamily

The `fetchFamily` method returns the children of the given `Account`.

See the input parameters for the ways in which to specify the payment methods. Use this method to conduct searches for all the accounts that have a familial relationship, that is, parent-to-child, donor-to-recipient, or sibling-to-sibling.

- For a parent account, get all the children (and return the parent and those children).
- For a child account, get the parent and all the siblings.

### *Input*

***account:*** the `Account` for which you wish to find the parent and/or sibling `Accounts`.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***parent:*** the parent `Account` for this family.

***child:*** the child or children `Accounts` in this family.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$soapCaller = new Account();
$childAcct1 = new Account();

// to fetch the family of this child
$childAcct1->setMerchantAccountId('son-101);

$response = $childAcct1->fetchFamily();

if ($response['returnCode'] == 200) {
   $fetchedParent = $response['parent'];
   print "Parent account id: ";
   print $fetchedParent->getMerchantAccountId() . "\n";

   $fetchedChildren = $response['child'];

   if ($fetchedChildren != null) {
      foreach($fetchedChildren as $fetchedChild) {
      print "Child account id: ";
      print $fetchedChild->getMerchantAccountId() . "\n";
   }
}
else {
   // Error while fetching the family
   print $response['returnString'] . "\n";
}
```

## grantCredit

The grantCredit method adds credit to an Account object. With credit available to an Account, you can conduct a one-time transaction for the Account. If the Account is associated with an AutoBill, and if the AutoBill has no associated credit, CashBox can draw credit down from the Account to sustain the AutoBill.

Specify credit you wish to grant to the Account as a Credit object. Time-based credit cannot be granted to an Account.

See the Credit Subobject, and the TimeInterval Subobject, for more information.

See Chapter 12: Credit Grants and Gift Cards in the ***CashBox Programming Guide*** for more information on working with credit.

### *Input*

***account:*** the Account object to which you wish to grant credit. Use the merchantAccountId or VID to identify the object.

***credit:*** a Credit object specifying the amount and type of credit you wish to grant to the Account.

***note:*** an optional memo regarding the credit grant.

### *Output*

***return:*** an object Account type Return that indicates the success or failure of the call.

***account:*** the Account object to which you granted credit. This object contains the updated array of Credit objects.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• Account not found.<br>• Failed to translate credit ***error-description.***<br>• Failed to grant credit ***error-description.***<br>• Failed to save Account after granting credit.<br>• Failed to reload Account after granting credit ***error-description.***<br>• Time interval credit cannot have amount 0. |

### *Example*

```
// to grant credit to an account

$acct = new Account();

// account id for an existing customer

$acct->setMerchantAccountId('jdoe101');

$tok = new Token();

// specify id of an existing token type.
// assumption here is that you have already created
// a Token object with this id

$tok->setMerchantTokenId('ANYTIME_PHONE_MINUTES_2010');

$tokAmt = new TokenAmount();
$tokAmt->setToken($tok);
$tokAmt->setAmount(100);

$cr = new Credit();
$cr->setTokenAmounts(array($tokAmt));

// Now make the SOAP API call to grant credit to the acct
$response = $acct->grantCredit($cr);

if ($response['returnCode'] == 200) {

   // Credit successfully granted to the account

   $updatedAcct = $response['data']->account;
   $availableCredits = $updatedAcct->getCredit();
   $availableTokens = $availableCredits->getTokenAmounts();

   print "Available token credits: \n";
   foreach($availableTokens as $tkAmt) {
      print "Token type: " . $tkAmt->getMerchantTokenId() . " ";
      print "Amount: " . $tkAmt->getAmount() . "\n";
   }
}
else {

   // Error while granting credit to the account
   print $response['returnString'] . "\n";
}
```

## grantEntitlement

The `grantEntitlement` method grants entitlements to an `Account`.

### Input

***account:*** the `Account` to which this grant applies.

***entitlement:*** the `Entitlement` being granted.

***note:*** an optional memo regarding the entitlement grant.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***account:*** the `Account` with new entitlements.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Entitlement not specified.`<br>• `Base Account not specified.`<br>• `Account not found.`<br>• `Entitlement grant failed:` ***error-description.***<br>• `Failed to save Account after entitlement extension:` ***error-description.***<br>• `Failed to reload account after entitlement extension:` ***error-description.*** |

### Example

```
$acct = new Account();
$acct->setMerchantAccountId('xyz123');

$entitle = new Entitlement();
$entitle->setDescription('For playing Scrabble');
$entitle->setStartTimestamp($today);
$entitle->setEndTimestamp($tomorrow);
$entitle->setMerchantEntitlementId('bac');

$response = $acct->grantEntitlement($entitle);

// check $response
```

## incrementTokens

The `incrementTokens` method adds the specified number of tokens to the `Account` object. Call this method to grant tokens (for example, virtual currency, frequent flier miles, or cell-phone minutes) to an `Account` object without conducting a formal CashBox transaction. Use this method to grant `Tokens` which will *not* be used as currency within CashBox.

### Input

***account:*** the `Account` object to which to add tokens. Use the `merchantAccountId` or `VID` to identify the object.

***tokenAmounts:*** an array of one or more `TokenAmount` objects, each of which specifies the type of token to add and its quantity. The quantity must be a positive number. Token types must exist before being used in `incrementTokens`.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***tokenAmounts:*** an array of one or more `TokenAmount` objects, each of which specifies a type of Token available to the `Account` after this call, and its balance (quantity) in the `Account` object, if the call succeeds. In some cases, this return might not occur, especially if you have not previously defined the specified token type.

The following table lists and describes the data members of the `TokenAmount` object.

Table 1-11 `TokenAmount` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `amount` | `Integer` | The number of tokens. |
| `Token` | `Token` | The token type, which must be previously defined. |

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

## *Example*

```
// to increment tokens for an account

$acct = new Account();

// Reference an existing account to which the tokens are to be granted
$acct = new Account();
$acct->setMerchantAccountId('9876-5432');

// Refer to an existing token type using its id
$tok = new Token();
$tok->setMerchantTokenId("US_FREQ_BOOK_BUYER_PT");

// create a TokenAmount object and populate it with token type and
// quantity
$tokAmt = new TokenAmount();
$tokAmt->setToken($tok);
$tokAmt->setAmount(5); // award the Account with 5 tokens of this type

// Refer to another existing token type using its id
$tok2 = new Token();
$tok2->setMerchantTokenId("US_FREQ_DVD_BUYER_PT");

// create a TokenAmount object and populate it with token type and
// quantity
$tokAmt2 = new TokenAmount();
$tokAmt2->setToken($tok2);
$tokAmt2->setAmount(2); // award the Account with 2 tokens of this type

$tokAmounts = array($tokAmt, $tokAmt2);

// make the SOAP call to increment tokens

$response = $acct->incrementTokens($tokAmounts);

if($response['returnCode']==200) {
   // the call returns new token balances on the account
   // print those out
   $newTokBalances = $response['tokenAmounts'];
   foreach ($newTokBalances as $newTokBal) {

      print "Token type" . $newTokBal->token->merchantTokenId . "\n";
      print "Token amount available" . $newTokBal->amount . "\n";
   }
}
```

## isEntitled

The `isEntitled` method determines whether or not an `Account` has an entitlement at the moment the method is called. `isEntitled` returns a Boolean `true`/`false`, and does not return the length of time, past or future, for which the `Account` is entitled. This will check account entitlements, as well as entitlements associated with the Account's AutoBills.

### *Input*

***account:*** the `Account` to which this grant applies.

***merchantEntitlementId:*** the merchant's unique ID for this entitlement.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***entitled:*** `true` if the `Account` is entitled; `false` if the `Account` is not.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Entitlement not specified.`<br>• `Base Account not specified.`<br>• `Account not found.`<br>• `Entitlement test failed:` ***error-description.*** |

## *Example*

```
// to determine if an account is entitled

$acct = new Account();
$acct->setMerchantAccountId('xyz123');

$response = $acct->isEntitled('bac');

if ($response['returnCode'] == 200) {
    if ($response['data']->entitled) {
        // proceed
    }
}
else {
    // not entitled yet
}

$entitle = new Entitlement();
$entitle->setDescription('For playing Scrabble');
$entitle->setStartTimestamp($today);
$entitle->setEndTimestamp($tomorrow);
$entitle->setMerchantEntitlementId('bac');

$acct->grantEntitlement($entitle);

$response = $acct->isEntitled('bac');

if ($response['returnCode'] == 200) {
    if ($response['data']->entitled) {
        // proceed
    }
}
else {
    print 'Should be entitled!!';
}
```

## makePayment

The `makePayment` method allows you to record a payment against an outstanding invoice. This method may be used to enter check or cash payments, payment of goods in trade, or payments made with active Payment Methods.

Using the `makePayment` method on the `Account` object will cause CashBox to allocate the payment to the oldest open invoice or AutoBill. To apply a payment directly to an outstanding AutoBill, use `AutoBill.makePayment` instead.

Whether you use a standard `PaymentMethod`, or a `MerchantAcceptedPayment`, the `makePayment` method generates a `Transaction`, and processes the `Transaction` through the auth/capture cycle appropriate to the input Payment Method. Credit Card, ECP, PayPal, and other standard Payment Methods are routed through the appropriate Payment Processor. The `MerchantAcceptedPayment` Payment Method is routed through Vindicia's internal transaction process. Both Payment Method types appear as a Transaction in the Account's history.

### *Input*

***account:*** the `Account` to which this payment applies.

***paymentMethod:*** the `PaymentMethod` to be used for this payment. (**Note:** Assign a unique ID for every `Account.makePayment` call that uses a `MerchantAcceptedPayment` Payment Method, for tracking purposes.)

***amount:*** the amount of the payment being made. (Required Float.)

***currency:*** the ISO 4217 currency code for amount. This must match the currency used for charges on the current invoice. (If not specified, the AutoBill/Invoice currency will be used.)

***invoiceId:*** the ID of the Invoice to make payment against. If `null`, the default payment order will be used. (Array of `InvoiceIds`.)

For more information, see Section 9.3: Working with Invoices in the ***CashBox Programming Guide***.

***overageDisposition:*** defines how to allocate payments in excess of a required AutoBill payment amount. Defaults to `applyToOldestInvoice` if not specified.

> ***overageDisposition***: an object of type `PaymentOverageDisposition`, with values `applyToThisAutoBill`, `applyToOldestInvoice`, and `applyToCredit`.

***note:*** an optional memo regarding the payment made.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transaction:*** the `Transaction` object reflecting the payment.

***summary:*** an object of type `TransactionAttemptSummary` that includes the summary of the payment attempt.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• Account not found.<br>• Failed to translate payment method.<br>• Failed to make payment: ***error-description.***<br>• Transaction not returned from payment attempt. |
| 405 | Payment transaction failed - payment not applied. |
| 406 | Specified Account could not be found - payment not applied. |
| 407 | Specified PaymentMethod could not be found - payment not applied. |

## Example

```
$acct = new Account();
$acct->setMerchantAccountId('xyz123');

$paymentMethod = new PaymentMethod();
$paymentMethod->setType('CreditCard');
$paymentMethod->setAccountHolderName('Jane Doe');
$paymentMethod->setCustomerSpecifiedType('Visa');
$paymentMethod->setCurrency('USD');
$paymentMethod->setActive(true);

$cc = new CreditCard();
$cc->setAccount('411111111111111');
$cc->setExpirationDate('201208');
$paymentMethod->setCreditCard($cc);

$response = $acct->makePayment(
   $paymentMethod,
   200,
   'USD',
   'inv-charles',
   null,
   '200 bucks for Charles'
);

// check $response
```

## redeemGiftCard

The `redeemGiftCard` method redeems a gift card represented by the input `GiftCard` object, and grants the resultant amount of credit to the `Account`. This method should be called after the `statusInquiry()` method is called on the `GiftCard` object that you provide as input to this method. If the `statusInquiry()` method indicates that status of the `GiftCard` object is `Active`, then call this method. For more information, see the `Credit` Subobject.

For redemption of a gift card, CashBox contacts a gift card processor. (CashBox currently supports InComm.) If the gift card is redeemable, the processor returns an SKU or a UPC number. This number is unique for each type of gift card and is decided by a prior agreement between you and the gift card processor. CashBox uses the number to look up a `Product` object with the same `merchantProductId`. CashBox then grants credit to the `Account` as defined in the `creditGranted` attribute of the `Product` object. For each type of gift card you wish to accept, create `Product` objects with the appropriate amount of credit specified in their `creditGranted` attributes.

CashBox currently supports only full redemption of the credit associated with a gift card.

See Chapter 12: Credit Grants and Gift Cards in the ***CashBox Programming Guide*** for more information on gift card redemption.

### *Input*

***account:*** an `Account` object to which credit will be granted if redemption of the gift card is successful. Populate the `merchantAccountId` or `VID` in this object so that CashBox can locate it in its database.

***giftcard:*** a `GiftCard` object encapsulating information about the gift card you wish to redeem. For more information, see Section 9: The GiftCard Object. Call `statusInquiry()` before calling this method, to return the `VID` of the `GiftCard` object. Populate the `VID` in this object so that CashBox can look it up in its database.

***credit:*** a `Credit` object specifying the amount and type of credit you wish to redeem. (This input parameter is currently unsupported.)

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***giftcard:*** the `GiftCard` object with updated credit as granted by the gift card redemption.

***account:*** the `Account` object to which credit was granted if redemption of the gift card was successful.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• Account not found.<br>• Failed to translate gift card ***error-description.***<br>• Failed to redeem gift card ***error-description.***<br>• Failed to retrieve gift card after redemption attempt.<br>• Failed to save Account after gift card redemption attempt.<br>• Failed to reload Account after gift card redemption attempt ***error-description.***<br>• Redemption attempt failed for Gift Card ID ***gift-card-ID.*** |

## Example

```
// to redeem a gift card

$acct = new Account();

// account id for a customer's Account object for which the gift card
// will be redeemed, and credit added to the Account.

$acct->setMerchantAccountId('JDOE1234');

$gc = new GiftCard();

// set the VID of the gift card, obtained when we checked the
// status of the gift card, and determined that it is active

$gc->setVID($gcVID);

// Now make the SOAP API call to redeem the gift card

$response = $acct->redeemGiftCard($gc, null);

if ($response['returnCode'] == 200) {

   // Redemption successful. Check if credit was added to the account
   $updatedAcct = $response['data']->account;
   $availableCredits = $updatedAcct->getCredit();
   $availableTokens = $availableCredits->getTokenAmounts();

   print "Available token credits: \n";
   foreach($availableTokens as $tkAmt) {
      print "Token type: " . $tkAmt->getMerchantTokenId() . " ";
      print "Amount: " . $tkAmt->getAmount() . "\n";
   }

   // Also make sure status of the gift card is 'Redeemed'
   $updatedGc = $response['data']->giftcard;

   print "Status of the gift card: ";
   print $updatedGc->getStatus()->getStatus() . "\n";

}
else {

   // Error while granting credit to the account
   print $response['returnString'] . "\n";
}
```

## removeChildren

This method removes one or more child `Accounts` from the parent `Account`.

### Input

***parent:*** the `Account` that should be parent to these children.

***child:*** the child or children that should be removed from this parent account.

***payerReplacementBehavior:*** an object of type `AccountPayerReplacementBehavior`, that controls how existing `AutoBills` of the children are affected.

`AccountPayerReplacementBehavior` may contain the following strings:

| | |
|---|---|
| `ReplaceOnAllAutoBills` | This option will replace the Payment Method on each of the child's AutoBills with the child's default (index_number = 0) Payment Method.<br>If the child does not have any Payment Methods, CashBox will set the Payment Method ID on the child's AutoBills to null. When CashBox later tries to process a Transaction for one of these AutoBills, it will detect the absence of a Payment Method, and send an email to the (child) account. |
| `ReplaceOnlyFutureAutoBills` | This option will simply break the link between the parent and child Accounts, leaving the parent's Payment Methods unavailable to the child when creating new AutoBills.<br>If a parent/child relationship is broken in this manner, and a subsequent `AutoBill.update` call is made against one of the child's AutoBills, CashBox may detect that the Payment Method on the child's AutoBill is no longer associated with the child, and issue an error. |

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

## *Example*

```
$parentAcct = new Account();

// account id for an existing account that is the parent
$parentAcct->setMerchantAccountId('dad-101);
$childAcct2 = new Account();

// account id of an existing child
$childAcct2->setMerchantAccountId('son-102);

// On each of the children that were deleted, correct
// the existing autobills to use the child's payment
// method instead of the parent's for any of the old
// parent's payment methods.
$payerReplacementBehavior = 'ReplaceOnAllAutoBills';
$response = $parentAcct->removeChildren (array($childAcct2),$payerReplace);

if ($response['returnCode'] == 200) {
   // child successfully removed
}
else {
   // Error while removing the child
   print $response['returnString'] . "\n";
}
```

## reversePayment

The `reversePayment` method allows merchants to reverse payments made using the `makePayment` method. This method may only be used against payments made using the `MerchantAcceptedPayment` payment method.

### Input

***account:*** the `Account` to which this reversal applies.

***timestamp:*** the time that payment reversal occurred. Set the `timestamp` for record keeping purposes, to record when the payment reversal was accepted, rather than when it was recorded in CashBox. CashBox will reverse payments immediately, regardless of when the `timestamp` is set.

***paymentId:*** the `paymentId` of the `MerchantAcceptedPayment` used for this Payment. Either the `paymentId`, or the `invoiceId` (and optional `indexNumber`) must be specified.

> The `paymentId` is automatically set by CashBox when a payment is made to an `Invoice`, `AutoBill`, or `Account`. In reversing a payment, you must reference the appropriate `paymentId`.

***invoiceId:*** the ID of the Invoice associated with the payment reversal. Either the `paymentId`, or the `invoiceId` (and optional `indexNumber`) must be specified.

***indexNumber:*** the `indexNumber` of the payment item (on the `invoiceId` invoice) that is being reversed.

***note:*** an optional memo regarding the payment reversal.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Account not found.`<br>• `Neither paymentId nor invoiceId:` ***indexNumber*** `provided for reversal attempt.`<br>• `Failed to add reverse payment:` ***error-description.*** |

### *Example*

```
// to reverse a payment made using the makePayment method

$acct = new Account();
$acct->setMerchantAccountId('xyz123');

$paymentMethod = new PaymentMethod();
$paymentMethod->setMerchantPaymentMethodId($pmId); // for some $pmId
$paymentId = $paymentMethod->merchantAcceptedPayment->paymentId;

$response = $acct->reversePayment(
    $now,
    $paymentId,
    undef,
    undef,
    'Changed my mind.'
);

// check $response
```

## revokeCredit

The `revokeCredit` method deducts credit from an `Account` object. If the deduction results in a negative amount for a certain type of credit, CashBox sets its balance to 0. This method returns the `Account` object with resultant credit balance.

Specify the amount and type of Credit you wish to revoke from the `Account` as a `Credit` object.

To revoke a specific credit grant, specify the VID of the `Credit` object you wish to revoke. If you do not specify a VID, CashBox will revoke credit in the order in which it would redeem Credits to fulfill an AutoBill Transaction, until the total amount specified is revoked. This process might revoke a partial Credit, a single Credit, or multiple Credits.

For more information on working with credit, see Chapter 12: Credit Grants and Gift Cards in the ***CashBox Programming Guide***.

### *Input*

***account:*** the `Account` object from which you wish to revoke credit. Use the `merchantAccountId` or `VID` to identify the object.

***credit:*** a `Credit` object specifying the amount and type of credit you wish to deduct from the `Account`. For more information, see the `Credit` Subobject.

***note:*** an optional memo regarding the credit revocation.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***account:*** the `Account` object from which you revoked credit. This object contains the updated array of `Credit` objects.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Account not found.`<br>• `Failed to translate credit` ***error-description.***<br>• `Failed to revoke credit` ***error-description.***<br>• `Failed to save Account after revoking credit.`<br>• `Failed to reload Account after revoking credit` ***error-description.***<br>• `Data validation error: Missing required parameter` ***credit.*** |

## *Example*

```
// to revoke credit from an account

$acct = new Account();

// account id for an existing customer

$acct->setMerchantAccountId('ff_flier_101');

$tok = new Token();

// specify id of an existing token type.
// assumption here is that you have already created
// a Token object with this id

$tok->setMerchantTokenId('UA_FF_MILES');

$tokAmt = new TokenAmount();
$tokAmt->setToken($tok);
$tokAmt->setAmount(25000);

$cr = new Credit();
$cr->setTokenAmounts(array($tokAmt));

// Now make the SOAP API call to deduct miles
$response = $acct->revokeCredit($cr);

if ($response['returnCode'] == 200) {

    // Credit successfully revoked from the account
    $updatedAcct = $response['data']->account;
    $availableCredits = $updatedAcct->getCredit();
    $availableTokens = $availableCredits->getTokenAmounts();

    print "Available token credits: \n";
    foreach($availableTokens as $tkAmt) {
        print "Token type: " . $tkAmt->getMerchantTokenId() . " ";
        print "Amount: " . $tkAmt->getAmount() . "\n";
    }
}
else {

    // Error while revoking credit from the account
    print $response['returnString'] . "\n";
}
```

## revokeEntitlement

The `revokeEntitlement` method revokes entitlement from an `Account`.

> **Note:** This method will revoke only those `Entitlements` granted using the `grantEntitlement` method; it will not revoke entitlements acquired through an `AutoBill`.

### *Input*

***account:*** the `Account` to which this revocation applies.

***entitlement:*** the `Entitlement` object to be revoked.

***merchantEntitlementId:*** the merchant's unique ID for this entitlement. This may be specified in lieu of the full `Entitlement` object. Note that either the `Entitlement` or the `merchantEntitlementId` must be specified.

***note:*** an optional memo regarding the entitlement revocation.

### *Output*

***account:*** the `Account` with entitlements revoked.

***return:*** an object of type `Return` that indicates the success or failure of the call.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Entitlement not specified.`<br>• `Base Account not specified.`<br>• `Account not found.`<br>• `Entitlement revocation failed: Could not find active entitlement for revocation.`<br>• `Entitlement revocation failed:` ***error-description.***<br>• `Failed to save Account after revoking entitlement:` ***error-description.***<br>• `Failed to reload account after entitlement revocation:` ***error-description.*** |

## *Example*

```
// to revoke an entitlement from an account

$acct = new Account();
$acct->setMerchantAccountId('xyz123');

$response = $acct->revokeEntitlement(
   null,
   'bac',      // the Id for playing Scrabble
   'You can play no more'
);

if ($response['returnCode'] == 200) {
   $entitlements = $response['data']->account->entitlements;
   foreach ($entitlements as $ent) {
      if ($ent->merchantEntitlementId == 'bac') {
         if ($ent->endTimeStamp < $now) {
              // yes, properly revoked
         }
         else {
              print "Failed to revoke 'bac' after $now\n";
         }
      }
   }
}
else {
   print "Failed to revoke 'bac'\n";
}
```

## stopAutoBilling

The `stopAutoBilling` method cancels one or more `AutoBill` objects (subscriptions) associated with this `Account` object. Rather than making separate `cancel` calls, cancel the `AutoBill` objects in a single call with this method.

### Input

***account:*** the `Account` object for which one or more `AutoBill` objects will be stopped. Use the `merchantAccountId` or `VID` to identify the object.

***autobills:*** an array of one or more `AutoBill` objects to cancel. If you do not specify this parameter, this method cancels all `AutoBill` objects associated with the `Account`.

***disentitle:*** a Boolean flag that specifies whether or not the customer is immediately denied further access to a product or service. Set ***disentitle*** to `true` to cancel the customer's subscription access immediately, and to `false` to allow the customer continued access until the currently paid subscription expires.

***force:*** *a Boolean flag that, if set to `true`, stops the `AutoBill` even if the subscription has not yet expired. (This parameter is a placeholder, and is not in use.)*

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***account:*** the `Account` object for which this method stopped one or more `AutoBills`.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Base Account not specified. |

### *Example*

```
// to stop all auto billing for an account

$customerID = '1234-5678-9000';

// Create an account object
$account = new Account();

// Set merchant account id in it so CashBox knows which account
// the autobills are to be cancelled
$account->setMerchantAccountId($customerID);

// To specify the autobills to cancel, construct AutoBill objects
$autobill1 = new AutoBill();
$autobill1->setMerchantAutoBillId('xyz-111');

$autobill2 = new AutoBill();
$autobill2->setMerchantAutoBillId('abc-222');

$autobillsToCancel = array($autobill1, $autobill2);
$immediateDisentitlement = true;

$response =
   $account->stopAutoBilling($autobillsToCancel,
   $immediateDisentitlement, false);

if($response['returnCode'] == 200) {
   print "Ok\n";
}
else if ($response['returnCode'] == 400) {
   print "Could not find account to cancel autobills for \n";
}
```

## tokenBalance

The `tokenBalance` method returns the balance of tokens of the specified type in the `Account` object. If you do not specify the token type, the call returns the balance of all the tokens currently available to the account.

### Input

***account:*** the `Account` object whose token balance you wish to return. Use the `merchantAccountId` or `VID` to identify the object.

***tokens:*** an array of one or more token types, whose balance you wish to return. If you do not specify a type, `tokenBalance` returns the balance for all the types available to the `Account` object.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***tokenAmounts:*** an array of one or more `TokenAmount` objects, each of which specifies the type of token, its quantity, and the balance of the tokens that are available to the `Account` object. If you do not specify a token type in the input, this array contains the balance of all token types available to the `Account`. Otherwise, this array contains the balances of only the specified token types.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$acct = new Account();

// Reference an existing account from which tokens are to be deducted
$acct = new Account();
$acct->setMerchantAccountId('9876-5432');

// make the SOAP call to retrieve tokens
$response = $acct->tokenBalances(null);

// return balances for all token types
if($response['returnCode']==200) {
   // the call returns new token balances on the account
   // print those out
   $tokBalances = $response['tokenAmounts'];
   foreach ($tokBalances as $tokBal) {
      print "Token type" . $tokBal->token->merchantTokenId . "\n";
      print "Token amount available" . $tokBal->amount . "\n";
   }
}
```

## tokenTransaction

The `tokenTransaction` method performs one or more token transactions, of multiple token types, on an `Account` object. The transactions may be positive, increasing the token balance; or negative, reducing the token balance.

Calling `tokenTransaction()` enables you to conduct a lightweight transaction with only tokens. Although Vindicia's internal token system tracks this type of transaction for audit logging, they are not a part of Vindicia's standard transaction framework for money-based transactions.

### Input

***account:*** the `Account` object for which to perform the transaction. Use the `merchantAccountId` or `VID` to identify the object.

***transactions:*** an array of one or more `TokenTransaction` objects to perform against the `Account` object. Each `TokenTransaction` object specifies the type of token and the quantity to increment or decrement from the object.

The following table lists and describes the data members of the `TokenTransaction` object.

Table 1-12 `TokenTransaction` Object Data Members

| Data Members | Data Type | Data Members |
|---|---|---|
| `authTimestamp` | `dateTime` | A timestamp that specifies the date and time of when you processed the transaction. Insert this data with your code. |
| `clearedTime-stamp` | `dateTime` | A timestamp that specifies the date and time of when Vindicia processed the transaction. CashBox inserts this data. |
| `description` | string | **Optional.** A memo for the transaction. |
| `tokenAmount` | `TokenAmount` | **Required**. An enumerated string value that categorizes the type of account activity you are recording. |

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***tokenAmounts:*** an array of `TokenAmount` objects, each of which contains the new balance and the token type available to the `Account` object after this call succeeds.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `401` | `Balance too low.`<br>Returned if one or more transactions requested would drop the user's balance below 0. |
| `404` | `Token not found.`<br>Returned if one or more tokens specified are not a saved type; however, the tokens available on the account are still returned, |

## Example

```
$tokTxn1 = new TokenTransaction();

// Reference an existing account to which this transaction is to be
// applied
$acct = new Account();
$acct->setMerchantAccountId('9876-5432');

$tokTxn1->setAccount($acct);

// Specify information about the tokens for this transaction
$tok1 = new Token();
$tok1->setMerchantTokenId("US_FREQ_BOOK_BUYER_PT");
$tokAmt1 = new TokenAmount();
$tokAmt1->setToken($tok1);
$tokAmt1->setAmount(4); // Number of tokens spent with this transaction

$tokTxn1->setTokenAmount($tokAmt1);

$tokTxn1->setDescription("Purchase: Stranger in a Strange Land");

$tokTxn2 = new TokenTransaction();

$tokTxn2->setAccount($acct);
// Information about the tokens that will pay for the transaction
$tok2 = new Token();
$tok2->setMerchantTokenId("US_FREQ_BOOK_BUYER_PT");
$tokAmt2 = new TokenAmount();
$tokAmt2->setToken($tok2);
$tokAmt2->setAmount(3); // Number of tokens for the transaction

$tokTxn2->setTokenAmount($tokAmt2);

$tokTxn2->setDescription("Purchase: Infinite Jest");

$tokTxns = array($tokTxn1, $tokTxn2);

// Make the SOAP call to perform the token transactions
// Ensure that account set in each TokenTransaction object is
// the same Account object on which you make the following SOAP call

$response = $acct->tokenTransaction($tokTxns);

if($response['returnCode']==200) {
    // print the new token balances on the account
    $newTokBalances = $response['tokenAmounts'];

    print "New token balances for account with id "
           . $acct>merchantAccountId . "\n";
    foreach ($newTokBalances as $newTokBal) {
        print "Token type"
            . $newTokenBal->token->merchantTokenId; . "\n";
    print "Token amount available" . $newTokenBal->amount; . "\n";

    }

}
```

# transfer

Customers often create multiple accounts on merchant sites. Because these accounts are essentially duplicates, you might receive a request from a customer to consolidate the billing for two accounts that customer has with you. Use the transfer method to consolidate billing for two accounts held by a single customer.

The `transfer` call merges the contents and related objects of two `Account` objects, and returns the target account with the merged content.

The `transfer` method:

- Transfers the payment methods, chargebacks, tax exemptions, `AutoBill` objects, activities, token grants and deductions, transactions, and name–value pairs associated with the source account into the target account.
- Strips all of the above contents from the source account, which will, however, continue to exist in CashBox with some basic attributes.
- Creates two name–value pairs with the names `VIN_MERCHANT_CUSTOMER_ID_UPDATED_FROM` and `VIN_MERCHANT_CUSTOMER_ID_UPDATED_DATE` in each of the `AutoBill` objects transferred, thus specifying the original `merchantAccountId` value associated with the `AutoBill` object and the date on which the ID was transferred.

The two `Accounts` specified as input must exist prior to the `transfer` call, or an error will be returned.

## Input

***targetAccount:*** the `Account` object into which you wish to transfer all content from the source account.

***sourceAccount:*** the `Account` object whose content you wish to transfer to the target account.

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***mergedAccount:*** the account that contains the merged content of the target and source accounts.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `No match found for target account.`<br>• `No match found for source account.`<br>• `Unable to transfer account.` |

### *Example*

```
$targetAcct = new Account();

// Reference an existing account into which the contents will be
// merged

$targetAcct->setMerchantAccountId('9876-5432');

$sourceAcct = new Account();

// Reference an existing account from which we want to transfer
// contents

$sourceAcct->setMerchantAccountId('4932-5301');

// make the SOAP call to retrieve tokens
$response = $targetAcct->transfer($sourceAcct);

if($response['returnCode']==200) {
   $mergedAcct = $response['mergedAccount'];
   // process or verify contents of the merged account here
}
```

## transferCredit

The `transferCredit` method transfers credits from a parent `Account` to a child `Account`, or from one child in a family to another.

### Input

***fromAccount:*** the `Account` from which credits will be transferred.

***toAccount:*** the child account to which credits will be transferred.

***credit:*** the credits to be transferred.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `No match found for toAccount.`<br>• `No match found for fromAccount.` |
| `500` | `Unable to transfer account.` |

## Example

```
// to transfer credits from a parent to a child account
// Create a new Account object for parent
$parent = new Account();

// Provide basic account information
$parent->setName('Somebody Q. Customer'); // Customer name
$parent->setMerchantAccountId('IN9430-8421'); // Unique customer id

// Create a new Account object for child
$child = new Account();
$child->setName('Somebody Q. Customer Jr.'); // Customer name
$child->setMerchantAccountId('IN9430-8421JR'); // Unique customer id

// Establish a parent->child relationship between $parent and $child
$childrenAdded = $anyOldAccountYouveGot->addChildren
          ($parent, array($child))

//Grant credit to the parent
$curAmt = new CurrencyAmount ;
$curAmt->setCurrency('USD');
$curAmt->setAmount(100.00);

$cr = new Credit();
$cr->setCurrencyAmounts(array($curAmt));
// Now make the SOAP API call to grant credit to the acct
$response = $acct->grantCredit($cr);
if ($response['returnCode'] == 200) {
   // Credit successfully granted to the account
   $updatedAcct = $response->['account'];
}
else {
   // Error while granting credit to the account
   print $response['returnString'] . "\n";
}

//Define credits to be transferred from parent to child
$curTranAmt = new CurrencyAmount ;
$curTranAmt->setCurrency('USD');
$curTranAmt->setAmount(12.34);

$crTran = new Credit();
$crTran->setCurrencyAmounts(array($curTranAmt));

//Transfer specified credits from parent to child account
$response = $parent->transferCredit($child, $crTran);
if ($response['returnCode'] == 200) {
   // Credit successfully granted to the account
}
else {
   // Error while transferring credit between accounts
   print $response['returnString'] . "\n";
}
```

## update

The update method creates an Account object, or updates an existing one. Use the update method when a new customer record is created.

To create an Account object, initialize it and set the values for its data members as appropriate, and then call the update method to store the changes. When creating a new Account object, do *not* set a value for VID; CashBox will automatically generate a VID for the object when you call update. When updating an existing Account object, identify it with its VID or your account ID (merchantAccountId).

| **Note** | Do not call update() to add or change a payment method for Account. Call Account.updatePaymentMethod() instead. (See the updatePaymentMethod method.) |
|---|---|

### *Input*

***account:*** the Account object to create or update. Use the merchantAccountId or VID to identify the object.

### *Output*

***return:*** an object of type Return that indicates the success or failure of the call.

***account:*** the Account object you created or updated.

***created:*** a Boolean flag that, if set to true, indicates that update has created a new Account object. A false setting means that update has updated an existing Account object.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• Data validation error. Failed to create Payment-Type-Specific Payment Record: Credit Card conversion failed: Credit Card failed Luhn check.<br>• Failed to save account. |

## *Example*

```
// Create a new Account object
$account = new Account();

// Provide basic account information
$account->setName('Somebody Q. Customer'); // Customer name
$account->setMerchantAccountId('IN9430-8421'); // Unique customer id

// To create address information, create an address object
$address = new Address();
$address->setAddr1('123 Main Street');
$address->setAddr2('Apt. 4');
$address->setCity('San Carlos');
$address->setDistrict('CA');
$address->setPostalCode('94070');
$address->setCountry('US');
$address->setPhone('123-456-7890');

// Associate the Address object with the account

$account->setShippingAddress($address);

// Emails
$account->setEmailAddress('John.Doe@gmail.com');
$account->setEmailTypePreference('html');
$account->setWarnBeforeAutoBilling(true);

// Okay, basic information is entered, so save the account
$response = $account->update();

// Check to see that the account was created
if($response['returnCode'] == 200) {
   // You can save the VID (Vindicia ID) for later use
   $accountVid = $account->getVid();
}
```

## updatePaymentMethod

The `updatePaymentMethod` method updates the `Account` object with the information for a payment method, such as a credit card that is on record. For example, call this method to change a credit card's expiration date. This method is especially useful if the `Account` object has associated active `AutoBill` objects and you would like to replace their payment methods with another one to apply to the next billing.

Call this method to catch up on the billing of an `AutoBill` object that has stalled due to a failed billing. For example, when your customers receive an email notification about a hard failure of a subscription (`AutoBill`), they are usually directed to your site to take remedial action (for example, to update the payment method), which, in turn, should invoke this method to send the updated payment method information to Vindicia.

If both ***ignoreAvsPolicy*** and ***ignoreCvnPolicy*** are `true`, no policy evaluation will be done. If only one of those flags is set to `true`, policy evaluation will not be considered for that element (AVS or CVN). If no value is passed in for either parameter, they will default to `false`, and the AVS and CVN policy evaluations will be used to determine `PaymentMethod` validation status.

For more detail on AVS and CVN Return Codes, please work with your Vindicia Client Services representative.

### *Input*

***account:*** the `Account` object whose payment method you would like to change. Use the `merchantAccountId` or `VID` to identify the object.

***paymentMethod:*** the required `PaymentMethod` object that contains the new data to apply to the `Account` object's payment method. (For more information, see Section 11: The PaymentMethod Object.)

If you specify a VID or `merchantPaymentMethodId` to identify ***paymentMethod***, this method updates the payment method in question. If you do not specify a `VID` or `merchantPaymentMethodId`, this method creates a new payment method, and attaches it to the `Account` object.

If you specify an existing sort order for the payment method (for example, 0, which is the default), `updatePaymentMethod` pushes down the payment method with the same sort order to the next increment (for example, 1), and increments the sort order of the subsequent payment methods accordingly.

***replaceOnAllAutoBills:*** a Boolean flag that, if set to `true`, replaces the payment method on all the `AutoBill` objects for this `Account` object. If you set the flag to `false`, `updatePaymentMethod` attaches the payment method to the `Account` object and saves it.

***updateBehavior:*** specifies whether to just update (`Update`) without validation, validate first (`Validate`), or catch up with billing first (`CatchUp`).

If you set the value to `CatchUp`, the call first finds the latest `AutoBill` object associated with the `Account` object. Depending on whether that `AutoBill` object's end date has passed, the call proceeds:

- If the latest `AutoBill` object is in `Hard Error` state, and its end date is in the future (that is, the retry period for that object's failed billing transaction is not yet over), the call reprocesses the failed transaction that caused the `AutoBill` object to enter `Hard Error` state with the newly input payment method. If successful, the call then:

  1. Updates the `AutoBill` object's payment method for future billings. If you have enabled the ***replaceOnAllAutoBills*** flag, then the call also updates the payment method for all other `AutoBill` objects associated with the `Account` object, but does *not* reactivate them or reprocess their failed billings if any of them are in `Hard Error` state.
  2. Reactivates the latest `AutoBill` object so that it is in `Good Standing` state, resulting in normal scheduling of future billings.
  3. Returns `200` as the SOAP call's return code, indicating success.

- If the latest `AutoBill` object is in `Hard Error` state, but the object's end date has passed (that is, the retry period is already over), the call validates the payment method and, if successful, updates the input payment method on the latest `AutoBill` object. If you have enabled the ***replaceOnAllAutoBills*** flag, then the call also updates the other `AutoBill` objects, if any, that are associated with the `Account` object. In this case, the call does *not* reactivate any `AutoBill` objects in `Hard Error` state that are associated with the `Account` object or process their failed billings. However, the call still returns the SOAP return code `200`.

  After calling `updatePaymentMethod`, always check the relevant `AutoBill` object's status by fetching it with one of the fetch methods, such as `fetchByAccount()`, to verify if the `AutoBill` object has been reactivated. If not, create a new `AutoBill` object with the new payment method and start a new subscription for the customer.

***ignoreAvsPolicy:*** a Boolean flag that, if set to `true`, will override the AVS policy, and update the `paymentMethod`, regardless of the AVS return code. If set to `false` or `null`, (and if ***validatePaymentMethod*** is set to `true`) the AVS return code will be used to determine whether to update the `paymentMethod`.

***ignoreCvnPolicy:*** an optional Boolean flag that, if set to `true` will override the CVN policy, and update the `paymentMethod`, regardless of the CVN return code. If set to `false` or `null`, (and if ***validatePaymentMethod*** is set to `true`) the CVN return code will be used to determine whether to update the `paymentMethod`.

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***account:*** the `Account` object whose payment method was changed.

***validated:*** a Boolean flag that, if set to `true`, indicates that this method has successfully validated the `PaymentMethod` object.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 261 | `All active AutoBills were updated. AutoBills which are both expired and Suspended cannot be updated.` |
| 400 | One of the following:<br>• `Invalid Payment Method Type. (You cannot change the Payment Method Type on an existing Payment Method.)`<br>• `No PaymentMethod specified in arguments.`<br>• `Data validation error Failed to create Payment-Type-Specific Payment Record: Credit Card conversion failed: Credit Card failed Luhn check.` |
| 402 | One of the following:<br>• `PaymentMethod failed validation.`<br>• `Error attempting to authorize card.`<br>• `Unable to authorize card.` |
| 404 | `No match found` ***error-description.***<br>Returned if CashBox cannot find an account that matches the input in the Vindicia database. |
| 407 | `AVS policy evaluation failed.` |
| 408 | `CVN policy evaluation failed.` |
| 409 | `AVS and CVN policy evaluations failed.` |
| 410 | `AVS and CVN policy evaluations could not be performed.` |

### *Example*

```
// to update a payment method

$accountId = "CUST219";

// Create an account object
$account = new Account();

$account->setMerchantAccountId($accountId);

$paymentMethod = new PaymentMethod();

// update an existing payment method to a new expiration
// date and a new billing address. The code here assumes that you know
// the merchantPaymentMethodId of the payment method

$paymentMethod->setMerchantPaymentMethodId("345abc678");
$newBillAddress = new Address();
$newBillAddress->setAddr1("123 Maple St");

// Populate rest of the address object here

// Set the new billing address in the payment method
$paymentMethod->setBillingAddress($newBillAddress);

// Create a new credit card object and populate it with updated
// information
$cc = new CreditCard();
$cc->setAccount("4343121267679193");
$cc->setExpirationDate("201211");

// Set the credit card information in the payment method
$paymentMethod->setType('CreditCard');
$paymentMethod->setCreditCard($cc);

// Now make the updatePaymentMethod call with validation and
// replacement on all autobills enabled

$replaceOnAutoBills = true;
$response =
   $account->updatePaymentMethod($paymentMethod, $replaceOnAutoBills,
"Validate", 0);

if($response['returnCode'] == 200) {
   print "Call succeeded\n";
}
else if($response['returnCode'] == 402) {
   print "Payment method validation failed\n";
}
else {
   // Handle other error situations here
}
```

# 2 The `Activity` Object

The `Activity` object enables you to record activities (events) on your site that are not direct purchase transactions, such as when customers access for-pay content like song downloads. That information can serve as evidence for chargeback disputes should they occur.

You make calls available for the `Activity` object to submit the activity data once per event. Alternatively, queue and submit the data periodically in a batch process. Usually, you collect events of interest only. For example, you need not record every page view by a customer, only those page views that contain for-pay content that the customer accessed or downloaded.

To use the `Activity` object in your application, first create the `Activity` object, then populate its data members with the appropriate information, and submit the event to Vindicia with the `record()` method.

## 2.1 `Activity` Data Members

To record an activity, fill in as many of the data-member fields of the `Activity` object as possible. The more information you collect, the more useful it will be for Vindicia to dispute chargebacks on your behalf should they occur.

The following table lists and describes the data members of the `Activity` object.

Table 2-1 `Activity` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `account` | `Account` | **Required**. The customer account for which you are recording this activity. This information serves as evidence of the customer's connection to the activity. Populating this object with either the VID or `merchantAccountId` suffices.<br>See Section 1.2: Account Data Members. |
| `activityArgs` | `ActivityTypeArg` | **Required.** An object that details the activity you are recording. The content varies, depending on the activity type specified.<br>See the `ActivityTypeArg` Subobject. |
| `activityType` | `ActivityType` | **Required**. An enumerated string value that categorizes the type of activity you are recording. For example, if a customer calls you, set this value to `Phone`. Be sure to set this value before calling `record()`.<br>See the `ActivityType` Subobject. |
| `timestamp` | `dateTime` | **Required**. A timestamp that specifies the date and time of when the event you are recording took place. Be sure to set this value before calling `record()`. |

## 2.2 `Activity` Subobjects

The `Activity` object has several subobjects:

- `ActivityCallType` Subobject
- `ActivityCancelInitType` Subobject
- `ActivityCancellation` Subobject
- `ActivityEmailContact` Subobject
- `ActivityFulfillment` Subobject
- `ActivityLogin` Subobject
- `ActivityLogout` Subobject
- `ActivityNamedValue` Subobject
- `ActivityNote` Subobject
- `ActivityPhoneContact` Subobject
- `ActivityType` Subobject
- `ActivityTypeArg` Subobject
- `ActivityURIView` Subobject
- `ActivityUsage` Subobject

### `ActivityCallType` Subobject

Supplies the type of phone contact made.

Table 2-2 `ActivityCallType` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `fromCustomer-ToMerchant` | string | The customer called you or your agent, for example, Technical Support. |
| `fromCustomer-ToOther` | string | The customer called someone other than you. |
| `fromMerchant-ToCustomer` | string | You called the customer. |
| `fromMerchant-ToOther` | string | You called someone other than the customer. |
| `fromOtherTo-Customer` | string | Someone other than you called the customer. |
| `fromOtherTo-Merchant` | string | Someone other than the customer called you. |

## ActivityCancelInitType Subobject

A list of known types if initiators for cancellation activities.

Table 2-3 ActivityCancelInitType Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| Chargeback | string | The service was cancelled due to a chargeback. |
| Customer | string | The customer initiated the cancellation. |
| Merchant | string | You initiated the cancellation. |

## ActivityCancellation Subobject

Supplies information about a customer cancellation.

Table 2-4 ActivityCancellation Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| confirmation-Code | int | The confirmation code for the cancellation. |
| initiator | ActivityCan-celInitType | The type of initiator for cancellation.<br>See the ActivityCancelInitType Subobject. |
| reason | string | The reason for cancellation. |

## ActivityEmailContact Subobject

Supplies information about an email contact with a customer.

Table 2-5 ActivityEmailContact Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| destEmail | string | The recipient's email address. |
| note | string | A note on the content of the email message. |
| srcEmail | string | The sender's email address. |

## `ActivityFulfillment` Subobject

Supplies information about physical fulfillment of an order.

Table 2-6 `ActivityFulfillment` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `delivered` | Boolean | A Boolean flag that, if set to `true`, indicates that the merchandise delivery is complete. |
| `merchantTransactionId` | string | Your unique identifier for the transaction. |
| `receiptName` | string | The recipient's name as reported by the shipping agent. |
| `receivedTs` | `dateTime` | A timestamp that corresponds to the date and time reported by the shipping agent of when the merchandise delivery was completed. |
| `shipper` | string | The identifier of the shipping agent (such as UPS or FedEx) if any. |
| `shippingAddress` | `Address` | The shipping address for the product. This data member encapsulates the customer's mailing address, billing address, or both.<br>See Section 3.1: Address Data Members. |
| `trackingString` | string | The tracking information on the physical package. |

## `ActivityLogin` Subobject

Supplies information about an `Account` login.

Table 2-7 `ActivityLogin` Object Data Member

| Data Members | Data Type | Description |
|---|---|---|
| `ip` | string | The IP address from which a login originated. |

## `ActivityLogout` Subobject

Supplies information about an `Account` logout.

Table 2-8 `ActivityLogout` Object Data Member

| Data Members | Data Type | Description |
|---|---|---|
| `ip` | string | The IP address from which a logout originated. Set `ip` to null if the logout is implicit due to, for example, a server timeout. |

## ActivityNamedValue Subobject

A generic activity type. This object should not be used permanently; it provides a temporary means to bridge to new activities without a SOAP release. Contact Vindicia before submitting data of this type.

Table 2-9 ActivityNamedValue Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| **Note: You must enter a value for all three data members.** | | |
| name | string | The Activity name. For example, if you sell music online, set the value to musicDownload. |
| type | string | The Activity type. For example, if you sell different types of music online, specify in this field the type, such as Rock. |
| value | string | The Activity value. For example, fill in this field with the name of the artist or song for the download. |

## ActivityNote Subobject

Supplies a note or memo.

Table 2-10 ActivityNote Object Data Member

| Data Members | Data Type | Description |
|---|---|---|
| note | string | Notes (maximum of 1,024 characters) on the Activity object. |

## `ActivityPhoneContact` Subobject

Supplies information about a phone contact with a customer.

Table 2-11 `ActivityPhoneContact` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `aniPhoneNumber` | string | The Automatic Number Identification (ANI) for the phone number from which the call originated. |
| `cidPhoneNumber` | string | The caller ID (CID) for the phone number from which the call originated. |
| `destPhoneNumber` | string | The phone number of the person who received the call. |
| `durationSeconds` | int | The length of the phone conversation in seconds. |
| `note` | string | **Optional.** Notes on the phone call. |
| `srcPhoneNumber` | string | The phone number from which the call originated. |
| `type` | `ActivityCallType` | **Required.** An enumerated value that specifies who originated and who received the call.<br>See the `ActivityCallType` Subobject. |

## `ActivityType` Subobject

Describes a list of known types of Activities.

Table 2-12 `ActivityType` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `Cancellation` | string | A cancellation of a product or service offered by you. |
| `Email` | string | An email interaction related to the account. |
| `Fulfillment` | string | An order fulfillment. |
| `Login` | string | A customer login on your site. |
| `Logout` | string | A customer logout from your site. |
| `NamedValue` | string | An activity that differs from the predefined activities specified by other `ActivityType` values. Setting this value means that you are defining a custom activity type for your product or service. |
| `Note` | string | An optional memo regarding the activity. |
| `Phone` | string | A phone interaction related to the account. |
| `URIView` | string | A viewing of a particular Web resource. |
| `Usage` | string | The amount of use of the resources provided by you, such as electronic downloads, or website access. |

## ActivityTypeArg Subobject

A "master class" for activity subclasses. While WSDL does not appear to allow for the definition of literal subclasses, this provides similar results. methodLink=report takes an argument of this class. Simply fill only the field necessary for the type of activity being recorded. Note that some activities may not require additional information. For example, if submitting a uriView, set uriviewArgs to a previously filled ActivityURIView.

Table 2-13 ActivityTypeArg Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| cancellationArgs | ActivityCancellation | The customer's cancellation of a service or product.<br>See the ActivityCancellation Subobject. |
| emailArgs | ActivityEmailContact | An email event.<br>See the ActivityEmailContact Subobject. |
| fulfillmentArgs | ActivityFulfillment | The status of your fulfillment of a customer order.<br>See the ActivityFulfillment Subobject. |
| loginArgs | ActivityLogin | The IP address from which a login originated.<br>See the ActivityLogin Subobject. |
| logoutArgs | ActivityLogout | The IP address from which a logout originated.<br>See the ActivityLogout Subobject. |
| namedValueArgs | ActivityNamedValue | An activity defined by you. With this data structure, you create Activity objects that are unique to your business, and that are not described by the predefined Activity events in ActivityTypeArg. Creating such an activity implies that it will likely occur regularly with your customers.<br>See the ActivityNamedValue Subobject. |
| noteArgs | ActivityNote | An optional memo on the Activity object.<br>See the ActivityNote Subobject. |
| phoneArgs | ActivityPhoneContact | A phone contact that relates to the Activity object.<br>See the ActivityPhoneContact Subobject. |
| uriviewArgs | ActivityURIView | A customer's visit to a Web page, and possible download activity.<br>See the ActivityURIView Subobject. |
| usageArgs | ActivityUsage | The amount of use of a resource (such as the number of downloads) you provide to the customer.<br>See the ActivityUsage Subobject. |

When constructing an Activity object, fill in the ActivityTypeArg object with a subobject, as appropriate, for activityType. For example:

- If activityType is phone, create an ActivityTypeArg object and fill phoneArgs with data in the form of an ActivityPhoneContact data structure.
- If activityType is email, create an ActivityTypeArg object and fill emailArgs with data in the form of an ActivityEmailContact data structure.

## ActivityURIView Subobject

Supplies information about a user viewing a document.

Table 2-14 ActivityURIView Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| bytesTransferred | int | The number of bytes actually transferred to the customer. |
| description | string | A description of the bytes transferred. |
| ip | string | The IP address to which the data was transferred. |
| size | int | The size of the download. |
| transferTime | int | The length of the data transfer in seconds. |
| uri | string | **Required**. The URI of the page. |

## ActivityUsage Subobject

Supplies information on the use of a service by a customer.

**Note:** Please convert all durations to seconds.

Table 2-15 ActivityUsage Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| description | string | The amount of use. |
| lastDay | int | The amount of use on the last day. |
| lastMonth | int | The amount of use in the last month. |
| lastUsageDate | dateTime | The last date of use. |
| lastWeek | int | The amount of use in the last week. |
| lastYear | int | The amount of use in the last year. |
| total | int | The duration of use. |

## 2.3 `Activity` Method

The method for `Activity` is `record()`, which posts one or more `Activity` objects to the CashBox database.

## record

To report one or more non-transaction activities to Vindicia, create an `Activity` object for each activity, insert the object into an array, and pass the array as an argument to `record()`. Every `Activity` object requires that you specify the associated `Account` object. For that purpose, you can create an `Account` object and populate it with only its VID or `merchantAccountId`.

### *Input*

***activities:*** an array of `Activity` objects to report to Vindicia.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unknown activity type` ***input-type.*** `Must be one of` ***list-of-allowed-types.***<br>• `Required field 'timestamp' missing!`<br>• `Required field 'account' missing!` |

### *Example*

```
// To report a phone call as an Activity, create an account object
$account = new Account();

// Specify account by the customer id
$account->setMerchantAccountId('9876-5432');

// Create Activity to report customer's phone call
// and corresponding ActivityTypeArgs objects

$activity = new Activity();
$typeArgs = new ActivityTypeArgs();

// fill in the relevant info for this activity record
$activity->setAccount($account); //associate the activity and account
$activity->setActivityType('Phone');
$activity->setTimestamp(getdate());

$phoneArgs = new ActivityPhoneContact();
$phoneArgs->setCidPhoneNumber('1234567890');
$phoneArgs->setDurationSeconds(367)
$phoneArgs->setType('FromCustomerToMerchant');
$phoneArgs=>setNote('Customer agreed to be rebilled for services');

$typeArgs->setPhoneArgs($phoneArgs);

// associate typeArgs to the Activity object
$activity->setActivityArgs($typeArgs);

// now record the data
$response = $activity->record(array($activity));

if($response['returnCode'] == 200) {
   print "ok\n"; # 200 is HTTP status code for success
}
```

# 3 The `Address` Object

The `Address` object encapsulates the contact information for a customer, including the full name, postal address, and fax and phone numbers. Save a customer's billing and shipping addresses with the `Address` object. For example, the `Account` object includes the `shippingAddress` data member, which in turn contains an `Address` object.

# 3.1 Address Data Members

The following table lists and describes the data members of the Address object.

Table 3-1 Address Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| addr1 | string | The first address line. |
| addr2 | string | The second, auxiliary address line. |
| addr3 | string | The third, auxiliary address line. |
| city | string | The city of the customer's address. |
| country | string | Specifies the geographical region for the customer's address. country is the ISO-3166-1 two-letter code for the country (for example, US, GB, or FR), for which CashBox computes sales tax. |
| county | string | The county of the customer's address if known. |
| district | string | The state, province, or district of the customer's address. |
| fax | string | The customer's fax number. |
| latitude | decimal | The customer's latitude as a signed decimal. In some cases, Vindicia fills in this field. |
| longitude | decimal | The customer's longitude as a signed decimal. In some cases, Vindicia fills in this field. |
| name | string | The customer's full name. (256 character limit.) |
| phone | string | The customer's preferred phone number. |
| postalCode | string | The postal code of the customer's address.<br>**Note:** Your payment processor may limit this field to 9 characters. |
| VID | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new Address object, leave this field blank; it will be automatically populated by CashBox. |

When creating a new Address object, do not specify a VID when you call the update() method for Address. CashBox will generate a VID, and return it in the resultant Address object.

To link an Address object to an Account object, call the Account.setShippingAddress() method. You can also construct an Address object without an explicit update() call. For example, if you create an Account object with its update() method, and specify the shippingAddress attribute without specifying a VID, the call will automatically create a new Address object.

## 3.2 `Address` Methods

The following table summarizes the methods for the `Address` object.

Table 3-2 `Address` Object Methods

| Method | Description |
|---|---|
| `fetchByVid` | Returns an `Address` object whose VID matches the input. |
| `update` | Creates or updates an `Address` object. |

## fetchByVid

The `fetchByVid` method returns an `Address` object whose VID matches the input. To update a stored customer address, first load it into your application with this method. The VID you specify as an argument is usually the one you obtain from an `Account` object.

### Input

***vid:*** the `Address` object's Vindicia identifier, which serves as the search criterion.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***address:*** the `Address` object whose VID matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `No addresses match VID` ***input-vid.***<br>• `Unable to load VID` ***input-vid: error-description.***<br>• `Missing required parameter vid.` |

### Example

```
$accountVid = 'MyVindiciaAccountVID';

// Create a SOAP caller object
$addr = new Address();
$addrVID = "14e1dce6f48e901464fce22145982a59642aa9f4";

// now load an address object by VID
$response = $addr->fetchByVid($addrVID);
if($response['returnCode'] == 200) {
    $fetchedAddr = $response['data']->address;
}
else {
    // The call was unsuccessful
    print "Return code: " . $response['returnCode'] . "\n";
    print "Return string: " . $response['returnString'] . "\n";
}
```

## update

The `update` method creates or updates an `Address` object. When creating a new `Address` object, do not set a value for `VID`; CashBox will automatically generate a VID for the new object when you call `update()`. When updating an existing `Address` object, identify it with its VID.

| **Note** | You can also create an `Address` object indirectly by specifying it inside other objects that you explicitly create. For example, specify `shippingAddress` when you create `Account`; specify `billingAddress` when you create `PaymentMethod`, and etc. |
|---|---|

### *Input*

***address:*** the populated `Address` object to create or update. To update an existing object, be sure to specify its VID.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***address:*** the `Address` object that was created or updated.

***created:*** a Boolean flag that, if set to `true`, indicates that this method has created a new `Address` object. A false setting indicates that `update` has updated an existing `Address` object.

### *Returns*

This call returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
// To create address information, instantiate an Address object
$address = new Address();

// populate the address object with data
$address->setAddr1('123 Main Street');
$address->setAddr2('Apt. 4');
$address->setCity('San Carlos');
$address->setDistrict('CA'); // this is US state or province
$address->setPostalCode('94070');
$address->setCountry('US');
$address->setPhone('123-456-7890');

$response = $address->update();
if($response['returnCode'] == 200) {
   $createdAddr = $response['data']->address;
   print "Address create with VID " . $createdAddr->getVID() . "\n";
}
```

# 4 The AutoBill Object

The AutoBill object defines the relationship between an Account object (the customer description), an AutoBillItem object (the product(s) or service(s) purchased), and a BillingPlan object (the frequency and amount of the bill). An AutoBill describes the purchase by encapsulating the data members and methods that control the purchase terms, frequency, and rates for recurring billing, and any additional subscription information.

AutoBills usually encapsulate the terms of a recurring or renewable subscription. Although you may use AutoBill for one-time purchases (typically when an entitlement system is required), they are best handled with the Transaction object instead. For details, see Section 18: The Transaction Object.

Once created, an AutoBill object automatically generates periodic Transaction objects within CashBox, according to the Billing Plan. CashBox processes those transactions with your payment processor. The status of a transaction determines the current status of the associated AutoBill object, which, in turn, affects the entitlements granted by AutoBill to the associated Account object. Be sure to define entitlements with either BillingPlan or Product (or both) when creating an AutoBill object.

The constituent objects of an AutoBill object, Account, AutoBillItem, and BillingPlan, may be preexisting objects, in which case, you can simply refer to them by their IDs when constructing AutoBill. You may also create these objects along with the AutoBill object by specifying them inside the AutoBill object with new IDs.

# 4.1 AutoBill Data Members

The following table lists and describes the data members of the AutoBill object.

Table 4-1 AutoBill Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| account | Account | **Required.** The Account object to which this AutoBill object applies. If you do not specify a valid VID or merchantAccountId, CashBox creates a new Account object. |
| billingDay | int | The day of the month on which to bill the customer, which, if unspecified, defaults to the day of the startTimestamp. The value ranges from 1 to 31. CashBox automatically handles calendaring anomalies. For example, if you set this value to 31 but the month in question contains only 30 days, recurring billing automatically adjusts to day 30 for that month.<br>This attribute is useful if AutoBill has a yearly or monthly billing plan, and if the customer desires to be billed on a specific day of the month. If the billing plan is in terms of a daily or weekly cycle, the next billing day is determined by the duration and length of the cycle.<br>**Note:** If the Billing Plan for the AutoBill includes a Season Set, and if the Billing Periods are set to repeat according to Seasons, this data member will be automatically reset by CashBox, according to the defined repetition cycle. |
| billingPlan | BillingPlan | The billing plan to be used for this AutoBill object. This attribute determines the frequency and amount of periodic billing transactions generated by this AutoBill object.<br>If you do not specify this attribute, CashBox uses the default billing plan associated with the primary Product object in this AutoBill object. If you have not defined a default billing plan for Product, be sure to specify it here.<br>If the BillingPlan object already exists, simply populate it with its VID or merchantBillingPlanId. If the BillingPlan does not yet exist, CashBox creates a new BillingPlan object along with this AutoBill object.<br>See Section 5.1: BillingPlan Data Members. |
| billingPlanCampaignCode | string | The Campaign code redeemed on this AutoBill against the price of the Billing Plan. To apply a Campaign, use this field to pass in a valid Coupon or Promotion code.<br>**Note:** This data member will not be returned. |
| billingPlanCampaignId | string | **Read only.** The unique identifier for a Campaign applied to this AutoBill's BillingPlan. This is a read-only field returned by CashBox for informational purposes. Values sent in with a SOAP call will be ignored. |
| billingPlanHistory | BillingPlanHistoryRecord | **Read Only.** An array of time periods during which BillingPlans were associated with the AutoBill. The endDate for the current Billing Plan will be blank (unless the AutoBill will not be billed again (for example: after a cancel() call).<br>See the BillingPlanHistoryRecord Subobject. |

Table 4-1 `AutoBill` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `billingStatementIdentifier` | string | The identifier on a customer's billing statement when the customer is charged for this `AutoBill` object.<br>If GlobalCollect, MeS, Chase Paymentech or Litle is your payment processor, see Appendix A: Custom Billing Statement Identifier Requirements in the ***CashBox Programming Guide*** for the rules for this string. |
| `credit` | `Credit` | This data member encapsulates credit available to the `AutoBill`.<br>Token-based credits stored in this attribute may be applied toward Transactions generated by this AutoBill for Billing Plans which are defined with a Payment Method of the same Token Type.<br>Currency-based credits must be of the same Currency type listed in the Billing Plan associated with this AutoBill, to be used toward Transactions generated by the AutoBill.<br>Time-based credits are stored in this attribute only until the next Billing Period, at which point they are immediately and fully applied toward the AutoBill.<br>Do not manipulate this attribute directly. Instead, use methods such as `grantCredit` or `revokeCredit` to manipulate the amount of credit available to the `AutoBill` object.<br>See the `Credit` Subobject. |
| `currency` | string | The ISO 4217 currency code (see www.xe.com/iso4217.htm) for this `AutoBill` object. The default is USD. |
| `customerAutoBillName` | string | **Optional.** A name you specify (on your customer's behalf) for this `AutoBill` object, such as 'Home Subscription.' |
| `endTimestamp` | dateTime | This is a read-only attribute in fetched `AutoBill` objects.<br>CashBox will automatically set this `timeStamp` based on the AutoBill's last successful billing date, plus the length of the next Billing Period, plus any grace period you may have defined. This value is reset with every successful billing.<br>**Note:** Do not set this value when creating or updating an `AutoBill`. |
| `invoiceTerms` | int | The number of days after the invoice date that a bill is considered delinquent, if the `AutoBill` payment method is `MerchantAcceptedPayment`. This value will be ignored for all other `AutoBill` payment methods. |
| `items` | `AutoBillItem` | An array of `AutoBillItems` to be included in the `AutoBill`.<br>See the `AutoBillItem` Subobject. |
| `merchantAffiliateId` | string | Your ID (a free-form string of 128 characters or less) for the affiliate that submitted this `AutoBill` object, if any. |
| `merchantAffiliateSubId` | string | Your ID (a free-form string of 128 characters or less) for the subaffiliate that submitted this `AutoBill` object. This ID enables more detailed tracking of affiliate programs, such as promotional campaigns. |
| `merchantAutoBillId` | string | Your unique identifier for this `AutoBill` object. |

Table 4-1 `AutoBill` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `minimumCommitment` | int | The number of billing cycles the customer is contractually obligated to complete before cancelling the subscription. For example, if you offer special pricing for a certain number of automatic billing renewals, you can track a customer's initial agreement to those terms with this data member.<br>When you make a call to cancel an `AutoBill` object, CashBox checks this attribute. If `AutoBill` has not completed the minimum commitment period, CashBox performs the cancellation only if the force parameter in the `cancel()` call is set to `true`. |
| `nameValues` | `NameValuePair` | An (optional) array of name–value pair items for this `AutoBill` object. Some names are reserved for specific purposes.<br>Use `vin:Division` to route this `AutoBill's` transactions to your payment processor as part of a business division, unit, or group you have registered with the processor.<br>CashBox provides four name-value pairs for use with European Direct Debit (EDD) payment methods:<br>Use name `vin:MandateFlag` and value `1` to associate the EDD Payment Method with the AutoBill.<br>Use name `vin:MandateVersion` and value `1.0.1`, to associate a mandate document of version 1.0.1 with the object.<br>Use name `vin:MandateID` to pass the Mandate ID field of the EDD Extension record to Chase Paymentech.<br>Use name `vin:MandateApprovalDate` to pass the Signature Date field of the EDD Extension Record to Chase Paymentech.<br>**Note:** All name-value pairs included with an AutoBill object will be automatically copied to any resultant Transactions.<br>See Section 10: The NameValuePair Object. |
| `nextBilling` | `Transaction` | An object of type `Transaction` that represents the next projected billing for this AutoBill, if any.<br>See Section 18.1: Transaction Data Members. |
| `paymentMethod` | `PaymentMethod` | Vindicia's identifier (VID) for the `PaymentMethod` object for this `AutoBill`. If you do not specify an existing VID or `merchantPaymentMethodId`, CashBox creates a new `PaymentMethod` object with this `AutoBill` object, and adds it to this `AutoBill` object's account.<br>If you do not specify this attribute, the `AutoBill` will automatically use the preferred `PaymentMethod` object associated with the `Account`.<br>See Section 11.1: PaymentMethod Data Members. |
| `sourceIp` | string | The IP address of the machine from which the customer requested the creation of this `AutoBill` object. This attribute is required if you wish to score a transaction associated with the `AutoBill` for risk screening. Some payment methods, such as European Direct Debit, also require this attribute. |
| `startTimestamp` | dateTime | A timestamp that specifies the start date and time for this `AutoBill` object. If unspecified, the value defaults to today and the current time. |

Table 4-1 `AutoBill` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `statementFormat` | `StatementFormat` | Defines the billing format used to send statements to a customer. Defaults to `DoNotSend` if not specified.<br>Valid input:<br>• `DoNotSend`<br>• `Inline`<br>• `Attachment` |
| `statementOffset` | int | Days prior to billing that a statement will be sent. This value must be "`null`" or "`0`" if the AutoBill's `PaymentMethodType` is `MerchantAcceptedPayment`. For conventionally-funded AutoBills, this value must be less than the prebilling notification days (if specified). The value will be ignored if `statementFormat` is `DoNotSend`. |
| `statementTemplateId` | string | Your identifier for a pre-defined statement template. If this value is `null` (or does not match any pre-defined statement templates), the CashBox-default template will be used. |
| `status` | `AutoBillStatus` | This `AutoBill` object's current status. CashBox automatically sets the status of an `AutoBill` object, depending on the success or failure of the latest billing transaction. Therefore, **do not set this status with an API call to CashBox**.<br>See the `AutoBillStatus` Subobject. |
| `VID` | `string` | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `AutoBill` object, leave this field blank; it will be automatically populated by CashBox. |
| `warnOnExpiration` | `Boolean` | A flag that, if set to `true`, specifies that the customer be warned by email of an upcoming expiration of a trial period or subscription. The default is `false`. For more information, see the `expireWarningDays` attribute in the `BillingPlanPeriod` object. |

# 4.2 AutoBill Subobjects

The AutoBill object has three subobjects:

- AutoBillItem Subobject
- AutoBillItemModification Subobject
- AutoBillStatus Subobject
- BillingPlanHistoryRecord Subobject
- PaymentDetails Subobject

## AutoBillItem Subobject

The AutoBillItem object allows you to add multiple items to an AutoBill, and define the duration of their inclusion.

The following table lists and describes the data members of the AutoBillItem object.

Table 4-2 AutoBillItem Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| addedDate | dateTime | **Read Only.** Specifies the dateTime when the AutoBillItem was added to the AutoBill. |
| amount | decimal | The amount to bill. If non-null, this field will override the Product's price. Value cannot be negative.<br>This field is populated only if the you wish to override the default (product-based) price for the item. Otherwise, it is blank.<br>**Note:** AutoBillItems may have an amount, or a ratePlan, but not both. |
| campaignCode | string | The Campaign code redeemed on this AutoBillItem. To apply a Campaign, use this field to pass in a valid Coupon or Promotion code.<br>**Note:** This data member will not be returned. |
| campaignId | string | **Read only.** The unique identifier for a Campaign applied to this AutoBillItem. This is a read-only field returned by CashBox for informational purposes. Values sent in with a SOAP call will be ignored. |
| currency | string | The ISO 4217 currency code to be used for the override amount. This value will be ignored if amount is null. |
| cycles | int | The number of billing cycles this item will be active. If null, the item will remain active until explicitly removed. |
| cyclesRemaining | int | A read-only field indicating how many billing cycles remain for this item. |

Table 4-2 `AutoBillItem` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `event-Initializer` | `Event` | The initial number of Rated Units associated with this item, if it is rated, and if it is License-based.<br>See the `Event` Subobject. |
| `index` | int | The index number of the item in the `items` field of an `AutoBill`. (Should be unique in array. First item should have index 0.) |
| `merchantAuto-BillItemId` | `string` | Your unique identifier for this `AutoBillItem` object. |
| `product` | `Product` | The Product to be AutoBilled. When creating a new `AutoBillItem`, an existing `VID` or `SKU` must be specified or a new `Product` will be created. It is generally recommended that `Products` be created explicitly in advance, rather than implicitly.<br>See Section 13.1: Product Data Members. |
| `ratePlan` | `RatePlan` | The Rate Plan associated with this Item.<br>**Note:** `AutoBillItems` may have an `amount`, or a `ratePlan`, but not both.<br>See Section 14.1: RatePlan Data Members. |
| `removedDate` | `dateTime` | A read-only attribute indicating the time this item was removed. |
| `startDate` | string | Specifies when the AutoBill will begin billing for the `AutoBillItem`, and when the item's entitlements will become Active.<br>Valid input includes `null` (for which the `startDate` will be ***today***), yyyy-mm-dd, or a time interval, such as 3 days, 1 year, or 2 seasons. |
| `token` | `Token` | The token associated with `amount` (if this is a Token-based AutoBill). This value will be ignored if `amount` is `null`. |
| `transitioned-FromAutoBillI-temVid` | string | **Read Only.** The unique Vindicia identifier of the `Au-toBillItem` this item replaced as the result of an `AutoBill.modify` call. |
| `transitioned-FromMerchant-AutoBillItemId` | string | **Read Only.** Your identifier for the `AutoBillItem` this item replaced as the result of an `Auto-Bill.modify` call. |
| `transitioned-ToAutoBillI-temVid` | string | **Read Only.** The unique Vindicia identifier of the `Au-toBillItem` that replaced this item as a result of an `AutoBill.modify` call. |

Table 4-2 `AutoBillItem` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `transitioned-ToMerchantAu-toBillItemId` | string | **Read Only.** Your identifier for the `AutoBillItem` that replaced this item as a result of an `Auto-Bill.modify` call. |
| `VID` | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `AutoBillItem` object, leave this field blank; it will be automatically populat-ed by CashBox. |

## AutoBillItemModification Subobject

This object is used with the AutoBill.modify call, and lists the AutoBillItems to add to or remove from an AutoBill. This object must uniquely identify an AutoBillItem, and may include the Item's VID, index, merchantAutoBillItemId, or Product.

The following table lists and describes the data members of the AutoBillItemModification subobject.

Table 4-3 AutoBillItemModification Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| removeAutoBillItem | AutoBillItem | An AutoBillItem to remove from the AutoBill. |
| addAutoBillItem | AutoBillItem | An AutoBillItem to add to the AutoBill. |

## AutoBillStatus Subobject

The AutoBillStatus object describes the status of the AutoBill object. Please note that the Status displayed in the Portal differs from that available through the API. Please use the CashBox Portal to view a more granular status for **Active** AutoBills.

Table 4-4 AutoBillStatus Object Data Members

| Data Members | Description | Status (CashBox GUI) |
|---|---|---|
| Active | The AutoBill object was recently created and CashBox has not yet completed its first billing transaction. This status is also used for a new AutoBill object that is due to start at a future date. | **New** |
| | The AutoBill is currently in force. This status is superseded by the end-date on the AutoBill object. After that date, the status remains Active, but billing by AutoBill stops unless the date is extended by the next billing. | **Good Standing** |
| | This status is used if the AutoBill is paid with payment methods such as ECP or PayPal. If the AutoBill object's status was **New** and the first transaction processed by CashBox reached an AuthorizedPending status (meaning that the payment processor has accepted the transaction but further action by the customer or a bank is necessary), CashBox sets the AutoBill's status to **Pending**. | **Pending** |
| | If CashBox processes a transaction for an AutoBill, but the payment processor declines it with a return code that suggests that the transaction might succeed on a retry, CashBox sets the AutoBill's status to **Soft Error**. That means CashBox will attempt to reprocess the transaction on a date determined by your retry schedule. | **Soft Error** |
| Cancelled | The customer has opted out of recurring billing, or has cancelled the service. This status is reached if you call cancel on the AutoBill object or stopAutoBilling on the Account object, or if a chargeback is received against a transaction generated by the AutoBill object. Note that the customer who owns the AutoBill object is entitled until the AutoBill's end-date. | **Stopped** |

Table 4-4 `AutoBillStatus` Object Data Members *(Continued)*

| Data Members | Description | Status (CashBox GUI) |
|---|---|---|
| `PendingCustomerAction` | The `AutoBill` object has been created but active billing will not start until the customer has completed a step that validates the `AutoBill` object's payment method. This status is reached if an `AutoBill` is paid through a payment method such as PayPal. Be sure to create such an `AutoBill` object with payment method validation turned on, so that the customer must complete the validation and confirm a recurring billing agreement on the PayPal site.<br>The `AutoBill` object will remain in this status until the customer has completed validation. If your customer does not validate the `Transaction` within 3 hours of initiation, CashBox will automatically cancel the `Transaction`, which will cause the `AutoBill` to move into a status of Hard Error. | **Pending Customer Action** |
| `Suspended` | The `AutoBill` object is no longer active because of a hard fail of the last transaction billing. That means the payment processor rejected the transaction with a return code that indicates that the transaction will not be approved even on a retry. | **Hard Error** |
| `Upgraded` | The `AutoBill` has been upgraded. You must explicitly set this status by calling `update()` on the `AutoBill` object. | **Upgraded** |

## BillingPlanHistoryRecord Subobject

This object is returned by the AutoBill.modify call, and lists a record of the period during which a BillingPlan was associated with the AutoBill.

The following table lists and describes the data members of the BillingPlanHistoryRecord subobject.

Table 4-5 BillingPlanHistoryRecord Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| merchantBillingPlanId | string | Identifier for the BillingPlan |
| startDate | dateTime | The date the BillingPlan was first associated with the AutoBill. |
| endDate | dateTime | The Date the BillingPlan was removed from the AutoBill.<br>**Note:** The endDate will be blank for the current BillingPlan. |

## PaymentDetails Subobject

This object is returned by the AutoBill.fetchRemainingPaymentDetails call, and lists payment details for the AutoBillItem.

The following table lists and describes the data members of the PaymentDetails object.

Table 4-6 PaymentDetails Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| autobillItemVid | string | Vindicia's unique name (VID) for the AutoBillItem. |
| merchantAutoBillItemId | string | Your unique identifier for this AutoBillItem object. |
| merchantProductId | string | Your unique identifier for the product. If you track your products internally by SKU, use the SKU as your merchantProductId, to allow you to map your local records to CashBox Transactions that have this AutoBillItem as a line item. |
| productVid | string | Vindicia's unique name (VID) for the Product. |
| remainingBalanceInSet | decimal | The balance remaining in the AutoBill for the AutoBillItem. |
| remainingPaymentsInSet | int | The number of payments remaining in the AutoBill for the AutoBillItem. |

## 4.3 AutoBill Methods

The following table summarizes the methods for the AutoBill object.

Table 4-7 AutoBill Object Methods

| Method | Description |
|---|---|
| addCampaign | Adds a Campaign to an existing AutoBill. |
| addCharge | Adds a non-recurring charge to an AutoBill. |
| cancel | Cancels an AutoBill object. |
| changeBillingDayOfMonth | Updates the monthly billing day. |
| delayBillingByDays | Delays the next billing by the specified number of days. |
| delayBillingToDate | Delays the next billing until the specified date. |
| fetchAllCreditHistory | Returns credit history for all AutoBills. |
| fetchAllInSeason | Returns an array of all in season AutoBills. |
| fetchAllOffSeason | Returns an array of all off-season AutoBills. |
| fetchByAccount | Returns one or more AutoBill objects whose Account object matches the input. |
| fetchByAccountAndProduct | Returns all AutoBill objects whose Account and Product objects match the input. |
| fetchByEmail | Returns one or more AutoBill objects whose email address matches the input. |
| fetchByMerchantAutoBillId | Returns an AutoBill object whose ID assigned by you (merchantAutoBillId) matches the input. |
| fetchByVid | Returns an AutoBill object whose VID matches the input. |
| fetchByWebSessionVid | Returns an AutoBill object whose WebSession VID matches the input. |
| fetchCreditHistory | Returns an audit log of credit-related events for the specified AutoBill, or for all AutoBills. |
| fetchDailyInvoiceBillings | Returns an array of Transaction objects, with MerchantAcceptedPayment Payment Methods, that must be billed for the day. |
| fetchDeltaSince | Returns one or more AutoBill objects whose status has changed since the specified timestamp. |
| fetchFutureRebills | Returns an array of planned future billing Transactions, that do not yet exist in CashBox, for the specified AutoBill object. |
| fetchInvoice | Returns an Invoice for the given invoice ID as plain text or a PDF. |
| fetchInvoiceNumbers | Fetches the list of invoice numbers of invoices in the given state, for the given AutoBill. |
| fetchRemainingPaymentDetails | Returns information on an AutoBill's remaining payments after the most recent payment. |

Table 4-7 `AutoBill` Object Methods *(Continued)*

| Method | Description |
|---|---|
| `fetchUpgradeHistoryByMerchantAutoBillId` | Returns an AutoBill's upgrade history, given the `merchantAutoBillId`. |
| `fetchUpgradeHistoryByVid` | Returns an AutoBill's upgrade history, given the `VID`. |
| `finalizeCustomerAction` | Completes processing of a Transaction after the customer finishes payment activities at the payment provider-hosted web pages, and is redirected to your site. |
| `finalizePayPalAuth` | Informs CashBox about the final authorization status of a validation transaction generated when you create an `AutoBill` paid for with a PayPal-based payment method. |
| `grantCredit` | Adds credit to an `AutoBill`. Token- and currency-based credit thus added are stored in the `AutoBill`'s `credit` data member. Time-based credit thus granted to the `AutoBill` is immediately applied to the `AutoBill` by extending it. |
| `makePayment` | Enters a payment against an `AutoBill`. |
| `migrate` | Allows you to import data to CashBox for billing cycles completed through a different system. |
| `modify` | Allows you to change an AutoBill, while maintaining its history. |
| `redeemGiftCard` | Redeems a specified gift card, and adds equivalent credit to the `AutoBill`. |
| `reversePayment` | Reverses an `AutoBill` payment made using `makePayment`. This method may only be used with payments using `MerchantAcceptedPayment` payment methods. |
| `revokeCredit` | Deducts from credit available to an `AutoBill`. Time-based credit cannot be revoked. |
| `update` | Creates a new `AutoBill` object, or updates an existing one. |
| `writeOffInvoice` | Marks an `Invoice` object as `writtenOff`, the debt unable to be collected. |

## addCampaign

The `addCampaign` method allows you to add a Campaign to an existing `AutoBill`.

This method will automatically validate the Campaign, and its eligibility, before adding the Campaign, and updating the AutoBill, and will not change the billing date for the AutoBill.

> **Note:** If neither ***product*** nor ***item*** is passed in with this call, CashBox will apply the Campaign discount to all eligible items on the AutoBill.

### *Input*

***autobill:*** the object of type `AutoBill` to which this Campaign should be added.

***product:*** an array of `Products` to which the Campaign discount should be applied. (This product must already exist in CashBox.) The discount will be applied to any `AutoBillItem` on the specified AutoBill that includes this product. Specify either ***item*** or ***product***; but not both. (**Optional.**)

***item:*** the `AutoBillItem` to which the Campaign discount should be applied. This item must already exist in CashBox and be associated with the specified AutoBill. Specify either ***item*** or ***product***; but not both. (**Optional.**)

***applyToBillingPlan:*** a Boolean flag that, if set to `true`, will apply the `Campaign` to the `BillingPlan` on the `AutoBill`. (May be combined with a discounted `AutoBillItem`.) Default is `false`.

***campaignCode:*** the Coupon or Promotion Code used to obtain a discount on the `AutoBill`. (**Required.**)

***dryrun:*** a Boolean flag that, if set to `true`, will return the updated `AutoBill`, without recording the result in the CashBox database. Use this method to compute the cost of an `AutoBill` without committing the change. (The projected billing amount will be returned in the `Transaction` object of the `nextBilling` data member of the returned `AutoBill`.)

If the `AutoBill` did not exist before, it will not exist afterward; if it did exist before, it will not change. (No payment method validations, authorizations or charges will be performed if ***dryrun*** is `true`.)

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobill:*** the updated `AutoBill`.

## *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `AutoBill not found.`<br>• `Campaign code` ***`input-campaignCode`*** `is not usable.`<br>• `Must specify a campaignCode with addCampaign.` |

## *Example*

```
$autobill = new AutoBill();
    $autobill->setMerchantAutoBillId($abID);// for some $abID
    $response = $autobill->addCampaign(
        'promoABC',
    );

    // check $response
```

## addCharge

The `addCharge` method allows you to add a non-recurring charge to an `AutoBill`.

### Input

***autobill:*** the object of type `AutoBill` to which this addition applies.

***sku:*** the SKU for the charge added to `AutoBill`. If SKU is specified, and matches a Product `merchantProductId`, and `amount` is `null`, an attempt will be made to determine the charge `amount` from the Product.

***description:*** a text string description of the charge. (256 or fewer characters.)

***amount:*** the amount to charge. Required, unless the price is based on the SKU.

***currency:*** the ISO 4217 currency code for the amount. Either token, or currency must be specified.

***token:*** the `Token` associated with the amount (if this is a Token-based AutoBill). Either token or currency must be specified.

***quantity:*** the value to be included in charge. Defaults to 1, if not specified.

***campaignCode:*** Optional Coupon or Promotion Code, used to obtain a discount on this charge.

***dryrun:*** a Boolean flag that, if set to `true`, will return the updated `AutoBill`, without recording the result in the CashBox database.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `AutoBill not found.`<br>• `Failed to load token:` ***error-description.***<br>• `Failed to add charge to AutoBill:` ***error-description.*** |

### Example

```
$autobill = new AutoBill();
$autobill->setMerchantAutoBillId($abID);             // for some $abID

$response = $autobill->addCharge(
   'prod-bac',// product Id
   'fee for swapping tiles',
   null,                          // will get tax class from Product
   1.50,
   'USD',
   null,     // not a token
   1         // just once
);
// check $response
```

## cancel

The `cancel` method cancels a subscription (`AutoBill`).

A cancelled `AutoBill` object no longer generates periodic billing transactions. However, if CashBox has already picked up a current billing transaction to send to your payment processor, this call does **not** cancel the transaction, and you might choose to refund it later.

With this method, you may cancel an `AutoBill` object within the minimum commitment period by enabling the ***force*** option.

Cancelling an `AutoBill` does not automatically disentitle the customer immediately. Calling `cancel` on an `AutoBill` allows entitlement to continue, as determined by the last successful billing. If you wish to disentitle immediately upon cancellation of the `AutoBill`, set the ***disentitle*** flag to `true`.

Cancelling an `AutoBill` before the minimum commitment period is over, will stop the `AutoBill`, but allow the customer to continue to access the service. To immediately disentitle the customer, set the ***disentitle*** flag to `true` when making this call.

### *Input*

***autobill:*** the `AutoBill` object to cancel. You can identify this object with either its VID or your `AutoBill` ID (`merchantAutoBillId`).

***disentitle:*** a Boolean flag that, if set to `true`, cancels the related entitlements immediately. Otherwise, the entitlements will last till the end-date, as determined by the last successful billing for this `AutoBill` object.

***force:*** a Boolean flag that, if set to `true`, cancels the subscription even if the minimum commitment period for this `AutoBill` object is not over yet.

***settle:*** a Boolean flag that specifies whether to settle the `AutoBill` when it is cancelled. If `true`, an attempt will be made to settle the `AutoBill` by either refunding the customer for a portion of the pre-paid use that will not be available, or by charging the customer for non-recurring-charges that the customer has not yet paid. If `false` or not specified, the charges or credits remaining on the `AutoBill` will not be changed, and will be carried forward in the balance for the `AutoBill`. How the `AutoBill` is settled will be reflected by the ***transactions***/***refunds*** included in the output.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobill:*** the `AutoBill` object that was cancelled.

***transactions:*** an array of `Transaction` objects that may be refunded if the ***settle*** input field is set to `true`.

***refunds:*** an array of `Refund` objects that may be refunded if the ***settle*** input field is set to `true`.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• Unable to load AutoBill: No match.<br>• Unable to load AutoBill: ***error-description.***<br>• Error saving AutoBill: ***error-description.*** |
| 403 | Minimum commitment not fulfilled for this AutoBill. |
| 405 | Unable to cancel upgraded AutoBill. |

## Example

```
$autobillVid = '14e1dce6f48e901464fce22145982a59642aa9f4';

// Create an autobill object
$autobill = new AutoBill();
$autobill->setVID($autobillVid);
$immediateDisentitlement = false;
$force = true; // allowing to cancel even if min commitment
          // is not fulfilled
$response = $autobill->cancel($immediateDisentitlement, $force)
if($response['returnCode'] == 200) {
   $cancelledAutoBill = $response['data']->autobill;

   print "AutoBill has been successfully cancelled\n";

   // If you are using CashBox API version 3.3 or greater you can
   // also use the following construct

   print "You are entitled to use current services till "
          . $cancelledAutoBill->getEndTimestamp() . "\n";

}
```

## changeBillingDayOfMonth

The `changeBillingDayOfMonth` method updates the monthly billing day for a customer's subscription, assuming that the `AutoBill` object that represents the subscription is in good standing. If the next `AutoBill` billing has not yet been processed, this method also adjusts its date.

This method is useful if monthly or yearly billing plans apply to the `AutoBill` object. Once you have updated the billing day with this method, subsequent billing will happen on the same day of every month or year. However, if your billing plan is in days or weeks, this method changes only the next billing date and CashBox will compute the subsequent billing dates according to the duration of your billing period.

### Input

***autobill:*** the `AutoBill` object whose monthly billing day you would like to update. You can identify this object with either its VID or your `AutoBill` ID (`merchantAutoBillId`).

***dayOfMonth:*** the numeric day of the month (1 to 31) for the new billing. This method automatically adjusts this day for the months in which the day is not reached. For example, if ***dayOfMonth*** is 31, the billing for February occurs on either the 28th or the 29th.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobill:*** the `AutoBill` object whose monthly billing day was updated.

***nextBillingDate:*** the date of the next billing, if available.

***nextBillingAmount:*** the amount of the next billing, if available.

***nextBillingCurrency:*** the currency of the next billing, if available.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Date to delay to must be specified and have a length.`<br>• `No AutoBill specified in arguments.`<br>• `Unable to delay billing to date:` ***internal-error.***<br>• `Unable to change billing day of month:` ***error-description*** |

### Example

```
// to change the billing day of the month

$autobill = new AutoBill();
$autobill->setMerchantAutoBillId('xyz');
$response = $autobill->changeBillingDayOfMonth($productVid, 15);
if($response['returnCode'] == 200) {
    $nextBillingDate = $response['nextBillingDate'];
    $nextBillingAmt = $response['nextBillingAmount'];

    print "Customer will be billed on " . $nextBillingDate.
           " for US $" . $nextBillingAmt . "\n";
}
```

## delayBillingByDays

The `delayBillingByDays` method delays the next billing and extends the `AutoBill` end-date, which corresponds to that for the entitlements granted by `AutoBill`, by the specified number of days. Call this method to credit the customer with additional subscription time, by postponing a customer's next billing by a finite duration.

### Input

***autobill:*** the `AutoBill` object whose billing to delay. Identify this object by populating it with its VID or `merchantAutoBillId`.

***delayDays:*** the number of days by which to delay the billing. (Must be a positive integer.)

***movePermanently:*** *this parameter is not in use.*

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobill:*** the `AutoBill` object whose billing was delayed by the input number of days.

***nextBillingDate:*** the date of the next billing after the delay is in effect.

***nextBillingAmount:*** the amount of the next billing after the delay is in effect.

***nextBillingCurrency:*** the currency of the next billing after the delay is in effect.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Days to delay must be a positive integer.`<br>• `Must specify AutoBill to delay billing for.`<br>• `Unable to delay billing to date:` ***error-description.*** |

### Example

```
$autobill = new AutoBill();
$autobill->setMerchantAutoBillId('xyz');
$response = $autobill->delayBillingByDays(25, true);

if($response['returnCode'] == 200) {
   $nextBillingDate = $response['nextBillingDate'];
   $nextBillingAmt = $response['nextBillingAmount'];
   print "Customer will be billed on " . $nextBillingDate.
          " for US $" . $nextBillingAmt . "\n";
}
```

## delayBillingToDate

The `delayBillingToDate` method is similar to `delayBillingByDays` but delays the next billing for an `AutoBill` object until the specified date. Instead of specifying the number of days for the delay, you specify the exact date on which you would like the next billing to occur.

### *Input*

***autobill:*** the `AutoBill` object whose billing to delay. Identify this object by populating it with its VID or `merchantAutoBillId`.

***newBillingDate:*** the date until which to delay billing.

***movePermanently:*** *this parameter is not in use.*

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobill:*** the `AutoBill` object whose billing was delayed to the specified date.

***nextBillingDate:*** the date of the next billing.

***nextBillingAmount:*** the amount of the next billing.

***nextBillingCurrency:*** the currency of the next billing.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Must specify AutoBill to delay billing for.`<br>• `Date to delay to must be specified and have a length.`<br>• `Unable to delay billing to date:` ***internal-error***`.` |

## *Example*

```
// to delay billing to a specified date

$autobill = new AutoBill();
$autobill->setMerchantAutoBillId('xyz');

    $darray = getdate();
    $day = $darray[mday];
    $year = $darray[year];
    $mon = $darray[mon] + 6;
    if ($mon > 12) {
        $year++;
        $mon -= 12;
    }
$timestamp = mktime(0, 0, 0, $mon, $day, $year);

$response = $autobill->delayBillingToDate($timestamp, true);

if($response['returnCode'] == 200) {
    $nextBillingDate = $response['nextBillingDate'];
    $nextBillingAmt = $response['nextBillingAmount'];

    print "Customer will be billed on " . $nextBillingDate.
           " for US $" . $nextBillingAmt . "\n";
}
```

## fetchAllCreditHistory

The `fetchAllCreditHistory` method returns credit history for all AutoBills.

For more information, see the `Account` object's `fetchAllCreditHistory` method.

### Input

***timestamp:*** the starting timestamp (lower limit) for the range of credit event logs you wish to retrieve.

***endTimestamp:*** the ending timestamp (upper limit) for the range of credit event logs you wish to retrieve.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***creditEventLogs:*** an array of `CreditEventLog` objects. Each of these objects describes a specific credit-related event or action associated with the input `AutoBill`. (See Table 1-9: `CreditEventLog` Object Data Members for details.)

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$rc = $autobill_factory->fetchAllCreditHistory("2012-12-25",
       "2013-01-01", 0, 100);

// from Christmas to New Year's, first page, limit 100 to the page
// check response in $rc

$event_log_array = $rc->{creditEventLogs};
foreach ($event_log_array as $event_log)
{
   print $event_log->timeStamp,
       "\t",
          $event_log->type,
          "\t",
       $event_log->credit->currencyAmounts->amount,
          "\n";
}
```

> **Note:** This example assumes that the credits are in currency amounts, and therefore specifies a minimal number of parameters.

## fetchAllInSeason

The `fetchAllInSeason` method returns an array of all in season AutoBills.

### Input

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and pageSize is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

***nowDate:*** the (optional) date to query. (Defaults to `today`.)

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autoBills:*** an array of in season `AutoBill` objects.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$rc = $autobill_factory->fetchAllInSeason(0, 100, "2013-10-01");
// check $rc
$ab_array = $rc->{autoBills};
foreach ($ab_array as $ab)
{
   print $ab->merchantAutoBillId, "\n";
}
```

| **Note:** | This example returns all AutoBills that were in season 2013-10-01. |
|---|---|

## fetchAllOffSeason

The `fetchAllOffSeason` method returns an array of all off-season `AutoBills`.

### Input

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize***: the number of records to display per page per call. This value must be greater than 0.

***nowDate:*** the (optional) date to query. (Defaults to ***today***.)

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autoBills:*** an array of off-season `AutoBill` objects.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$rc = $autobill_factory->fetchAllOffSeason(0, 100, "2013-10-01");
// check $rc
$ab_array = $rc->{autoBills};
foreach ($ab_array as $ab)
{
   print $ab->merchantAutoBillId, "\n";
}
```

| **Note:** | This example returns all AutoBills that were ***not*** in season on 2013-10-01. |
|---|---|

## fetchByAccount

The `fetchByAccount` method returns one or more `AutoBill` objects whose `Account` object matches the input. This method is useful for looking up a customer's subscriptions on your site.

### Input

***account:*** the `Account` object that serves as the search criterion. Use the `merchantAccountId` or `VID` to identify the object.

***includeChildren:*** an optional Boolean flag that, if set to `true`, includes any children associated with this `Account`. If this flag is omitted, CashBox will interpret it as `false`, and constructs the query without looking at any child's account.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobills:*** an array of one or more `AutoBill` objects whose `Account` object matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Data validation error: ***error-description.*** |

### Example

```
// Create and populate an Account object
$account = new Account();
$account->setMerchantAccountId('abc101');

$autobill = new AutoBill();

$response = $autobill->fetchByAccount($account);
if($response['returnCode'] == 200) {
   $fetchedAutoBills= $response['data']->autobills;

   foreach ($fetchedAutoBills as $autobill) {

   // process each autobill found here
   print "Found account with id: "
          . $autobill->getMerchantAutoBillId() . "\n";

   }
}
```

## fetchByAccountAndProduct

The `fetchByAccountAndProduct` method returns all `AutoBill` objects that have the passed in `Account` and `Product`. This method is useful for looking up a customer's subscriptions to a specific product on your site.

### Input

***account:*** the `Account` object that serves as one of the two search criteria. Use the `merchantAccountId` or `VID` to identify the object.

***product:*** the `Product` object that serves as one of the two search criteria. Identify this object with either its VID or your product ID (`merchantProductId`).

***includeChildren:*** an optional Boolean flag that, if set to `true`, includes all children associated with this `Account`. If `false` or omitted, children will not be included in the query.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobills:*** an array of one or more `AutoBill` objects whose `Account` and `Product` objects match the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Data validation error: ***error-description.*** |

### Example

```
// Create and populate an Account object
$account = new Account();
$account->setMerchantAccountId('abc101');

// Create and populate an Product object
$prod = new Product();
$prod->setMerchantProductId('xyz212');

$autobill = new AutoBill();

$response = $autobill->fetchByAccountAndProduct($account, $prod);
if($response['returnCode'] == 200) {
   $fetchedAutoBills= $response['data']->autobills;

   foreach ($fetchedAutoBills as $autobill) {

      // process each autobill found here
      print "Found account with id: "
          . $autobill->getMerchantAutoBillId() . "\n";
   }
}
```

## fetchByEmail

The `fetchByEmail` method returns one or more `AutoBill` objects associated with the `Account` objects whose email address matches the input. This method is useful for identifying all the subscriptions for a specific email address.

### Input

***email:*** the email address (a string) that serves as the search criterion.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobills:*** an array of one or more `AutoBill` objects whose `Account` objects' email address matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Must specify email address to load by! |
| 404 | No AutoBills found for email address ***input-email-address:*** No match. |

### Example

```
$autobill = new AutoBill();

$response = $autobill->fetchByEmail('xyz@mail.com');
if($response['returnCode'] == 200) {
    $fetchedAutoBills= $response['data']->autobills;

    foreach ($fetchedAutoBills as $autobill) {

        // process each autobill found here
        print "Found account with id: "
            . $autobill->getMerchantAutoBillId() . "\n";
    }
}
```

## fetchByMerchantAutoBillId

The `fetchByMerchantAutoBillId` method returns an `AutoBill` object whose ID assigned by you (`merchantAutoBillId`) matches the input.

### Input

***merchantAutoBillId:*** your `AutoBill` ID (`merchantAutoBillId`), which serves as the search criterion.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobill:*** the `AutoBill` object whose ID assigned by you (`merchantAutoBillId`) matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `No autobill matches serial number` ***input-merchantAutoBillId.***<br>• `Unable to load autobill by serial number` ***input-merchantAutoBillId: error-description.*** |

### Example

```
// Create a SOAP caller object
$autobill = new AutoBill();
$abId = "34583";

$response = $autobill->fetchByMerchantAutoBillId($abId);
if($response['returnCode'] == 200) {
   $fetchedAutoBill = $response['data']->autobill;
}
else {
   // The call was unsuccessful
   print "Return code: " . $response['returnCode'] . "\n";
   print "Return string: " . $response['returnString'] . "\n";
}
```

## fetchByVid

The `fetchByVid` method returns an `AutoBill` object whose VID matches the input.

When you create a new `AutoBill` object with the `update()` call, CashBox assigns the object a unique ID (VID), which is inside the `AutoBill` object returned to you by the call. Store this VID locally to use it to retrieve or reference the `AutoBill` object in later calls.

### *Input*

***vid:*** the `AutoBill` object's Vindicia identifier, which serves as the search criterion.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobill:*** the `AutoBill` object whose VID matches the input.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load autobill by VID` ***input-vid:*** *`error`*.<br>• `Must specify VID to load by!` |
| `404` | `Unable to load autobill by VID` ***input-vid:*** `No match.` |

### *Example*

```
$autobillVid = '8367ae7148d071a4e25c24bef856f68f71ee03e3';

// Create an autobill object
$autobill = new AutoBill();

// now load an autobill into the autobill object by VID
$response = $autobill->fetchByVid($autobillVid);

if($response['returnCode'] == 200) {
    $fetchedAutoBill = $response['data']->autobill;

    // process fetched autobill here
}
```

## fetchByWebSessionVid

Use Vindicia's Hosted Order Automation (HOA) feature to create CashBox objects that contain sensitive payment information, such as credit-card account numbers, directly on Vindicia's servers, after your customers have submitted such data through a specially designed Web order form you serve from your server. Because HOA bypasses your server altogether at form submission, you need not comply with PCI requirements. See Chapter 13: Hosted Order Automation in the ***CashBox Programming Guide*** for details.

Within your HOA implementation, you may call the `fetchByWebSessionVid` method to retrieve the `AutoBill` object, created by HOA on Vindicia's servers when a customer submits an order form which results in a one-time or recurring bill. You must create a `WebSession` object on Vindicia's servers before serving the form to your customer to track the form's submission to Vindicia. For more information, see Section 19: The WebSession Object.

The `WebSession` object's VID serves as the tracking ID for various activities, starting from serving the order form to a customer, and ending in returning a success or failure page to that same customer. This method is useful when programming the success page (see the `returnURL` attribute in Section 19.1: WebSession Data Members), to which HOA redirects the customer's browser after successfully processing the data in the order form. On the success page, the `WebSession` object's VID is available to you because HOA passes it during the redirection. In turn, you can pass that VID as the input parameter to this call, and retrieve the `AutoBill` object created by HOA. Finally, you can extract the contents of the `AutoBill` object and include them, as appropriate, in the success page to be returned to the customer.

### *Input*

***vid:*** the `WebSession` object's Vindicia unique identifier for tracking the submission of the order form.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobill:*** an `AutoBill` object that was created by HOA as a result of an order form submitted by a customer.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Missing required parameter 'vid'. |
| 404 | Unable to find requested AutoBill: No matches. |

### *Example*

```
// to use the fetchByWebSessionVid call on a success web page

$webSessionVid = …; //passed in by redirected page

$soap = new WebSession($soapLogin, $soapPwd);

$response = $soap->fetchByVID($webSessionVid);

if ($response['returnCode'] == 200) {

    $fetchedWs = $response['data']->session;

    // check if the CashBox API call made by HOA was successful

$retCode = $fetchedWs->apiReturn->returnCode;

if ($retCode == 200) {

    // Assuming HOA created an AutoBill object, let's fetch it

    $soapAbill = new AutoBill($soapLogin, $soapPwd);
    $resp = $soapAbill->fetchByWebSessionVid($webSessionVid);

    if ($resp['returnCode'] == 200) {
        $createdAutoBill = $resp['data']->autobill;

        // Get AutoBill contents here to be included in
        // HTML returned to the customer.
    }
    else {
        // Return error message to customer
    }
}
else {
    // return failure page to customer
}
}
else {
    // Return error message to the customer
}
```

## fetchCreditHistory

The `fetchCreditHistory` method returns an array of `CreditEventLog` objects.

For more information, see the Account object's `fetchAllCreditHistory` method, and Table 1-9: `CreditEventLog` Object Data Members.

### Input

***autobill:*** the (optional) `AutoBill` object for which you wish to retrieve credit event history. Use the `AutoBill`'s `merchantAutoBillId` or `VID` to identify it. Leave this variable blank if you wish to fetch credit history across all `AutoBills`.

***timestamp:*** the starting timestamp (lower limit) for the range of credit event logs you wish to retrieve.

***endTimestamp:*** the ending timestamp (upper limit) for the range of credit event logs you wish to retrieve.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize***: the number of records to return per call. This value must be greater than 0.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***creditEventLogs:*** an array of `CreditEventLog` objects. Each of these objects describes a specific credit-related event or action associated with the input `AutoBill`. See Table 1-9: `CreditEventLog` Object Data Members for details.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load autobill.`<br>• `Invalid value or values of timestamp, and/or page, and/or page size.` |
| `404` | `No matching credit events found.` |

## *Example*

```
// to fetch credit history for an AutoBill

$abill = new AutoBill();

// autobill id for an existing customer whose
// credit history you want to retrieve

$abill->setMerchantAccountId('jdoe101');

$page = 0; // paging begins at 0
$pageSize = 5; // five records
$startTime = '2010-01-01T22:34:32.265Z';
$endTime = '2010-01-30T22:34:32.265Z';

do {
    $ret =
        $abill->fetchCreditHistory($startTime, $endTime $page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedLogs = $ret['creditEventLogs'];
        $count = sizeof($fetchedLogs);
        foreach ($fetchedLogs as $log) {
            $credit = $log->getCredit();
            $ts = $log->getTimeStamp();
            $eventType = $log->getType();
            // process retrieved credit event log
            // details here.
        }
        $page++;
    }
} while ($count > 0);
```

## fetchDailyInvoiceBillings

The `fetchDailyInvoiceBillings` method returns an array of `Transaction` objects, with `MerchantAcceptedPayment` Payment Methods, that must be billed for the day.

### Input

***startTimestamp:*** the starting timestamp for the range of `Transactions` you wish to retrieve. If set, the fetch will return `Transactions` with a timestamp on or after the day of ***startTimestamp***. If not set, the method will return `Transactions` beginning with the day prior to the day the method is called.

***endTimestamp:*** the ending timestamp for the range of `Transactions` you wish to retrieve. If set, the fetch will return `Transactions` with a timestamp on or before the day of ***endTimestamp***. If not set, the method will return `Transactions` with a timestamp on or before the ***startTimestamp*** input.

> **Note:** If neither ***startTimestamp*** nor ***endTimestamp*** are specified, the fetch will return `Transactions` with a timestamp of the day previous to the day the method is called.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transactions:*** an array of returned `Transaction` objects.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$ab = new AutoBill($login, $password);
$page = 0;
$pageSize = 100;
$startTime = '2012-01-01T00:00:00.000Z';
$endTime = '2012-01-01T23:59:59.000Z' ;
do {
    $ret = $ab->fetchDailyInvoiceBillings($startTime,
           $endTime, $page, $pageSize);
    $count = 0;
    if($ret['returnCode'] == 200) {
        $fetchedTransactions = $ret['transactions'];
        $count = sizeof($fetchedTransactions);
        foreach ($fetchedTransactions as $transaction)
        {
             //process a fetched transaction here…
        }
        $page++;
    }
    else
    {
        //handle error condition
    }
} while($count == $pageSize);
```

## fetchDeltaSince

The `fetchDeltaSince` method returns one or more `AutoBill` objects whose status has changed since the specified timestamp. Call this method to discover which `AutoBill` objects still active, and which are not. The inactive status might be triggered by several events, including a “hard error” received by CashBox while processing a payment with a payment processor, a `cancel()` call that explicitly stopped an `AutoBill` object, or a chargeback against a billing transaction generated by `AutoBill`.

This method supports paging to limit the number of records returned per call. Occasionally, returning a large number of records in one call swamps buffers and might cause a failure. Vindicia recommends that you call this method in a loop, incrementing the page for each loop iteration with an optimal page size (number of records returned in one call) until the page contains a number of records that is less than the given page size.

| **Note** | Do not rely on the data returned by this method to determine a customer’s entitlements. Even if an `AutoBill` object is in `Stopped` or `Hard Error` status, the entitlements might still be valid. The entitlements are determined by the `AutoBill` object’s current end-date, which, in turn, is determined by the success of the last billing (transaction). Thus, the end-date indicates the period for which the customer has already paid. The customer’s subscription (`AutoBill`) may stop before the end-date, but the entitlements might remain valid until that date. The correct way to determine a customer’s entitlements is by making calls on the `Entitlement` object. |
|---|---|

### *Input*

***timestamp:*** the date and time after which to return the `AutoBill` objects whose status has changed.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

***endTimestamp:*** the time window’s upper threshold by which to limit the search. If unspecified, this value defaults to the current time.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobills:*** an array of one or more `AutoBill` objects.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Invalid value or values of timestamp, and/or page, and/or page size. |

## Example

```
$ab = new AutoBill();
$page = 0;
$pageSize = 100;
$startTime = '2010-01-01T22:34:32.265Z';
$endTime = '2010-01-02T22:34:32.265Z';
do {
    $ret = $ab->fetchDeltaSince($startTime, $page, $pageSize, $endTime);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedAutoBills = $ret['autobills'];
        $count = sizeof($fetchedAutoBills);
        foreach ($fetchedAutoBills as $autobill) {

            // process a fetched autobill here …
        }
        $page++;
    }
} while ($count > 0);
```

## fetchFutureRebills

The `fetchFutureRebills` method returns the planned future billing transactions, that do not yet exist in CashBox, for the specified `AutoBill` object. The returned `Transaction` objects are constructed on the fly in response to this call. You can then inform a customer how they will be billed for a subscription to your product or service with a certain billing plan.

(This method will calculate and return any discounts applied as a result of an applied Campaign Code.)

For this call to succeed, the `AutoBill` object must be in an actively billing state.

### Input

***autobill:*** the `AutoBill` object for which to obtain the future billing transactions. Identify this object with either its VID or `merchantAutoBillId`.

***quantity:*** the number of future rebill transactions to be returned by this call. This input must be a positive integer.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transactions:*** an array of `Transaction` objects, each of which corresponds to a projected billing for this `AutoBill` object. For for information, see Section 18: The Transaction Object.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Quantity must be a positive integer.`<br>• `No AutoBill specified in arguments.` |
| `404` | `No matching AutoBill found.` |
| `405` | `AutoBill is in an inactive state` ***<state>.*** |

### Example

```
$autobillVid = 'a209408014a33fec3dcd4a3339d78efc33603bfe';

// Create an autobill object
$autobill = new AutoBill();
$autobill->setVID($autobillVid);
$response = $autobill->fetchFutureRebills(5);

if($response['returnCode'] == 200) {
   $futureTxns = $response['data']->transactions;

   print "This subscription will be billed at the following
          dates and amounts:\n";

   for ($i = 0; $i < 5; $i++) {
      print "Date: " . $futureTxns[i]->getTimestamp() . " ";
      print "Amount: " . $futureTxns[i]->getAmount() . "\n";
   }
}
```

## fetchInvoice

The `fetchInvoice` method generates and returns an Invoice, for the given `invoiceId`, as plain text or as a PDF. (It does not fetch a previously sent Invoice.)

### Input

***autobill:*** the `AutoBill` for the requested invoice.

***invoiceId:*** the ID of the invoice.

***asPDF:*** a Boolean flag, which, if set to `true`, returns the object as a PDF. Default is `false`.

***statementTemplateId:*** the Merchant Identifier for a pre-defined statement template. If `null`, the template defined by the `AutoBill` will be used.

***dunningIndex:*** the index number of the requested invoice. (If the invoice was the first issued to the customer, its `dunningIndex` is 0.)

> For more information, see Section 9.3: Working with Invoices in the ***CashBox Programming Guide***.

***language:*** the language of the invoice.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***invoice:*** the specified invoice, rendered as a PDF or as plain text. This field encodes both plain text and PDF as xsd:base64Binary.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load AutoBill: No match.`<br>• `Unable to load AutoBill:` ***error-description.***<br>• `Unable to load TransactionBilling:` ***error-description.***<br>• `Unable to load StatementTemplate:` ***error-description.***<br>• `Failed to render invoice:` ***error-description.*** |

### Example

```
$autobill = new AutoBill();
$autobill->setMerchantAutoBillId($abID);      // for some $abID

$response = $autobill->fetchInvoice(
    'inv-bac',
    true,           // PDF, please
    null,
    0,
    'de-AT'         // German in Austria
);

if ($response['returnCode'] == 200) {
    $pdf = $response['data']->invoice;
}
```

## fetchInvoiceNumbers

Returns an array of `Invoices` matching the search criteria. (If no input parameters are specified, the 12 most recent `Invoice` objects will be returned.)

### Input

***autobill:*** an object of type `AutoBill` that contains the desired invoices.

***invoicestate:*** an optional object of type `InvoiceStates`, which limits the returned objects to the specified state: `Open`, `Due`, `Paid`, `Overdue`, or `WrittenOff`.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***invoicenum:*** the invoice number which uniquely identifies the fetched invoice within the `AutoBill`.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load AutoBill: No match.`<br>• `Unable to load AutoBill:` ***error-description.*** |

### Example

```
$autobill = new AutoBill();
$autobill->setMerchantAutoBillId($abID);      // for some $abID

$response = $autobill->fetchInvoiceNumbers('Due');

if ($response['returnCode'] == 200) {
    $inv_nums = $response['data']->invoicenum;
    foreach ($inv_nums as $inv_n) {
        // process invoice $inv_n
    }
}
```

## fetchRemainingPaymentDetails

The `fetchRemainingPaymentDetails` method returns `AutoBill` information after the most recent payment.

> **Note:** If an `AutoBillItem` has a price basis of `Included`, `fetchRemainingPaymentDetails` will return `undefined`.

### Input

***autobill:*** the `AutoBill` object to query.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobillRemainingBalanceInSet:*** the balance remaining on the `AutoBill`.

***billingPlanRemainingBalanceInSet:*** the balance remaining on the `BillingPlan`.

***billingPlanRemainingPaymentsInSet:*** the number of payments remaining on the `BillingPlan`.

***autobillItemRemainingPaymentDetails:*** an array of `PaymentDetails` objects, listing information about the payment due to each `AutoBillItem`.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$rc = $autobill_factory->fetchRemainingPaymentDetails($autobill);

// check response in $rc

print "AutoBill remaining balance is ",
       $rc->{autobillRemainingBalanceInSet}, "\n";
print "There are ", $rc->{billingPlanRemainingPaymentsInSet},
       " payments remaining.\n";
$pd_ar = $rc->{autobillItemRemainingPaymentDetails};

foreach ($pd_ar as $item_data)
{
   print "Payment details for product ",
          $item_data->merchantProductId, ":\n";
   print "    Item remaining balance is ",
          $item_data->remainingBalanceInSet, ".\n";
   print "    There are ", $item_data->remainingPaymentsInSet,
          " payments left.\n";
}
```

## fetchUpgradeHistoryByMerchantAutoBillId

The `fetchUpgradeHistoryByMerchantAutoBillId` method returns the specified AutoBill's upgrade history given the `MerchantAutoBillId`.

### Input

***merchantAutoBillId:*** the ID for any item in the AutoBill's upgrade history for which you want the entire history series.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***upgradeHistorySteps*** produces an array of steps or revisions in the AutoBill's history. The `AutoBillUpgradeHistoryStep` object contains

- `vid`: the Vindicia ID for the object,
- `startTimestamp`: the date and time the step began, and
- `endTimestamp`: the date and time the step ended. This timestamp is omitted if the step is current.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

See `fetchUpgradeHistoryByVid` for an example of how to process the return parameters.

## fetchUpgradeHistoryByVid

This method allows you to track customers' changes in products, billing plans, and payment methods, based on the `AutoBill`'s `VID`. If you provide the VID for any item in the `AutoBill`'s upgrade history, CashBox will return the entire series of upgrades.

Note that the VID changes each time a customer upgrades the `AutoBill`. Use `fetchAutoBillUpgradeHistoryByMerchantAutoBillId` to generate a complete list of VIDs for the `AutoBill`.

### *Input*

***vid:*** the ID for any revision in the `AutoBill`'s upgrade history.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***upgradeHistorySteps:*** an array of steps or revisions in the AutoBill's history. The `AutoBillUpgradeHistoryStep` object contains

- `vid`: the Vindicia ID for the object,
- `startTimestamp`: the date and time the step began, and
- `endTimestamp`: the date and time the step ended. This timestamp is omitted if the step is current.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$autobillVid = 'a458e923453e3e2737a4f2142b396b100fbc8d3a';

// This code sample shows how to fetch the chain of upgraded autobills

$autobill = new AutoBill();

// now fetch the upgrade history
$response = $autobill->fetchUpgradeHistoryByVid($autobillVid);

if($response['returnCode'] == 200) {
    $history = $response['data']->upgradeHistorySteps;
    $first_autobill_vid = $history[0]->vid;
}
```

## finalizeCustomerAction

Completes processing of a transaction after the customer finishes payment activities at the payment provider hosted web pages, and is redirected to your site.

| | |
|---|---|
| **Note:** | This method works only for Direct Debit payment products. Calling any other payment product with this method will fail.<br>The customer's Account must exist before calling `finalizeCustomerAction`. |

The AutoBill will start billing only after this finalization is completed and the underlying transaction is authorized (captured).

| | |
|---|---|
| **Note:** | This flow requires that full amount auth be set to `true`. |

### *Input*

***transactionVid:*** Vindicia generated unique ID for the underlying transaction. This will be available to you through the URL when your customer is redirected to your site by the payment provider.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***authStatus:*** an object of type `TransactionStatus` that indicates the status of the initial Transaction. This object will also contain the response received from the payment provider.

***autobill:*** the `AutoBill` object for which this method finalized the `HostedPage` validation transaction. It contains the updated status of the `AutoBill` after the finalization. CashBox will populate this only if there was no error in processing this call.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
// Create an AutoBill with payment product = 712
$autobill = set_ab($identifier, "712");

// Call AutoBill.update with validate=1
$rc = $autobill->update($autobill, undef, 1, 99);

// Set the status of the AutoBill to "Active"
// in anticipation of success.
$rc = $autobill->finalizeCustomerAction($vin_id);

$status = $rc->{autobill}->status;
is ($status,"Active", "Status is Active (New)");
```

## finalizePayPalAuth

The `finalizePayPalAuth` method completes the authorization of a PayPal payment method validation transaction. This method enables you to report the status of the validation transaction to CashBox. Use this method only when you are working with an `AutoBill` that is paid for with a PayPal-based payment method.

CashBox generates the validation transaction when you create the `AutoBill` by calling the `update()` method with the ***validatePaymentMethod*** flag turned on. The `update()` call returns a PayPal site URL to you; ask your customer to visit that URL so that they may complete the authorization activities necessary to validate the payment method at PayPal's site. After the customer finishes the authorization at the PayPal Web site, and is redirected to your site, call `finalizePayPalAuth()` from either the success page (`returnUrl` specified in the PayPal payment method) or failure page (`cancelUrl` specified in the payment method) to which the customer was redirected. The `AutoBill` will start billing only after this finalization is completed and authorization of the underlying validation is known to CashBox.

For more information on applying tax to PayPal transactions, please see The Transaction Object's addressAndSalesTaxFromPayPalOrder method.

### *Input*

***payPalTransactionId:*** Vindicia's ID for the PayPal payment method validation `Transaction`, generated when you called `AutoBill.update`. Retrieve this ID from the value associated with the name: `vindicia_vid` in the name–value pairs attached to the redirect URL.

***success:*** set by you. Set it to `true` if the customer successfully authorized the validation transaction at PayPal's site and was redirected to the success page (`returnUrl`) hosted by you. If the customer was redirected to the failure page (`cancelUrl`), set this to `false`.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***authStatus:*** a `TransactionStatus` object. Its `payPalStatus` attribute contains return codes received from PayPal while authorizing the transaction.

***autobill:*** the `AutoBill` object for which this method finalized the PayPal validation transaction. It contains the updated status of the `AutoBill` after the finalization. For example, if the validation was successful, the `AutoBill` should have an `Active` status.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | ***Internal-error-string.*** |

**Note:** In some cases, after your customer has authorized payment on the PayPal site, PayPal will (invisibly) return a 10417 response code:

```
Hard Failure: Account not associated with a usable
funding source. Credit card or Billing Agreement is
required to complete payment method.
```

Upon calling `Transaction.finalizePayPalAuth`, CashBox will then return the error message:

```
Merchant and PayPal consider transaction to be in
different states: 0 vs. 1.
```

## Example

```
// to finalize a PayPal authorization

$soap_caller = new AutoBill();

// obtain id of the PayPal validation transaction
// from the redirect URL. It is the value associated with name
// 'vindicia_vid'

$payPalTxId = … ;

// if calling from return URL which is reached when the PayPal
// transaction is successfully authorized you should set the
// success input parameter to true

$success = true;
$response =
   $soap_caller->finalizePayPalAuth($payPalTxId, $success);

if($response['returnCode'] == 200) {
   printLog "PayPal validation transaction successful";
   $updatedAutoBill = $response['data']->autobill;
   printLog " AutoBill id " .
          $updatedAutoBill->getMerchantAutoBillId() . "\n";

   printLog " AutoBill status "
          . $updatedAutoBill->getStatus() . "\n";

}
```

## grantCredit

The `grantCredit` method adds credit to an `AutoBill` object. With credit available to an `Account`, you may extend the life of an `AutoBill` object, thus allowing a customer to keep their subscription active.

- Token-based credit may be granted to an AutoBill to pay for billing transactions. To grant Token-based credit to an AutoBill, the credit must be of the same token type as the Payment Method on the AutoBill, and the BillingPlan must also be defined in terms of the same Token Type. Token Credits granted will be deducted from the amount billed to the AutoBill's payment method at the next billing cycle.
- You may also grant time-based credit to an `AutoBill`. With a `TimeInterval` object, define a time extension to be given to an `AutoBill` in terms of years, months, weeks, or days. When you grant time credit to an `AutoBill`, CashBox delays the next billing for the `AutoBill` by the specified amount of time, similar to calling `delayBillingByDays()` on an `AutoBill` object. This delay does not occur until the billing date *following* the time credit grant. Until then, the time credit remains on the AutoBill, and the next billing date appears unchanged. Note that CashBox does not generate a transaction to account for such a time-based credit grant.
- When granting a currency credit to an `AutoBill`, the currency (i.e. USD) for the credit grants must be the same as the currency the customer has specified for the `AutoBill`.
- Time and Currency Credits may be tracked by timestamp and `sortValue`. When granted, they are assigned a VID, which may be used when revoking credit.

See the `Credit` Subobject for a list of data members of the `Credit` object and related subobjects. See Chapter 12: Credit Grants and Gift Cards in the ***CashBox Programming Guide*** for more information.

### *Input*

***autobill:*** an `AutoBill` object to which you wish to grant credit. Identify the `AutoBill` using its `merchantAutoBillId` or `VID`.

***credit:*** a `Credit` object specifying the amount and type of credit you wish to grant to the `AutoBill`.

***note:*** an optional note regarding the credit granted.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobill:*** the `AutoBill` object to which you granted credit. This object contains the updated array of `Credit` objects.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• AutoBill not found.<br>• Failed to translate credit ***error-description.***<br>• Failed to grant credit ***error-description.***<br>• Failed to save AutoBill after granting credit.<br>• Failed to reload AutoBill after granting credit ***error-description.***<br>• Time interval credit cannot have amount 0. |

## Example

```
$abill = new AutoBill();

// autobill id for an existing subscription

$abill->setMerchantAutoBillId('SBCR312345');

// We want to grant 2 days of credit
$time = new TimeInterval();
$time->setType('Day');
$time->setAmount(2);

$cr = new Credit();
$cr->setTimeIntervals(array($time));
$note = "optional note explaining credit grant";

// Now make the SOAP API call to grant credit to the autobill
$response = $abill->grantCredit($cr,$note);

if ($response['returnCode'] == 200) {

   // Credit successfully granted to the autobill

   $updatedABill = $response['data']->autobill;
   print "Current entitlements are valid till: ";
   print $updatedABill->getEndDate() . "\n";
}
else {
   // Error while granting credit to the account

   print $response['returnString'] . "\n";
}
```

## makePayment

The makePayment method allows you to record a payment against an outstanding invoice. This method may be used to enter check or cash payments, payment of goods in trade, or payments made with active Payment Methods.

Using the makePayment method on the AutoBill object will cause CashBox to allocate the payment directly to the selected AutoBill. To apply a payment against the oldest outstanding Invoice, use Account.makePayment instead.

Whether you use a standard PaymentMethod, or a MerchantAcceptedPayment, the makePayment method generates a Transaction, and processes the Transaction through the auth/capture cycle appropriate to the input Payment Method. Credit Card, ECP, PayPal, and other standard Payment Methods are routed through the appropriate Payment Processor. The MerchantAcceptedPayment Payment Method is routed through Vindicia's internal transaction process. Both Payment Method types appear as a Transaction in the Account's history.

> **Note:** When using a MerchantAcceptedPayment, you must create a new PaymentMethod object for each makePayment call.

### *Input*

***autobill:*** the AutoBill to which this payment applies.

***paymentMethod:*** the PaymentMethod to be used for this payment. (**Note:** Assign a unique ID for every Account.makePayment call that uses the MerchantAcceptedPayment Payment Method, for tracking purposes.)

> **Note:** PaymentMethods may not be duplicated for an Account. Passing in an existing credit card number and expiration date in an attempt to create a new PaymentMethod for an Account will return the pre-existing PaymentMethod instead.

***amount:*** the amount of the payment being made.

***currency:*** the ISO 4217 currency code for amount. This must match the currency used for charges on the current invoice. (If not specified, the AutoBill/Invoice currency will be used.)

***invoiceId:*** the ID of the Invoice against which the payment is to be made. If not specified, the oldest unpaid invoice for this AutoBill will be selected for payment.

***overageDisposition:*** defines how to allocate payments in excess of a required AutoBill payment amount. Defaults to applyToOldestInvoice if not specified.

***note:*** an optional memo regarding the payment made.

### *Output*

***return:*** an object of type Return that indicates the success or failure of the call.

***transaction:*** the Transaction object generated by the payment attempt. This Transaction must be inspected to assess the details of the payment attempt.

***summary:*** an object of type TransactionAttemptSummary, which describes the payment attempt: Success, Failure, or Pending.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• Account not found.<br>• Failed to translate payment method: ***error-description.***<br>• Failed to make payment: ***error-description.***<br>• Transaction not returned from payment attempt. |

## Example

```
$autobill = new AutoBill();
$autobill->setMerchantAutoBillId($abID);      // for some $abID

$paymentMethod = new PaymentMethod();
$paymentMethod->setMerchantPaymentMethodId($pmId);     // for some $pmId

$response = $autobill->makePayment(
    $paymentMethod,
    4.50,
    'USD',
    'inv-bac',
    null,
    '$4.50 for Scrabble'
);

// check $response
```

## migrate

The `migrate` method allows you to import existing subscription and transaction information from your current billing system to CashBox. This call will create new `AutoBills` which reflect the imported information.

`AutoBill.migrate` may be called multiple times for a given `AutoBill`. For the first call, CashBox will create the `AutoBill`, and build the billing schedule records that correspond to any `MigrationTransactions` that are included in the call.

If you call `AutoBill.migrate` on an existing `AutoBill` (i.e. if you specify a `VID` or `AutoBillId` of an AutoBill that already exists), CashBox will backfill the existing AutoBill's Transaction history, (import older Transactions for the subscription), and no attempt will be made to update the `AutoBill` itself.

> **Note:** While you may create new `Accounts` and `PaymentMethods` with this call, you may not create new `Products` or `BillingPlans`. Be certain that any `Products` or `BillingPlans` referenced by an input `MigrationTransaction` object are created before making the call.

### *Input*

***autobill:*** the `AutoBill` object to migrate to CashBox.

> **Note:** For this call, the following `AutoBill` data members are required:
>
> - `account`
> - `billingPlan`
> - `currency`
> - `items`
> - `paymentMethod` (For this call, this field must be `CreditCard`.)
> - `startTimestamp`

***nextPeriodStartDate:*** the next scheduled billing date for this AutoBill. If not provided, it will be assumed that this AutoBill is terminal and no future billings are to be scheduled. (For the first `AutoBill.migrate` call for a given `AutoBill`, this field is required.)

***migrationTransactions:*** an array of `MigrationTransaction` objects which define the history of this AutoBill.

The most-recent Transaction for this `AutoBill` must be included in the initial `AutoBill.migrate` call; and the latest status record must show that the `Transaction` is either `Captured`, `Cancelled`, `Refunded`, `Settled`, or `Void`. No other `TransactionStatus` values may be used for this call.

> **Note:** Any Transactions migrated to CashBox will follow your defined retry sequence. For example, an `AutoBill` migrated to CashBox with a single `Transaction` with status `Cancelled` will trigger a retry.

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobill:*** the `AutoBill` object that was created through migration.

If you specify a VID or `merchantAutoBillId` for ***autobill,*** and that ID does not yet exist in the CashBox database, this method will create a new `AutoBill` object. If the ID does exist, CashBox will update the corresponding `AutoBill` object.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `At least one migrationTransaction must be included in AutoBill.migrate request.`<br>• `Error validating parameters for AutoBill Migration.`<br>• `Migrated AutoBills must have at least one item.`<br>• `Product must be pre-defined for AutoBillItem` ***autoBillItem.***<br>• `Product not defined for AutoBillItem` ***autoBillItem.***<br>• `Invalid AutoBillItem addedDate` ***date***.<br>• `Invalid AutoBillItem removedDate:` ***date.***<br>• `BillingPlan must be pre-defined.`<br>• `BillingPlan with identifier` ***identifier*** `not found.`<br>• `Campaign code` ***campaignCode*** `is not valid.`<br>• `Invalid value for transitionedFromMerchantAutoBillItemId:` ***autoBillItemId***.<br>• `Invalid value for transitionedToMerchantAutoBillItemId:` ***autoBillItemId.***<br>• `Invalid value for nextPeriodStartDate.`<br>• `Transaction Migration attempt failed:` ***error***.<br>  • *(See the `Transaction.migrate` error list for details that could be appended to this message.)*<br>• `Migration Transactions not specified.`<br>• `PaymentMethod not specified.`<br>• `Unsupported Payment Type:` ***type***.<br>• `TransactionBilling migration details not provided for initialization of AutoBill.`<br>• `No BillingPlan: Cannot determine TransactionBillingSequence details.`<br>• `Billing Plan on last Transaction submitted does not match AutoBill BillingPlan.`<br>• `AutoBillCycle (transaction_billing_sequence) not defined.`<br>• `Unable to determine current AutoBill BillingPlanPeriod.`<br>• `Unable to determine Billing Period billing sequence.Failed to determine BillingPlanPeriod index for AutoBill Migration.`<br>• `BillingPlanCycle cannot be greater than autoBillCycle (transaction_billing_sequence).`<br>• `Currency code not defined in row` ***n*** `of Transaction Array for AutoBill Migration.` |

| Return Code | Return String |
|---|---|
| 400 *(continued)* | • Transaction currency (***currency***) does not match AutoBill currency (***currency***).<br>• Transaction for retry # ***n*** in billing sequence ***m*** already exists.<br>• Transaction migration failed for Transaction # ***n*** during Migration of AutoBill ***autoBill***.<br>  • *(See the `Transaction.migrate` error list for details that could be appended to this message.)*<br>• Mismatch found between AutoBill billing sequence (***n***), and the max TransactionBilling sequence (***m***).<br>• Transactions cannot be associated with an AutoBill's final TransactionBilling.<br>• TransactionBilling for sequence ***sequence*** is in invalid state (***state***) for AutoBill migration.<br>  • *(TransactionBillings generated by the migrate process should be in one of the following states: Success, Free, or Deferred.)*<br>• Original activity date for historical TransactionBilling sequence ***n*** is in the future (***date***).<br>• billing date for TransactionBilling (***date***) is greater than latest allowed (***date***).<br>• Transaction (***index***) in sequence ***sequence*** is in an invalid state (***state***) for AutoBill migration.<br>• *(The Transaction state (which is derived from the disposition log) must be one of the following: Captured, Cancelled, Refunded, Settled, or Void.)*<br>• NRC Transaction in sequence $sequence is in an invalid state (***state***) for AutoBill migration<br>  • *(Non-Recurring-Charge Transactions must be in one of the states specified for Recurring Transactions (above).)*<br>• Transaction autoBillCycle not defined.<br>• billingPlanCycle not defined.<br>• billingPeriodStart not defined.<br>• Failed to map AutoBill Items to Transaction Items during Transaction Migration.<br>• Cannot map Transaction Items to AutoBill Items - No Transaction Items!<br>• Cannot map Transaction Items to AutoBill Items - No AutoBill Items!<br>• Failed to determine Product SKU for AutoBill Item index ***n*** while attempting to map Transaction Items to AutoBill Items.<br>• Failed to map AutoBillItem ID ***autoBillItemId***, to a TransactionItem.<br>• Unable to determine AutoBill BillingPlanPeriod for Transaction ident ***transactionId.***<br>• Unable to determine Billing Period billing sequence for Transaction ident ***transactionId.***<br>• Failed to determine BillingPlanPeriod index associated with Transaction ident ***transactionId***. |

| Return Code | Return String |
|---|---|
| 400 *(Transaction migration error messages)* | • MigrationTransaction not provided.<br>• Invalid paymentProcessor: ***paymentProcessor***.<br>• MigrationTransaction must include at least one statusLog record.<br>• Failed to convert salesTaxAddress.<br>• Attempt to migrate Transaction which already exists.<br>• Unsupported Payment Type: ***paymentType***.<br>• Failed to prepare auth_response for Migrated Transactions.<br>• Unable to determine currency for migrated Transaction.<br>• Calculated Transaction amount (***XXX.XX***) does not match input amount (***YYY.YY***) on migrated Transaction. |

## *Example*

```
//To migrate an AutoBill for a pre-existing BillingPlan,
//Product, Account, and PaymentMethod.
//Note that it is possible to define new Account and
//PaymentMethod objects within AutoBill.migrate.
//The Product(s) and BillingPlan must, however, be pre-defined.

$billplanVid = 'c6743226ea41afd9db71c0c612a870bfcaa68fa7';
$productVid = '124a5540e359d59ba2a301a4b86cd5434f5c99d3';
$accountVid = '3915038987280cc31b103dbdf291cfd68181b385';
$paymentmethodVid = '822690237671dbb37014bb4c5262e0067ab94f97';
$addressVid = '3c14d744baa5cd618c1e0ff2c1f54b408e8c65e9';

$billPlan = new BillingPlan();
$product = new Product();
$account = new Account();
$autobill = new AutoBill();
$paymentMethod = new PaymentMethod();
$address = new Address();

$billPlan->setVid($billplanVid);
$product->setVid($productVid);
$account->setVid($accountVid);
$paymentMethod->setVid($paymentmethodVid);
$address->setVid($addressVid);

//Define the AutoBill to be Migrated
$item = new AutoBillItem();
$item->setIndex(0);
$item->setAddedDate('2014-01-06T10:14:14-08:00');
$item->setMerchantAutoBillItemId('merchantAutoBillItem1391710456');
$item->setProduct($product);

$autobill->setaccount($account);
$autobill->setItems(array($item));
$autobill->setBillingPlan($billPlan);
$autobill->currency('USD');
$autobill->setCustomerAutoBillName('MGRT_1391710573_cabn_1391710456');
$autobill->setMerchantAutoBillId('MGRT_1391710573_valg_1391710456');
```

```
$autobill->setPaymentMethod($paymentMethod);
$autobill->setStartTimestamp('2014-01-06T10:14:14-08:00');

//Next, define the MigrationTransaction to be included
//in the AutoBill.migrate call

$taxItemA = new MigrationTaxItem();
$taxItemA->setAmount(.92);
$taxItemA->setJurisdiction('COUNTY_19');
$taxItemA->setName('SALES TAX');

$taxItemB = new MigrationTaxItem();
$taxItemB->setAmount(6.67);
$taxItemB->setJurisdiction('DISTRICT');
$taxItemB->setName('CA DISTRICT SALES TAX');

$txItemA = new MigrationTransactionItem();
$txItemA->setItemType('RecurringCharge');
$txItemA->setMigrationTaxItems(array($taxItemA, $taxItemB));
$txItemA->setName('product 1391710450 default plan');
$txItemA->setPrice(49.99);
$txItemA->setServicePeriodStartDate('2014-01-06T00:00:00');
$txItemA->setServicePeriodEndDate('2014-03-05T00:00:00');
$txItemA->setSku('bp_1391710450');
$txItemA->setTaxClassification('DC010500');
   // This should be the Avalara tax code associated with this product

$txItemB = new MigrationTransactionItem();
$txItemB->setItemType('RecurringCharge');
$txItemB->setMerchantAutoBillItemId('merchantAutoBillItem1391710456');
$txItemB->setName('product_1391710450_1');
$txItemB->setPrice(42.00);
$txItemB->setServicePeriodStartDate('2014-01-06T00:00:00');
$txItemB->servicePeriodEndDate('2014-03-05T00:00:00');
$txItemB->setSku('1391710450_1');
$txItemB->setTaxClassification('DC010500');
   // This should be the Avalara tax code associated with this product

$creditCardStatusA = new CreditCardStatus();
$creditCardStatusA->setAuthCode('000');
$statusLogA = new TransactionStatus();
$statusLogA->setCreditCardStatus($creditCardStatusA);
$statusLogA->setPaymentMethodType('CreditCard');
$statulLogA->setStatus('Captured')
$statusLogA->setTimestamp('2014-02-06T10:16:06-08:00');

$creditCardStatusB = new CreditCardStatus();
$creditCardStatusB->setAuthCode('000');
$statusLogB = new TransactionStatus();
$statusLogB->setCreditCardStatus($creditCardStatusB);
$statusLogB->setPaymentMethodType('CreditCard');
$statusLogB->setStatus('New');
$statusLogB->setTimestamp('2014-02-06T10:14:51-08:00');
```

```
$migrationTransaction = new MigrationTransaction();
$migrationTransaction->setAmount(99.58);
$migrationTransaction->setAutoBillCycle(0);
$migrationTransaction->setBillingDate('2014-01-06T00:00:00');
$migrationTransaction->setBillingPlanCycle(0);
$migrationTransaction->setCurrency('USD');
$migrationTransaction->setDivisionNumber('iAmTheWalrus');
$migrationTransaction->setMerchantBillingPlanId('bp_1391710450');
$migrationTransaction->setMigrationTransactionItems
          (array($txItemA, $txItemB));
$migrationTransaction->setPaymentMethod($paymentMethod);
$migrationTransaction->setPaymentProcessor('Litle');
$migrationTransaction->setPaymentProcessorTransactionId('1069115');
$migrationTransaction->setRetryNumber(0);
$migrationTransaction->setSalesTaxAddress($address);
$migrationTransaction->setShippingAddress($address);
$migrationTransaction->setStatusLog(array($statusLogA, $statusLogB));
$migrationTransaction->setType(Recurring);

//Migrate AutoBill into CashBox
$response = $autobill->migrate('2014-03-06T00:00:00',
array($migrationTransaction));
if($response['returnCode'] == 200)
{
   //AutoBill and Transaction(s) migrated successfully
   print "AutoBill migrated with VID " .
   $response['data']->autobill->getVID() . "\n";
}
else
{
   //AutoBill migration failed
}
```

## modify

The `AutoBill.modify` call allows you to modify existing AutoBills, and generate any resultant charges or refunds to your customers, with a single call.

> **Note:** This call may **not** be used to create new AutoBills or Billing Plans. Both the AutoBill to modify, and any Products or Billing Plans used in modification must exist before making this call.
>
> This call may not be used with AutoBills using Rate Plan pricing, or with Seasonal Billing Plans.

`AutoBill.modify` allows you to:

- Modify `AutoBills` that have any number of `AutoBillItems`.
- Add, remove, or replace multiple `AutoBillItems`, in a single call.
- Work with Campaigns. (Added items may have a Campaign Code, and the AutoBill may have a `billingPlanCampaignCode`; the Campaign Code is redeemable on the Billing Plan only if a new Billing Plan is sent in.)
- Maintain the history of modified AutoBills.
- Generate a single, pro-rated net charge or refund for the combined modification activity. (This charge or refund will appear through the API and Portal with other Transactions from this AutoBill.)
- Retain the AutoBill in its original state if any aspect of the call, including the modification-based charge or refund, fails.

> **Note:** You must be integrated with the Avalara Tax system to use this call. Please work with your Avalara Support Representative to guarantee a cross-platform implementation.

### *Input*

***autobill:*** the `AutoBill` object to modify. Identity this object with its VID or `merchantAutoBillId`.

***billProratedPeriod:*** a Boolean flag which sets whether to prorate the price for the modification. If `true`, and ***effectiveDate*** is `today`, new items will be billed for the prorated remainder of the current Billing Period, and prorated credits will be issued for previously billed items. If false, your customer will not be billed for any changes to the AutoBill for the current Billing Period. Defaults to `false` if not specified.

Changing a billing plan with ***effectiveDate*** = `today` will always prorate regardless of the ***billProratedPeriod*** setting.

Transactions generated as a result of this option will automatically include the `vin:AutoBillVID` and the `vin:MerchantAutoBillIdentifier` name-value pairs.

***effectiveDate:*** indicates when the modification should become effective.

- `nextBill` begins changes with the next Billing Cycle.
- `today` begins changes effective ***today***.

  With ***effectiveDate*** `today`, the billing date (and billing day of month) will change ***only*** if the Billing Plan is changed (in which case, CashBox will perform the first billing for the new `BillingPlan` and `AutoBillItems` immediately). If only `AutoBillItems` are changed with this call, CashBox will prorate both credits and charges and bill/refund immediately; future recurring billings will remain scheduled as before.

***changeBillingPlanTo:*** the `BillingPlan` to replace the current on the `AutoBill`, identified by its `merchantBillingPlanId` or `VID`. (The new Billing Plan must already exist in your CashBox system before making this call.)

| **Note:** | Changing the Billing Plan will prorate any difference in cost between the original and the new Billing Plans, and reset the billing date for the AutoBill to ***today***. |
|---|---|

***autoBillItemModifications:*** an array of `AutoBillItemModification` objects, which define the changes to be made to the existing `AutoBill`.

`removeAutoBillItem`: an `AutoBillItem` to remove from the `AutoBill`. Identify this object with its `index`, `merchantAutoBillItemId`, `product`, or `VID`.

`addAutoBillItem`: an `AutoBillItem` to add to the `AutoBill`.

| **Note:** | The `transitionedFromMerchantAutoBillItemVid` and `transitionedToMerchantAutoBillItemId` data members will be populated ***only*** if a removed item is paired with an added item in a single `AutoBillItemModification` object. If submitted with separate objects, they will not be linked. |
|---|---|

***dryrun:*** a Boolean flag that, if set to `true`, will return the modified `AutoBill`, without recording the result in the CashBox database. Use this variable to compute the cost of an `AutoBill` modification without committing to the change.

| **Note:** | No payment method validations, authorizations or charges will be performed if ***dryrun*** is `true`. |
|---|---|

## *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobill:*** the `AutoBill` object that was modified.

***transaction:*** the `Transaction` object created (if ***billProratedPeriod*** was `true`) for the non-recurring charge or credit.

| **Note:** | Transactions generated as a result of this call will include a name-value pair with name `vin:type` and value `modify`. |
|---|---|

***refunds:*** an array of `Refund` objects created for the AutoBill as a result of the ***billProratedPeriod*** option.

## *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• Could not load the AutoBill.<br>• Invalid value for effectiveDate: ***value***.<br>• BillingPlan used in modify must already exist.<br>• No AutoBillItem found for ***criteria***.<br>• Could not determine a unique item for ***criteria***.<br>• The currency field on an added AutoBillItem does not match the currency on the AutoBill.<br>• Could not convert added item: ***error message***.<br>• Can only apply a billingPlanCampaignCode to a new BillingPlan.<br>• Cannot modify an AutoBill in Stopped\|Upgraded\|PendingCustomerAction state.<br>• AutoBill has ended and cannot be modified.<br>• PaymentMethod on AutoBill does not support recurring billing.<br>• Cannot create new BillingPlan as part of an AutoBill::modify().<br>• AutoBill cannot be modified for 'nextBill'; no further billing periods remain.<br>• Can only apply a campaign code to a new billing plan.<br>• Campaign code ***code*** could not be redeemed.<br>• Cannot redeem a time grant campaign on a BillingPlan via AutoBill::modify().<br>• BillingPlan ***merchantPlanId*** is not eligible for campaign code ***code***.<br>• Modifications would result in an AutoBill with no active AutoBillItems.<br>• Cannot remove items or change the BillingPlan effective today for an AutoBill with a Token PaymentMethod.<br>• Could not set AutoBill to 'Processing' status.<br>• Saving modified AutoBill failed; rolling back.<br>• AutoBillItem to be removed has already been removed.<br>• Could not determine start date for AutoBillItem.<br>• Campaign code ***code*** could not be redeemed.<br>• Cannot redeem a time grant campaign on a single AutoBillItem via AutoBill::modify().<br>• Product ***merchantProductId*** is not eligible for campaign code ***code***.<br>• Product ***merchantProductId*** is not eligible for an active time grant campaign. |
| 402 | • Modify transaction authorization failed. |

### *Example*

```
// This example will replace one item, add a new item,
// and change the billing plan effective today

$autoBillIdent = 'autobillToModify';
$annualPlanIdent = 'annualBillingPlan';

$replacementProductIdent = 'replacementProduct';
$newProductIdent = 'newProduct';

$oldAutoBillItemVID = 'd79ae3429ff102383b76d8f1eae8da52bd7dc1af';
$replacementItemIdent = 'upgradedProductItem';
$newItemIdent = 'addedProductItem';

// create and identify the AutoBill to be modified
$autobill = new AutoBill();
$autobill->merchantAutoBillId($autobillIdent);

// existing item must be uniquely identified
// (e.g., by VID, index, or by merchantAutoBillItemId or product,
// if these last two are unique within the given AutoBill)
$oldItem = new AutoBillItem();
$oldItem->VID($oldAutoBillItemVID);

// create the product objects to be added; they only need identifying
// information (e.g., a VID or merchantProductId)
$replacementProduct = new Product();
$replacementProduct->merchantProductId($replacementProductIdent);

$newProduct = new Product();
$newProduct->merchantProductId($newProductIdent);

// now build the new AutoBillItems
$replacementItem = new AutoBillItem();
$replacementItem->merchantAutoBillItemId($upgradedProductItem);
$replacementItem->product($replacementProduct);
$replacementItem->index(1);

$newItem = new AutoBillItem();
$newItem->merchantAutoBillItemId($newItemIdent);
$newItem->product($newProduct);
$newItem->index(2);

// and the AutoBillItemModification objects
$addModification = new AutoBillItemModification();
$addModification->addedAutoBillItem($newItem);

$replaceModification = new AutoBillItemModification();
$replaceModification->removeAutoBillItem($oldItem);
$replaceModification->addAutoBillItem($newItem);

// create the object for the new billing plan
$billingPlan = new BillingPlan();
$billingPlan->merchantBillingPlanId($annualPlanIdent);
```

```
// now perform the modification, effective today,
// prorating for any item or billing plan changes
$response = $autobill->modify('true', 'today', $billingPlan,
          [$addModification, $replaceModification], 'false');

if ($response['returnCode'] == 200)
{
   // return includes the updated autobill
   $updatedAutoBill = $response['data']->autobill();

   // return will include a transaction with all of the
   // details for the prorated amounts
   $transaction = $response['data']->transaction();

   // if the net value of all prorated changes was negative,
   // there will also be an array with one or more refunds
   // against the original transactions
   $refunds = $response['data']->refunds();
}
```

## redeemGiftCard

The `redeemGiftCard` method redeems a gift card represented by the input `GiftCard` object, and grants the resultant amount of credit to the `AutoBill`. This method should be called after the `statusInquiry()` method is called on the `GiftCard` object provided as input to this method. If the `statusInquiry()` method indicates that status of the `GiftCard` object is `Active`, you may call this method. For more information, see Section 9: The GiftCard Object.

For redemption of a gift card, CashBox contacts a gift card processor. (InComm is the only gift card processor with whom CashBox has a working relationship at this time.) If the gift card is redeemable, the processor returns an SKU or a UPC number. This number is unique for each type of gift card, and is defined by a prior agreement between you and the gift card processor. CashBox uses the number to look up a `Product` object with the same `merchantProductId`. CashBox then grants credit to the `AutoBill` as defined in the `creditGranted` attribute of the `Product` object. For each type of gift card you wish to accept, create (in advance) `Product` objects with the selected amount of credit specified in their `creditGranted` attributes.

CashBox currently supports only *full* redemption of the credit associated with a gift card.

See the `Credit` Subobject for more information on it and related subobjects. See Chapter 12: Credit Grants and Gift Cards in the ***CashBox Programming Guide***, for more information on gift card redemption.

### *Input*

***autobill:*** an `AutoBill` object to which you wish to grant credit, if redemption of the gift card is successful. Use the `merchantAutoBillId` or `VID` to identify the object.

***giftcard:*** a `GiftCard` object encapsulating information about the gift card you wish to redeem. For more information, see Section 9: The GiftCard Object. If you called the `statusInquiry()` method before calling this method, you should have the `VID` of the `GiftCard` object. Use the `VID` to identify the `GiftCard`.

***credit:*** *a `Credit` object specifying the amount and type of credit you wish to redeem. (This parameter is used with partial credit redemption, and is currently not in use.)*

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***giftcard:*** the `GiftCard` that was redeemed, with an updated `status`.

***autobill:*** the `AutoBill` input object to which CashBox grants credit if redemption of the input gift card is successful. This object contains the updated array of `Credit` objects.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• AutoBill not found.<br>• Failed to translate gift card ***error-description.***<br>• Failed to redeem gift card ***error-description.***<br>• Failed to retrieve gift card after redemption attempt.<br>• Failed to save AutoBill after gift card redemption attempt.<br>• Failed to reload AutoBill after gift card redemption attempt ***error-description.***<br>• Redemption attempt failed for Gift Card ID ***gift-card-details.*** |

## Example

```
$abill = new AutoBill();

// autobill id for an existing subscription to which the customer
// wishes to redeem a gift card and add credit to the autobill

$abill->setMerchantAutoBillId('SBCR312345');

$gc = new GiftCard();

    // set the VID of the gift card, obtained when we checked the
    // status of the gift card and determined that it was active

$gc->setVID($gcVID);

// Now make the SOAP API call to redeem the gift card

$response = $abill->redeemGiftCard($gc);

if ($response['returnCode'] == 200) {

    // Redemption successful. Check if credit was added to the autobill

    $updatedABill = $response['data']->autobill;

    $availableCredits = $updatedABill->getCredit();
    $availableTokens = $availableCredits->getTokenAmounts();

    print "Available token credits: \n";
    foreach($availableTokens as $tkAmt) {
        print "Token type: " . $tkAmt->getMerchantTokenId() . " ";
        print "Amount: " . $tkAmt->getAmount() . "\n";
    }

    // Also make sure status of the gift card is 'Redeemed'

    $updatedGc = $response['data']->giftcard;
    print "Status of the gift card: ";
    print $updatedGc->getStatus()->getStatus() . "\n";
}
else {

    // Error while granting credit to the account
    print $response['returnString'] . "\n";
}
```

## reversePayment

The `reversePayment` method allows you to reverse payments made using the `makePayment` method. This method may only be used against payments made using the `MerchantAcceptedPayment` payment method.

### Input

***autobill:*** the `AutoBill` to which this reversal applies.

***timestamp:*** the time that payment reversal occurs.

***paymentId:*** the `paymentId` of the Payment to be reversed. Either the `paymentId`, or the `invoiceId` (and optional `indexNumber`) must be specified.

> The `paymentId` is automatically set by CashBox when a payment is made to an Invoice, AutoBill, or Account. In reversing a payment, you must reference the appropriate `paymentId`.

***invoiceId:*** the ID of the Invoice associated with the payment reversal. Either the `paymentId`, or the `invoiceId` (and optional `indexNumber`) must be specified.

***indexNumber:*** the `indexNumber` of the payment item (on the `invoiceId` invoice) to be reversed.

***note:*** an optional memo regarding the payment reversal.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `AutoBill not found.`<br>• `Neither paymentId nor invoiceId:` ***indexNumber*** `provided for reversal attempt.`<br>• `Failed to add reverse payment:` ***error-description.*** |
| `404` | `Payment ID not found.` |

### Example

```
// to reverse a payment made using makePayment

$autobill = new AutoBill();
$autobill->setMerchantAutoBillId($abID);      // for some $abID

$paymentMethod = new PaymentMethod();
$paymentMethod->setMerchantPaymentMethodId($pmId); // for some $pmId
$paymentId = $paymentMethod->merchantAcceptedPayment->paymentId;

$response = $autobill->reversePayment(
    $now,
    $paymentId,
    undef,
    undef,
    'Changed my mind.'
);
// check $response
```

## revokeCredit

The `revokeCredit` method deducts credit from an `AutoBill` object. If the deduction results in a negative amount for a given type of credit, CashBox sets its balance to 0. This method returns the `AutoBill` object with resultant credit balance.

Specify the amount, type, and VID of the Credit you wish to revoke from the `Account` as a `Credit` object.

`Credit` and related subobjects are described with the `Credit` Subobject. See Chapter 12: Credit Grants and Gift Cards in the ***CashBox Programming Guide*** for more information on working with credit.

### *Input*

***autobill:*** the `AutoBill` object from which you wish to revoke credit. Use the `merchantAutoBillId` or `VID` to identify the object.

***credit:*** a `Credit` object specifying the amount and type of credit you wish to deduct from the `AutoBill`.

***note:*** an optional memo regarding the credit revocation.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobill:*** the `AutoBill` object from which you revoked credit. This object contains the updated array of `Credit` objects.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `AutoBill not found.`<br>• `Failed to translate credit` ***error-description.***<br>• `Failed to revoke credit` ***error-description.***<br>• `Failed to save AutoBill after revoking credit.`<br>• `Failed to reload AutoBill after revoking credit` ***error-description.***<br>• `Data validation error: Missing required parameter` ***credit.*** |

### *Example*

```
// to revoke credit from an autobill

$abill = new AutoBill();

// autobill id for customer's existing subscription
// to a game

$abill->setMerchantAutoBillId('STARWARS-239181');

$tok = new Token();

// specify id of an existing token type.
// the autobill has a payment method defined in terms
// of this token type. Also the billing plan used by
// the autobill specifies a price in terms of this token
// type.

$tok->setMerchantTokenId('STARWARS_POINTS');

$tokAmt = new TokenAmount();
$tokAmt->setToken($tok);
$tokAmt->setAmount(100); // customer lost 100 points in the game

$cr = new Credit();
$cr->setTokenAmounts(array($tokAmt));

// Now make the SOAP API call to deduct points from customer's
// subscription

$response = $abill->revokeCredit($cr);

if ($response['returnCode'] == 200) {

    // Credit successfully revoked from the autobill

    $updatedAbill = $response['data']->autobill;
    $availableCredits = $updatedAbill->getCredit();
    $availableTokens = $availableCredits->getTokenAmounts();

    print "Available points to subscription: \n";
    foreach($availableTokens as $tkAmt) {
        print "Token type: " . $tkAmt->getMerchantTokenId() . " ";
        print "Amount: " . $tkAmt->getAmount() . "\n";
    }
}
else {

    // Error while revoking credit from the autobill
    print $response['returnString'] . "\n";
}
```

## update

The update method is used to create a new AutoBill object. An AutoBill object represents a customer subscription with recurring billing. Be certain to properly construct the AutoBill (for AutoBill attributes see Section 4.1: AutoBill Data Members) before passing it into this call.

| | |
|---|---|
| **Note:** | While this method may be used to alter an existing AutoBill, it is not recommended. Use upgrade to change an existing AutoBill. |

Call this method to first risk-screen the related Payment Method for the AutoBill object. Once you have enabled risk screening, this method scores a Transaction for the related Payment Method. (For more information, see the score method.) If the score is below the acceptable threshold specified in the minChargebackProbability parameter, the AutoBill object will be created. If the score is equal to or greater than the threshold, the object will not be created. For scoring to succeed, you must specify, in the AutoBill object, the source IP address from which the customer requested this subscription, and, in the associated payment method, the billing address.

To have CashBox contact your payment processor to validate the payment method, set the ***validatePaymentMethod*** flag to true.

To create an AutoBill object, initialize the object and set the values for its data members as appropriate, and then call the update() method to create the object on Vindicia's servers. When creating a new AutoBill object, do not set a value for VID; CashBox automatically generates the VID when you call update(). When updating an existing AutoBill object, identify it with its VID or your AutoBill ID (merchantAutoBillId).

Products may have an array of Products beneath them ("sub-Products"). When a sub-Product is placed on an AutoBill, all attributes of the top-level Product will apply to the AutoBill, except the Entitlements, which will be the union of the Entitlements of all of the Products and sub-Products.

| | |
|---|---|
| **Note:** | The AutoBill.update method may not be used to add Products to an AutoBill. To add Products, use AutoBill.addProduct. |

To apply a Campaign discount to an AutoBill, the Billing Plan must have prices defined in a currency which matches the AutoBill's. If the Billing Plan price is defined in currencies which differ from those used for the AutoBill, the discount will not be applied.

| | |
|---|---|
| **Note:** | The customer's Account must exist before any Hosted Page related call references that Account. |

## Input

***autobill:*** the `AutoBill` object to create or update. Identity this object with its VID or `merchantAutoBillId`.

***duplicateBehavior*** *is a placeholder only, and is currently not in use.*

***validatePaymentMethod:*** a Boolean flag that, if set to `true`, causes this method to validate the payment method for the `AutoBill` object. The nature of validation depends on the type of the payment method. For example, for some payment processors that do not support validation calls, CashBox validates credit cards by authorizing a transaction that uses the ***validatePaymentMethod*** method to pay for a very small amount, such as $1.00.

When ***validatePaymentMethod*** is `true`, the AVS and CVN policies (or, in their absence, the default evaluation policy) are used to determine the status of the validation. If validation fails, the `PaymentMethod` is not updated.

**Note:** The `AutoBill` will not be saved if validation is requested and fails.

The evaluation policy results are mapped to the AutoBill and Entitlement creation as follows:

Table 4-8 AVS / CVN Policy Evaluation Results

| Policy Evaluation | Result |
|---|---|
| Success | AutoBill is active and entitlement is granted. |
| Pending | AutoBill is pending and entitlement is granted. |
| Fail | AutoBill is cancelled and entitlement is inactive. |

CashBox also supports a configuration parameter that enables you to fully bill the first rebill transaction for this `AutoBill` object, in order to validate the payment method. Consult your Vindicia Client Services representative if you wish to use this configuration parameter.

For more detail on AVS and CVN Return Codes, please work with your Vindicia Client Services representative.

***minChargebackProbability:*** a number between 0 and 100 by which you specify your fraud risk score tolerance level. A chargeback probability (also called the risk-screening score or risk score) of 100 indicates that CashBox is 100% certain that a transaction is fraudulent and will result in a chargeback. Specify your acceptable threshold for chargeback possibility with this parameter. If the score evaluates to be more than your tolerance level, the `update` call will fail.

If you do not set ***minChargebackProbability***, it defaults to 100, meaning that all transactions are acceptable and that no risk screening occurs. For more information on CashBox risk-screening, see Section 14: Common ChargeGuard Programming Tasks in the ***CashBox Programming Guide***.

***ignoreAvsPolicy:*** a Boolean flag that, if set to `true`, will override the AVS policy, and update the `paymentMethod`, regardless of the AVS return code. If set to `false` or `null`, (and if `validatePaymentMethod` is set to `true`) the AVS return code will be used to determine whether to update the `paymentMethod`.

***ignoreCvnPolicy:*** an optional Boolean flag that, if set to `true`, will override the CVN policy, and update the `paymentMethod`, regardless of the CVN return code. If set to `false` or `null`, (and if ***validatePaymentMethod*** is set to `true`) the CVN return code will be used to determine whether to update the `paymentMethod`.

***campaignCode:*** an optional Coupon or Promotion code, used in conjunction with a Campaign, to obtain a discount on this AutoBill.

***dryrun:*** a Boolean flag that, if set to `true`, will return the updated `AutoBill`, without recording the result in the CashBox database. Use this method to compute the cost of an `AutoBill` without committing to the change. (The projected billing amount will be returned in the `Transaction` object of the `nextBilling` data member of the returned `AutoBill` object.)

If the `AutoBill` did not exist before, it will not exist afterward; if it did exist before, it will not change. (No payment method validations, authorizations or charges will be performed if ***dryrun*** is `true`.)

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***autobill:*** the `AutoBill` object that was created or updated. If you specify a VID or your `AutoBill` ID for ***autobill,*** but that ID does not exist in the CashBox database, this method creates a new `AutoBill` object. Otherwise, CashBox updates the `AutoBill` object whose ID matches the input.

***created:*** a Boolean flag that, if set to `true`, indicates that this method has created a new `AutoBill` object. A `false` setting indicates that `update` or `upgrade` has updated an existing `AutoBill` object.

***authStatus:*** if ***validatePaymentMethod*** is set to `true`, `authStatus` contains the response from the payment processor. For example, the Address Verification Service (AVS) and Card Verification Number (CVN) response codes.

***firstBillDate:*** the date of the first billing.

***firstBillAmount:*** the amount of the first billing.

***firstBillingCurrency:*** the currency of the first billing.

***score:*** the risk score for the payment method used for the `AutoBill` if you enabled risk scoring by specifying the value of the input parameter `minChargebackProbability` to be less than 100.

Normally, this value is between 0 and 100, where 100 is the highest risk score, indicating maximum chargeback probability. A value of -1 indicates that CashBox could not evaluate the score because of missing data such as an IP address or a full billing address. A value of -2 indicates an error condition.

***scoreCodes:*** an array of `ScoreCode` objects, each of which includes a code and corresponding message explaining why the risk score evaluated to a certain value.

## *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• Failed to translate credit ***error-description.***<br>• Unable to create AutoBill: ***error-description.***<br>• Data validation error: Failed to create Payment-Type-Specific Payment Record: Credit Card conversion failed: Credit Card failed Luhn check.<br>• Unable to create autobill: Must specify product to create autobill with!<br>• Campaign code XYZ is not usable: Code XYZ is not valid.<br>• No eligible, undiscounted items found for campaign code. |
| 402 | Unable to create AutoBill: ***error-description.***<br>(This return code means that validation failed.) |
| 403 | Cannot update an AutoBill that has completed the retry cycle, and is past its endTimestamp. |
| 407 | AVS policy evaluation failed. |
| 408 | CVN policy evaluation failed. |
| 409 | AVS and CVN policy evaluations failed. |
| 410 | AVS and CVN policy evaluations could not be performed. |

## Example

```
// To create a subscription, to an existing product, for an
// existing customer, using an existing billing plan

$autobill = new AutoBill();
$account = new Account();
$product = new Product();
$billPlan = new BillingPlan();

// Identify a previously created product by your unique ID
$product->setMerchantProductId('12345');

// Identify a previously created billing plan by your unique ID
$billPlan->setMerchantBillingPlanId('bp12345');

// Identify a previously created account by your unique ID
// Assumption: Account already has a payment method attached to it
// which will be used by the AutoBill automatically

$account->setMerchantAccountId('acct12345');

$autobill->setAccount($account);

// AutoBills may have multiple products
// each in an AutoBillItem as an array:

$item = new AutoBillItem();
$item->setIndex(0);


// set the Product in the AutoBillItem
$item->setProduct($product);

// set the Product (AutoBillItem)
$response = $autobill->setItems(array($item));

$autobill->setBillingPlan($billPlan);
$autobill->setMerchantAutoBillId('ab-44822'); // your ID for the AutoBill
$autobill->setCurrency('USD');

$validate = true;
$fraudScore = 100 ; // do not want to do risk screening

$response = $autobill->update('SucceedIgnore',
       $validate, $fraudScore, true, true);

if($response['returnCode'] == 200 && $response['created'] ) {
   print "AutoBill created with VID "
          . $response['data']->autobill->getVID() . "\n";
   if ( $response['authStatus'] != null ) {
       $txnStatus = $response['authStatus'];
       log (" CVN return code: "
           . $txnStatus->getCreditCardStatus()->getCvnCode()
           . "AVS return code"
           . $txnStatus->getCreditCardStatus()->getAvsCode() . "\n");
   }
}
```

## writeOffInvoice

Marks an `Invoice` as `writtenOff`, the debt unable to be collected.

### *Input*

***autobill:*** the `AutoBill` associated with the Invoice to be written off.

***invoiceId:*** the ID of the Invoice to write off.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `AutoBill not found.`<br>• `Invoiced ID not provided.`<br>• `Failed to write off invoice:` ***error-description.*** |
| `404` | `Invoice ID not found.` |

### *Example*

```
$autobill = new AutoBill();
$autobill->setMerchantAutoBillId($abID);      // for some $abID

$response = $autobill->writeOffInvoice('inv-bac');

// check $response
```

# 5 The `BillingPlan` Object

The periodicity, frequency, and amount of rebilling transactions generated by an `AutoBill` object are determined by its associated `BillingPlan` object, which enables you to define a billing model for a service or product for subscription. The `BillingPlan` object also allows you to define complex pricing logic, that employs variably priced billing periods. For example, a Billing Plan may include an initial, two-month free period, followed by an introductory price of $19.99/month for six months, and then by the regular price of $44.99/month for an indeterminate time. (Billing Plans may be defined in any currency that CashBox supports.)

Billing Plans may also be used to customize entitlement access in relation to the Billing Plan. For example, entitlements may be coincident with the plan (a customer is granted access for every month on which they make a payment), or entitlements may be separated from the billing sequence (a customer may be granted access for a year, as a result of only three monthly payments).

Create new Billing Plans when you launch or update a subscription service. Billing Plans may be created using the CashBox API, or the CashBox Portal.

# 5.1 BillingPlan Data Members

The following tables list and describe the data members of the BillingPlan object and its subobjects.

Table 5-1 BillingPlan Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| billingStatementIdentifier | string | The transaction description on the customer's billing statement from the bank when the customer is charged through this BillingPlan object. This field's value and its format are constrained by your payment processor; consult with Vindicia Client Services before setting the value.<br>If GlobalCollect, MeS, Chase Paymentech or Litle is your payment processor, see Appendix A: Custom Billing Statement Identifier Requirements in the ***CashBox Programming Guide***. |
| daysBeforeSeasonToBill | int | If the Billing Period set repeats for multiple Seasons, this value defines the number of days before the Season begins that the Account should be billed. (Default is 0.) |
| DaysEntitledAfterSeason | int | Defines the number of days after a Season ends that Entitlements will remain Active. |
| daysEntitledBeforeSeason | int | Defines the number of days before the Season begins that Entitlements will become Active. |
| description | string | Your description of the Billing Plan. |
| endOfLifeTimestamp | dateTime | **Optional.** A timestamp that specifies the expiration date for this BillingPlan object. This value is for your information only, and does not affect CashBox operations. |
| entitledOffSeason | Boolean | A Boolean flag that, if set to true, sets Entitlements to remain Active in the off-season. (Default is false.) |
| entitlementsValidFor | string | The length of time for which Entitlements are valid after the last Billing date. |
| merchantBillingPlanId | string | Your unique identifier for this BillingPlan object. This value enables you to look up a BillingPlan object with the fetchByMerchantBillingPlanId method. Reference the plan with this ID when making a call that requires you to specify a billing plan. |
| merchantEntitlementIds | MerchantEntitlementId[] | An array of identifiers, specified by you, to define the customer's entitlements. These IDs have special meaning in your application. For example, your application might contain logic such that the Gold Access ID enables a customer to access certain special features of your service.<br>CashBox returns these IDs to you inside Entitlement objects along with the dates till which they are considered valid for a given customer. |

Table 5-1 `BillingPlan` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `nameValues` | `NameValuePair[]` | **Optional.** An array of name–value pairs, each of which enables you to include `BillingPlan` information other than that in the `description` field.<br>See Section 10: The NameValuePair Object. |
| `periods` | `BillingPlanPeriod[]` | An array of items that describe the billing. The `AutoBill` object uses this array sequentially for actual billing transactions, enabling the creation of complex billing plans in numerous currencies, for example, one free month, followed by three months at $9.99/month, and then 12 months for $15.99/month.<br>For example, a Billing Plan may define a free trial period, followed by a monthly subscription service; or it may define a Seasonal Billing Plan, whereby a customer is billed only during the Season to which they are subscribed.<br>See the `BillingPlanPeriod` Subobject. |
| `prenotifyDays` | `int` | **Optional.** The number of days before an `AutoBill` object's billing date to notify yourself or the customer of an impending billing. |
| `repeatEvery` | string | If a Billing Period set repeats, this value defines the length of time after the first billing that the set should repeat. |
| `seasonSet` | `SeasonSet` | The `SeasonSet` to which the Billing Plan applies. (May be `null`.)<br>See Section 16.1: SeasonSet Data Members. |
| `skipInitialFreeWhenRepeating` | Boolean | A Boolean flag that, if set to `true`, excludes initial free periods when repeating a Billing Period set. (Default is `true`.) |
| `status` | `BillingPlanStatus` | An enumerated string value that describes the current state of the `BillingPlan` object. This value is for your information only, and does not affect CashBox operations.<br>See the `BillingPlanStatus` Subobject. |
| `timesToRun` | string | The number of times the sequence of Billing Periods should be repeated. Valid input includes positive integers, or "unlimited." (Default is `null`.) |
| `VID` | `string` | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `BillingPlan` object, leave this field blank; it will be automatically populated by CashBox. |

# 5.2 BillingPlan Subobjects

The BillingPlan object has several subobjects:

- BillingPlanPeriod Subobject
- BillingPlanPeriodType Subobject
- BillingPlanPrice Subobject
- BillingPlanStatus Subobject

## BillingPlanPeriod Subobject

Describes a quantity of time and a set of prices to use for the BillingPlan.

Table 5-2 BillingPlanPeriod Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| cycles | int | The number of billing cycles that pertain to this billing period. Set the value to 0 to specify an infinite number of billing cycles; set it to 3 to use this billing-plan period three times in succession. |
| doNotNotify-FirstBill | Boolean | A Boolean flag that, if set to true, prevents the prenotification email message from being sent. Use this flag to prevent email notification for the first bill after a free trial, for which an expiration warning message has already been sent. |
| expireWarning-Days | int | The number of days before the expiration of this billing period to send a warning by email. CashBox sends the warning *X* number of days before the expiration date, where *X* is the value specified in this attribute. |
| free | Boolean | A Boolean flag that, if set to true, guarantees that CashBox will not bill for the AutoBill's Products, regardless of whether they are added or included. CashBox *will* bill for Charges, which may have been explicitly added during the period.<br>Note that setting this flag to true causes CashBox to ignore any price defined elsewhere for the Billing Period, and set the period to free. |
| prices | BillingPlan-Price[] | The price of this billing period, in a specific currency or token type, but not both. The actual price for the transactions generated for the associated AutoBill object depends on the price picked from this array that matches the currency on AutoBill.<br>See the BillingPlanPrice Subobject. |

Table 5-2 `BillingPlanPeriod` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `quantity` | int | The number of units of the billing period `type` to count as a single billing period. For example, for a bi-weekly billing cycle, set this value to `2` and `BillingPlanPeriodType` to `Week`. (Default is 1.) |
| `type` | `BillingPlanPeriodType` | An enumerated string that specifies the unit (day, week, month, or year) for the duration of the billing period.<br>See the `BillingPlanPeriodType` Subobject. |

## `BillingPlanPeriodType` Subobject

The unit of time the Period describes.

Table 5-3 `BillingPlanPeriodType` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `Day` | string | The billing period is by day. |
| `Week` | string | The billing period is by week. |
| `Month` | string | The billing period is by month. |
| `Year` | string | The billing period is by year. |

## `BillingPlanPrice` Subobject

A price for the `BillingPlan`.

Table 5-4 `BillingPlanPrice` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `amount` | decimal | The amount to bill. Must be zero (for free trials) or positive. |
| `currency` | string | The ISO 4217 currency code (see www.xe.com/iso4217.htm) of billing. The default is USD. |
| `priceListName` | string | **Optional**. The name of the price list that contains this price. This is a free-form string of a maximum of 255 characters that describes this price point. |
| `tokenAmount` | `TokenAmount` | The price of this billing plan period, expressed in terms of the number of tokens of a certain type. CashBox decrements this amount from the `Account` object when billing it for this period. |

## `BillingPlanStatus` Subobject

Describes whether the `BillingPlan` is `Active` or `Suspended`. Suspended Billing Plans may not be AutoBill renewed.

Table 5-5 `BillingPlanStatus` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `Active` | string | The `BillingPlan` object is active (accessible to the customer). |
| `Suspended` | string | The `BillingPlan` object is inactive (inaccessible to the customer). |

## 5.3 BillingPlan Methods

The following table summarizes the methods for the BillingPlan object.

Table 5-6 BillingPlan Object Methods

| Method | Description |
|---|---|
| fetchAll | Returns all the BillingPlan objects. |
| fetchAllInSeason | Returns all in season Billing Plans. |
| fetchAllOffSeason | Returns all off-season Billing Plans. |
| fetchByBillingPlanStatus | Returns one or more BillingPlan objects whose status matches the input (Active or Suspended). |
| fetchByMerchantBillingPlanId | Returns a BillingPlan object whose ID assigned by you matches the input. |
| fetchByMerchantEntitlementId | Returns one or more BillingPlan objects whose entitlement ID assigned by you (merchantEntitlementId) matches the input. |
| fetchByVid | Returns a BillingPlan object whose VID matches the input. |
| update | Creates or updates a BillingPlan object. |

## fetchAll

The `fetchAll` method returns all your `BillingPlan` objects.

This method supports paging to limit the number of records returned per call. Returning a large number of records in one call may swamp buffers, and might cause a failure. Vindicia recommends that you call this method in a loop, incrementing the page for each loop iteration with an optimal page size (number of records returned in one call) until the page contains a number of records that is less than the given page size.

### Input

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize***: the number of records to display per page per call. This value must be greater than 0.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***billingPlans:*** an array of returned `BillingPlan` objects.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 404 | No BillingPlans found for merchant. |

### Example

```
$bp = new BillingPlan();
$page = 0;
$pageSize = 10;
do {
    $ret = $bp->fetchAll($page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedPlans = $ret['billingPlans'];
        $count = sizeof($fetchedPlans);
        foreach ($fetchedPlans as $plan) {

            // process a fetched plan here …
        }
    $page++;
    }
} while ($count > 0);
```

## fetchAllInSeason

The `fetchAllInSeason` method returns all `BillingPlan` objects with in season `SeasonSets`.

### Input

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize***: the number of records to display per page per call. This value must be greater than 0.

***nowDate:*** the (optional) date to query. (Defaults to ***today***.)

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***billingPlans:*** an array of returned `BillingPlan` objects.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$bp = new BillingPlan();
$page = 0;
$pageSize = 10;
do {
    $ret = $bp->fetchAllInSeason($page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedPlans = $ret['billingPlans'];
        $count = sizeof($fetchedPlans);
        foreach ($fetchedPlans as $plan) {

            // process a fetched plan here …
        }
    $page++;
    }
} while ($count > 0);
```

## fetchAllOffSeason

The `fetchAllOffSeason` method returns all `BillingPlan` objects with off-season `SeasonSets`.

### Input

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize***: the number of records to display per page per call. This value must be greater than 0.

***nowDate:*** the (optional) date to query. (Defaults to ***today***.)

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***billingPlans:*** an array of returned `BillingPlan` objects.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$bp = new BillingPlan();
$page = 0;
$pageSize = 10;
do {
    $ret = $bp->fetchAllOffSeason($page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedPlans = $ret['billingPlans'];
        $count = sizeof($fetchedPlans);
        foreach ($fetchedPlans as $plan) {

            // process a fetched plan here …
        }
    $page++;
    }
} while ($count > 0);
```

## fetchByBillingPlanStatus

The `fetchByBillingPlanStatus` method returns one or more `BillingPlan` objects whose status matches the input (either `Active` or `Suspended`). For example, call this method to retrieve all active billing plans, and present them to a customer as subscription choices.

### Input

***status:*** a string that describes the `BillingPlan` status (either `Active` or `Suspended`), which serves as the search criterion.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***billingPlans:*** an array of one or more `BillingPlan` objects whose status matches the input.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
// Create an array of billing plan object
$plan = new BillingPlan();

// now load all billing plans that have a status of Suspended
$response = $plan->fetchByBillingPlanStatus('Active');
if($response['returnCode'] == 200) {
    $fetchedPlans = $response['data']->billingPlans;

    if ($fetchedPlans != null ) {
        foreach ($fetchedPlans as $billPlan) {
            // process a fetched plan here
        }
    }
}
```

## fetchByMerchantBillingPlanId

The `fetchByMerchantBillingPlanId` method returns a `BillingPlan` object whose ID, assigned by you, matches the input.

### Input

***merchantBillingPlanId:*** your billing plan ID (`merchantBillingPlanId`), which serves as the search criterion.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***billingPlan:*** the `BillingPlan` object whose `merchantBillingPlanId` matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 404 | No BillingPlans found for Merchant Billing Plan ID ***input-merchantBillingPlanId.*** |

### Example

```
$bpMerchantId = '12345';

// Create a billing plan object
$plan = new BillingPlan();

// now load a billing plan record into the Billing Plan object
$response = $plan->fetchByMerchantBillingPlanId($bpMerchantId);
if($response['returnCode'] == 200) {
    $fetchedBillingPlan = $response['data']->billingPlan;

    // process fetched billing plan here
}
```

## fetchByMerchantEntitlementId

The `fetchByMerchantEntitlementId` method returns one or more `BillingPlan` objects that offer an entitlement whose ID matches the input. For example, call this method if a customer would like to see all the billing plans which grant a specific privilege on your site.

### Input

***merchantEntitlementId:*** the merchant's unique ID for the `Entitlement`.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***billingPlans:*** an array of one or more `BillingPlan` objects with an entitlement whose ID matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 404 | No BillingPlans found for entitlementId ***input-merchantEntitlementId.*** |

### Example

```
$plan = new BillingPlan();

    // now load all billing plans that have an
    // entitlement id of download $meId = 'Gold Access';
    // This is the id we want to retrieve plans by

$response = $plan->fetchByMerchantEntitlementId($meId);

if($response['returnCode'] == 200) {

    $fetchedPlans = $response['data']->billingPlans;

    if ($fetchedPlans != null ) {
        foreach ($fetchedPlans as $billPlan) {
            // process a fetched plan here
        }
    }
}
```

## fetchByVid

The `fetchByVid` method returns a `BillingPlan` object whose VID matches the input.

When you first create a `BillingPlan` object with the `update()` method, leave the `VID` field empty; CashBox automatically assigns the object a unique VID inside the `BillingPlan` object that you receive in response to the call. Use this VID to retrieve the object later.

### *Input*

***vid:*** the `BillingPlan` object's Vindicia identifier, which serves as the search criterion.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***billingPlan:*** the `BillingPlan` object whose VID matches the input.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 404 | No BillingPlans found for VID ***input-vid.*** |

### *Example*

```
$planVid = 'MyVindiciaVID';

// Create a billing plan object
$plan = new BillingPlan();
// now load a billing plan record into the BillingPlan object by VID
$response = $plan->fetchByVid($planVid);

if($response['returnCode'] == 200) {
    $fetchedPlan = $response['data']->billingPlan;

    // process fetched billing plan here
}
```

## update

The `update` method creates a new `BillingPlan` object (that is, a new billing plan), or updates an existing `BillingPlan` object.

Billing Plans may be created using either the CashBox API, or the CashBox Portal. Use the `BillingPlan.update` method of the API to create or update a large number of billing plans.

To create a `BillingPlan` object, initialize the object, set the values for its data members as appropriate, and then call the `update()` method to store the changes. When creating a new `BillingPlan` object, do not set a value for `VID`; CashBox will automatically generate a VID for the object when you call `update()`.

Set the `BillingPlanPeriod` object's `free` flag to `true` to override any price setting for Products included in the AutoBill. If set to `true`, no Product price will be applied to the AutoBill; only Charges.

> **Note:** Setting this `BillingPlanPeriod` flag to `true` causes CashBox to ignore any price defined elsewhere for the Billing Period, and set the period to free.

When updating an existing `BillingPlan` object, identify it with its VID or your billing plan ID (`merchantBillingPlanId`). Be certain to add billing plan periods and prices in the appropriate currencies.

> **Note:** Changing the pricing structure for a Billing Plan will change the price for any active AutoBills associated with the plan. If your customer has already received a pre-billing notification before you change the Billing Plan's price, but before they are billed, they will be charged the old price for that Billing Cycle. If they have not received a pre-billing notification, the new Billing Plan price will take effect upon the next Billing Cycle.

### *Input*

***billingPlan:*** the `BillingPlan` object to be created or updated. If you are updating an existing plan, identify this object with its VID or your billing plan ID (`merchantBillingPlanId`).

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***billingPlan:*** the `BillingPlan` object that was created or updated.

***created:*** a Boolean flag that, if set to `true`, indicates that this method has created a new `BillingPlan` object. A `false` setting indicates that update has updated an existing `BillingPlan` object.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

## *Example*

```
// to create a billing plan

// Create a new billing plan
$plan = new BillingPlan();

   // Identify the billing plan by your unique identifier, etc.
$plan->setMerchantBillingPlanId('12345');
$plan->setPreNotifyDays(7);
$plan->setStatus('Active');
$plan->description('1 Free Month then 2 Months at $5.00 (USD),
          $5.60 (CAD) then $120.00(USD), $135.00(CAD) per year');
$plan->periods[0]=(new BillingPlanPeriod(type => 'Month',
          quantity => 1,
          cycles => 1, //Just once
          prices => [new BillingPlanPrices('amount' => 0,
          'currency' => 'USD'),
          new BillingPlanPrices('amount' => 0, 'currency' => 'CAD')]));
$plan->periods[1]=(new BillingPlanPeriod(type => 'Month',
          quantity => 1,
          cycles => 2, //for 2 months
          prices => [new BillingPlanPrices
          ('amount' => 5.00, 'currency' => 'USD'),
          new BillingPlanPrices('amount' => 5.60, 'currency' => 'CAD')]));
$plan->periods[2]=(new BillingPlanPeriod(
          type => 'Year',
          quantity => 1,
          cycles => 0, //Repeat infinitely
          prices => [new BillingPlanPrices('amount' => 120.00,
          'currency' => 'USD'),
          new BillingPlanPrices('amount' => 135.00, 'currency' => 'CAD')]));

$response = $plan->update();
if($response['returnCode'] == 200 && $response['created'])
{
   print "Billing plan successfully created. VID: "
          . $response['data']->billingPlan->getVID() . "\n";
}
```

# 6 The `Campaign` Object

CashBox Campaigns allow you to offer special discounts on your existing products. Campaigns are discounts given over a period of time for a service or subscription, and may be applied to multiple Products, and multiple Billing Cycles.

***Promotion Campaigns*** generate a single Campaign Code, which may be distributed to multiple customers.

***Coupon Campaigns*** generate multiple unique Campaign Codes, which may be used a defined number of times. Coupon Campaigns are often highly targeted, and Coupon Code distribution and redemption may be tracked.

The CashBox Portal offers a single page from which you may create Campaigns, from selecting the product, pricing change, and time frame, to defining the Campaign description and Coupon or Promotion code.

Once a Campaign is underway, that is, once a Campaign has been activated, and Promotions or Coupons have been redeemed, you may not change any Campaign parameters that define the discount. To change parameters, such as `flatAmountDiscount`, or the number of weeks in a Rolling Campaign, you must cancel the Campaign, and deactivate any existing Coupon or Promotion Codes.

For more information on Campaigns, see Chapter 10: Campaigns in the ***CashBox Users Guide***.

# 6.1 Campaign Data Members

The Campaign object encapsulates the information for a Campaign, including Campaign Type, Status, and Coupon Codes, if applicable.

The following table lists and describes the data members of the Campaign object.

Table 6-1 Campaign Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| campaignId | string | Your unique identifier for this Campaign object.<br>**Note:** Read-only once the Campaign has been created. |
| campaignType | CampaignType | Specifies whether the Campaign is a Coupon or Promotion. Valid CampaignTypes include:<br>• Undefined (Used only for errors.)<br>• Coupon<br>• Promotion<br>**Note:** Read-only once the Campaign is Active. |
| couponCodePrefix | string | Defines a prefix for the CashBox randomly generated Coupon Code. |
| couponCodeQuantity | integer | The number of Coupon Codes to generate. |
| couponCodeRequiresActivation | Boolean | A Boolean flag which, if set to true, creates inactive Coupon Codes. (Inactive Coupon Codes must be individually activated before use.) (Default is false.) |
| couponCodeSeparator | string | The (optional) character used to separate the couponCodePrefix from the CouponCode string. This may be any printable, non-alphanumeric ASCII character. |
| cycles | integer | The number of Billing Cycles to which the Campaign discount will be applied.<br>**Note:** A Campaign must include either the cycles or the expirationDate data member, and it may not include both. |
| description | string | Description of the Campaign.<br>**Note:** Read-only once the Campaign is Active. |
| eligibleProduct | Product | One or more Products eligible for this Campaign.<br>See Section 13.1: Product Data Members. |
| expirationDate | dateTime | The date the Campaign discount expires. (If null, the offerEndDate will be used.)<br>This date may be after the Campaign's offerEndDate, but cannot be before it.<br>**Note:** A Campaign must include either the cycles or the expirationDate data member, and it may not include both. |

Table 6-1 `Campaign` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `flatAmountDiscount` | `CurrencyAmount` | Defines the discount, or discounts, as a `CurrencyAmount` pair object.<br>**Note:** `flatAmountDiscount` and `percentageDiscount` are mutually exclusive.<br>**Note:** A `Campaign` must include either the `flatAmountDiscount` or the `percentageDiscount` data member, and it may not include both. |
| `maxRedemptions` | integer | Sets the maximum number of different AutoBills to which a Campaign Code may be applied. |
| `name` | string | Name of the Campaign.<br>**Note:** Read-only once the Campaign is Active. |
| `note` | string | An optional memo regarding the Campaign. |
| `offerEndDate` | dateTime | The last date on which the Campaign Code may be redeemed. |
| `offerStartDate` | dateTime | The first date on which the Campaign Code may be redeemed. |
| `percentageDiscount` | decimal | Defines the discount as a percentage of the original Product price.<br>**Note:** `flatAmountDiscount` and `percentageDiscount` are mutually exclusive.<br>**Note:** A `Campaign` must include either the `flatAmountDiscount` or the `percentageDiscount` data member, and it may not include both. |
| `promotionCode` | string | The redemption code associated with the Promotion. |
| `promotionCodeAliases` | string (0 or more) | An array of alternative redemption codes for the Promotion.<br>**Note:** Setting this array will replace any existing list of aliases; it will not add new values to an existing list. |
| `restrictToNewSubscription` | Boolean | A Boolean flag which, if `true`, indicates that the Campaign offer applies only to new AutoBills, and may not be applied to existing AutoBills. |
| `state` | `CampaignState` | State of the campaign:<br>• `Undefined`<br>• `Active`<br>• `Inactive`<br>• `Pending`<br>• `Complete` |

Table 6-1 `Campaign` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `timeGrant` | `CampaignTime-Grant` | Defines the grant, as a `CampaignTimeGrant` object.<br>This object includes two data members:<br>• `quantity`: the number of time units to grant.<br>• `type` = an enumeration of type `CampaignTimeGrantLengthType`, which may be Day, Week, Month, or Year. |
| `VID` | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `Campaign` object, leave this field blank; it will be automatically populated by CashBox. |

## 6.2 `Campaign` Related Object

The `Campaign` object has one related object:

- `CouponCode` Object

## CouponCode Object

Created by CashBox in response to a Generate Coupon Codes request, this object stores the randomly generated string Codes for the Coupon Campaign.

Each coupon code may be redeemed a fixed number of times. When a coupon code is applied to multiple AutoBillItems within a single AutoBill, or multiple transaction items on a single Transaction, it will count as a single Redemption. If a Coupon Code is applied to multiple AutoBills, or multiple Transactions, it will count as multiple redemptions. (Applying a Coupon Code to a single AutoBill at multiple points in time will also count as multiple redemptions.)

For more information on generating Coupon Codes, see Chapter 11: Working with Campaigns in the ***CashBox Programming Guide***.

Table 6-2 CouponCode Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| campaignId | string | **Read only.** A unique identifier for the Campaign object. |
| code | string | The Coupon value, which consists of <couponCodePrefix><separator (if defined)><randomly generated string>.<br>This field is available only when creating the Coupon. When retrieving the CouponCode object, this field will always be returned blank. |
| note | string | An optional memo regarding the CouponCode. |
| redeemedBy | CouponRedeemedBy | An array of CouponRedeemedBy objects, which lists the Account and date on which the Coupon was redeemed. Fields include:<br>• merchantAccountId (string)<br>• accountVID (string)<br>• date (dateTime) |
| sequence | int | A unique number for each Coupon Code generated, starting with 1. |
| state | CouponCodeState | The state of the Coupon Code:<br>• Not Yet Activated<br>• Activated<br>• Redeemed<br>• Expired<br>• Marked Used<br>• Retrieved<br>• Invalidated<br>• Initialized |
| VID | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new CouponCode object, leave this field blank; it will be automatically populated by CashBox. |

## 6.3 Campaign Methods

The following table summarizes the methods for the Campaign object.

Table 6-3 Campaign Object Methods

| Method | Description |
|---|---|
| activateCampaign | Sets the state of an Inactive or Pending Campaign to Active. |
| activateCode | Activates a CouponCode. |
| cancelCampaign | Cancels a Campaign and all of its existing promotionCodes or couponCodes. |
| *cloneCampaign* | *Vindicia best practices recommendation is to use the CashBox GUI interface, rather than the API, to clone a Campaign.* |
| *createCampaign* | *Vindicia best practices recommendation is to use the CashBox GUI interface, rather than the API, to create new Campaigns.* |
| deactivateCampaign | Sets the status of an Active or Pending Campaign to Inactive. |
| fetchAllCampaigns | Returns an array of Campaign objects, filtered by CampaignState, if specified. |
| fetchByCampaignId | Loads a Campaign by your Campaign ID. |
| fetchByVid | Loads a Campaign by its VID. |
| *generateCouponCodes* | *Vindicia best practices recommendation is to use the CashBox GUI interface, rather than the API, to generate Coupon Codes.* |
| *markAllCouponsUsed* | *Vindicia best practices recommendation is to use the CashBox GUI interface, rather than the API, to mark all Coupons Used.* |
| retrieveCouponCodes | Fetches previously generated CouponCodes. |
| *updateCampaign* | *Vindicia best practices recommendation is to use the CashBox GUI interface, rather than the API, to update a Campaign.* |
| validateCode | Checks if a Coupon or Promotion may be used. |

## activateCampaign

Sets the state of an `Inactive` or `Pending` `Campaign` to `Active`.

This method will fail if `CouponCodes` have been created, but not yet retrieved.

This method fails silently if the Campaign is already `Active`. Activating a Campaign from `Pending` sets the offer date to the current date.

### *Input*

***campaign:*** the `Campaign` object to be activated. Identify this object with its VID or `campaignId`.

***forcePending:*** a Boolean flag which, if set to `true`, allows the campaign to be activated, even from the `Pending` state. If this flag is `false` or omitted, the Campaign must be in the `Inactive` state to be activated.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Failed to update SOAP object in DB.`<br>• `Current state of` ***<state>*** `is an invalid state for Campaign activation.`<br>• `Failed to obtain SOAP object from DB.`<br>• `Tried to activateCampaign without doing createCampaign first.`<br>• `Can't activate campaign.` |

### *Example*

```
$camp = new Campaign();
   $camp->setCampaignId('camp132');
   $response = $camp->activateCampaign(false);
      // false is for the forcePending parameter

   // check $response
```

## activateCode

Activates a Coupon Code. Use this method to activate individual Coupon Codes before they may be used.

Coupon Codes may not be activated if their Campaign is not `Active`.

Use the `validateCode` method to activate a Code at the same time it is validated.

### *Input*

***code***: the Coupon Code to be activated. Because all Coupon Codes are unique, it is sufficient to specify the Code.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Code` ***code*** `doesn't represent a CouponCode.`<br>• `Code` ***code*** `can't be activated:` ***error-description.*** |

### *Example*

```
$camp = new Campaign();
    $response = $camp->activateCode(
        'promo123',     // the campaign code
    );
```

## cancelCampaign

This method cancels a Campaign, and sets its state to `Inactive`. Once cancelled, a campaign's discounts are unobtainable. Cancel cannot be reversed. To “reactivate” a cancelled Campaign, use the CashBox Portal to clone the Campaign. (Note that in cloning a Campaign, you must assign a new `campaignId` to the clone.)

Use this method to cancel a Campaign if something goes wrong, such as lost Coupon Codes, or a security breach.

### Input

***campaign:*** the `Campaign` object to be cancelled. Identify this object with its VID or `campaignId`.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Failed to update SOAP object in DB.`<br>• `Campaign is not in the right state for cancellation.`<br>• `Failed to obtain SOAP object from DB.`<br>• `Tried to cancelCampaign without doing createCampaign first.`<br>• `Can't cancel campaign until campaign code generation is complete.`<br>• `Can't cancel campaign: unable to invalidate campaign codes`<br>`Campaign codes requested =` ***<requested>***<br>`Campaign codes generated =` ***<generated>*** |

### Example

```
$camp = new Campaign();
    $camp->setCampaignId('camp132');
    $campaign->cancelCampaign();
```

## deactivateCampaign

Sets the status of an `Active` or `Pending` Campaign to `Inactive`.

This method fails silently if the Campaign is already `Inactive`.

### Input

***campaign:*** the `Campaign` object to be deactivated. Identify this object with its VID or `campaignId`.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Tried to deactivateCampaign without doing createCampaign first.`<br>• `Current state of <`***`state`***`> is an invalid state for Campaign deactivation.`<br>• `Failed to update SOAP object in DB.`<br>• `Failed to obtain SOAP object from DB.` |

### Example

```
$camp = new Campaign();
    $camp->setCampaignId('camp132');
    $response = $camp->deactivateCampaign();

    // check $response
```

## fetchAllCampaigns

This method returns an array of `Campaign` objects, filtered by `CampaignState`, if specified.

### Input

***status***: the (optional) `CampaignState` of the `Campaign`(s) you wish to have returned. To fetch all Campaigns, set status to `MatchAnyState`.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***campaign***: an array of all `Campaign` objects whose `status` matches the input.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$camp = new Campaign();
    $response = $camp->fetchAllCampaigns();
    // check $response

    $fetchedCampaigns = $response['campaign'];
    foreach ($fetchedCampaigns as $campaign) {
        print "got campaign "
            . $campaign->campaignId() . " "
            . $campaign->name()       . "\n";
    }
```

## fetchByCampaignId

Loads a `Campaign` by its Campaign ID.

### *Input*

***campaignId***: the ID of the Campaign you wish to return.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***campaign***: the `Campaign` object whose `CampaignId` matches the input.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Can't load Campaign with ID ***input-campaignId.*** |

### *Example*

```
$camp = new Campaign();
   $response = $camp->fetchByCampaignId('camp132');

   // check $response

   $campaign = $response['campaign'];
   print "got campaign " . $campaign->name() . "\n";
```

## fetchByVid

Loads a `Campaign` by its VID.

### Input

***vid***: the VID of the Campaign you wish to return.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***campaign***: the `Campaign` object whose `VID` matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | `Can't load Campaign with VID` ***input-vid.*** |

### Example

```
$camp = new Campaign();
   $response =
          $camp->fetchByVid('8367ae7148d071a4e25c24bef856f68f71ee03e3');

   // check $response

   $campaign = $response['campaign'];
   print "got campaign " . $campaign->name() . "\n";
```

## retrieveCouponCodes

Fetches previously generated Coupon Codes.

Coupon Codes must be retrieved *before* a Campaign is set to Active. Coupon Codes may not be retrieved for an Active Campaign.

For more information on generating Coupon Codes, see Section 10.2: Campaign Code Generation and Distribution in the ***CashBox User Guide***.

If you attempt to retrieve a page of Campaign Codes, and Code generation is not yet complete, `retrieveCouponCodes` will return an error, and the error string will indicate how many Codes have been generated, and how many have been requested. For example:

Campaign codes requested = ***nnn***; campaign codes generated = ***mmm***.

### *Input*

***campaign:*** the `Campaign` object for which `CouponCodes` should be returned. Identify this object with its VID or `campaignId`.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***couponCode:*** An array of Coupon Codes fetched.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Page number must be >= 0 and pageSize must be > 0.`<br>• `Can't load Campaign.`<br>• `Claimed to load a campaign by ID but it has no VID.`<br>• `Can't retrieve campaign codes when campaign state is 'Active.'`<br>• `Can't retrieve campaign codes:`<br>`Coupon codes not initialized.`<br>`Number of campaign codes requested not yet set up.`<br>`Number of campaign codes requested is zero.`<br>`Campaign codes requested = <`***`requested`***`>;`<br>`campaign codes generated = <`***`generated`***`>` |

## *Example*

```
$camp = new Campaign();
$camp->setCampaignId('camp132');
$response = $camp->retrieveCouponCodes(
    0,      // page num
    10,     // page size
);

$codes = $response['couponCode'];
for ($codes as $c) {
    print "code " . $c->sequence
           . " "     . $c->code
           . " "     . $c->state
           . "\n";
    }
```

## validateCode

Checks if a Coupon or Promotion Code may be used.

### *Input*

***code***: the `CampaignCode` to be validated.

***activateCodeNow:*** a Boolean flag which, if `true`, activates the code as soon as it has been validated. If `false` or omitted, an inactive `CampaignCode` will remain inactive.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***valid:*** a Boolean flag which indicates whether or not the Code is valid.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Code` ***input-code*** `doesn't represent a CouponCode.`<br>• `Code` ***input-code*** `can't be activated:` ***error-description.***<br>• `Code` ***input-code*** `is not redeemable:` ***error-description.*** |

### *Example*

```
$camp = new Campaign();
$response = $camp->validateCode(
    'promo123',     // the campaign code
    true            // activate now
);

// check $response
```

# 7 The `Chargeback` Object

A chargeback is initiated by a customer to reverse a specific transaction charge on their billing statement. Work with the `Chargeback` object when you subscribe to Vindicia's ChargeGuard service to dispute chargebacks on your behalf. (See Chapter 14: Common ChargeGuard Programming Tasks in the ***CashBox Programming Guide*** for more information.)

Each `Chargeback` object holds information about a chargeback against a specific transaction. This transaction could be a one-time transaction, or a rebilling transaction generated by a CashBox `AutoBill` object (subscription). If you are using ChargeGuard only, and are conducting transactions outside of CashBox, the transaction is simply a transaction reported by you.

Chargebacks are usually automatically downloaded by Vindicia from your payment processor. As Vindicia takes steps to dispute a chargeback on your behalf, the status of the `Chargeback` object will change.

# 7.1 Chargeback Data Members

The Chargeback object encapsulates information on a chargeback: the amount, date, reference number, and, most importantly, status.

The following table lists and describes the data members of the Chargeback object.

Table 7-1 Chargeback Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| amount | decimal | This chargeback's settlement amount, which usually matches the amount of the original transaction. In some cases, customers charge back only part of a transaction. (Vindicia does not provide information on the items that are charged back.)<br>**Note:** Given exchange-rate fluctuations, transactions across currencies might be charged back at amounts that differ from the original amounts. |
| caseNumber | string | Your bank's case number for this Chargeback object, if any. |
| currency | string | The ISO 4217 currency code (see www.xe.com/iso4217.htm) of this Chargeback object. This currency applies to the settlement amount (see the amount attribute). The default is USD. |
| divisionNumber | string | The number of your division or group your payment processor used when processing the original Transaction. Chase Paymentech refers to this number as the Division Number; Litle calls it the Report Group; MeS calls it the Profile ID. |
| merchantNumber | string | Your bank's merchant number, which identifies you as the merchant. |
| merchantTransactionId | string | Your unique identifier for the transaction associated with this Chargeback object. If CashBox generated the transaction, for example, for a recurring bill, CashBox created this ID for you when processing the transaction with your payment processor. If you did not process the transaction through CashBox, but only reported it to Vindicia, then this ID must match the order number you used when processing the transaction with your payment processor. |
| merchantTransactionTimestamp | dateTime | A timestamp that specifies the date and time when the original transaction occurred. |
| merchantUserId | string | Your unique identifier for the account of the customer who conducted the original transaction. See the merchantAccountId attribute of the Account object in Section 1.2: Account Data Members. |
| nameValues | NameValuePair[] | **Optional.** An array of name–value pairs for the Chargeback object.<br>See Section 10: The NameValuePair Object. |
| note | string | Notes on the Chargeback object. Vindicia personnel might make entries here during the dispute process. |
| presentmentAmount | decimal | The amount charged back (in the presentment currency), which usually matches the amount of the original transaction. Specify this attribute if the original transaction was processed with Chase Paymentech in a currency other than USD. |

Table 7-1 `Chargeback` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `presentmentCurrency` | string | The ISO 4217 currency code (see www.xe.com/iso4217.htm) of this transaction at presentment. The default is USD. |
| `postedTimestamp` | `dateTime` | A timestamp that specifies the date and time when the chargeback was posted in the Vindicia database. The difference in time between the chargeback, and this posted timestamp, will depend on the frequency at which Vindicia downloads chargebacks from your bank or payment processor. |
| `processorReceivedTimestamp` | `dateTime` | A timestamp that specifies the date and time when your bank received the chargeback from the customer. |
| `reasonCode` | string | The reason code reported by your bank for this `Chargeback` object. Reason codes vary from bank to bank. |
| `referenceNumber` | string | Your bank's reference number for this `Chargeback` object, if any. |
| `status` | `ChargebackStatus` | The current chargeback status in ChargeGuard. A chargeback goes through a life cycle as Vindicia disputes the chargeback on your behalf.<br>See Table 7-3: `ChargebackStatus` Object Values. |
| `statusChangedTimestamp` | `dateTime` | A timestamp that specifies the date and time for the last status change. |
| `VID` | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `Chargeback` object, leave this field blank; it will be automatically populated by CashBox. |

## 7.2 `Chargeback` Methods

The following table summarizes the methods for the `Chargeback` object.

Table 7-2 `Chargeback` Object Methods

| Method | Description |
|---|---|
| *`fetchByAccount`* | *(This method is not in use.)* |
| `fetchByCaseNumber` and `fetchByReferenceNumber` | Returns one or more `Chargeback` objects whose case or reference number matches the input. |
| `fetchByMerchantTransactionId` | Returns one or more `Chargeback` objects for the transaction whose ID assigned by you (`merchantTransactionId`) matches the input. |
| `fetchByStatus` | Returns one or more `Chargeback` objects whose status matches the input. |
| `fetchByStatusSince` | Returns one or more `Chargeback` objects whose status has changed since the specified timestamp. |
| `fetchByVid` | Returns a `Chargeback` object whose VID matches the input. |
| `fetchDelta` | Returns the `Chargeback` objects whose status has changed since this call was last made. |
| `fetchDeltaSince` | Returns the `Chargeback` objects whose status has changed since the specified timestamp. |
| `report` | Reports a batch of `Chargeback` objects to ChargeGuard. |
| `update` | Creates or updates a `Chargeback` object in the Vindicia database. |

## `fetchByCaseNumber` and `fetchByReferenceNumber`

Case and reference numbers are usually assigned by payment processors to track chargebacks in their systems. Some processors assign case numbers; others, reference numbers; and some assign both. In some cases, multiple chargebacks have the same case or reference number.

The `fetchByCaseNumber` method returns one or more `Chargeback` objects whose case number matches the input. The `fetchByReferenceNumber` method returns one or more `Chargeback` objects whose reference number matches the input.

### *Input*

For `fetchByCaseNumber()`, ***caseNumber*** is the payment processor's case number, which serves as the search criterion.

For `fetchByReferenceNumber()`, ***referenceNumber*** is the payment processor's reference number, which serves as the search criterion.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***chargebacks:*** an array of one or more `Chargeback` objects whose case or reference number matches the input.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, `fetchByCaseNumber` also returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load chargebacks by case number` ***input-case-number:*** `No match.`<br>• `Unable to load chargebacks by case number` ***input-case-number: error-description.***<br>• `Must specify a case number to load by!` |

In addition to those listed in Table 1: Standard Return Codes, `fetchByReferenceNumber` also returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load chargebacks by reference number` ***input-reference-number:*** `No match.`<br>• `Unable to load chargebacks by reference number` ***input-reference-number: error-description.***<br>• `Must specify a reference number to load by!` |

### *Example*

```
// The following example uses the fetchByCaseNumber call
// Call fetchByReferenceNumber similarly

$cb = new Chargeback();
$caseNo = "34593201";
$ret = $cb->fetchByCaseNumber($caseNo);
if ($ret['returnCode'] == 200) {
$fetchedChargebacks = $ret['chargebacks'];
   if ($fetchedChargebacks != null) {
      foreach ($fetchedChargebacks as $chargeback) {

         // process a fetched chargeback here …
         $status = $chargeback->getStatus();
         $amount = $chargeback->getAmount();
      }
   }
}
```

## fetchByMerchantTransactionId

The `fetchByMerchantTransactionId` method returns one or more `Chargeback` objects for the transaction whose ID assigned by you (`merchantTransactionId`) matches the input. Multiple chargebacks may be associated with one transaction, because a customer can charge back a transaction's line items separately.

### Input

***merchantTransactionId:*** your ID (`merchantTransactionId`) of the transaction whose chargebacks you wish to fetch.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***chargebacks:*** an array of one or more `Chargebacks` associated with the transaction whose ID matches the one specified as the input parameter.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load chargebacks by merchantTransactionId` ***input-merchantTransactionId:*** `No match.`<br>• `Unable to load chargebacks by merchantTransactionId` ***input-merchantTransactionId: error-description.***<br>• `Must specify merchant transaction id to load by!` |

### Example

```
$cb = new Chargeback();

$ret = $cb->fetchByMerchantTransactionId($txnId);
if ($ret['returnCode'] == 200) {
$fetchedChargebacks = $ret['chargebacks'];
    if ($fetchedChargebacks != null) {
        foreach ($fetchedChargebacks as $chargeback) {

            // process a fetched chargeback here …
            $status = $chargeback->getStatus();
            $amount = $chargeback->getAmount();
        }
    }
}
```

## fetchByStatus

The `fetchByStatus` method returns one or more `Chargeback` objects whose status matches the input.

Because multiple chargebacks can be of the same status, this method supports paging to limit the number of records returned per call. Occasionally, returning a large number of records in one call swamps buffers and might cause a failure. Vindicia recommends that you call this method in a loop, incrementing the page for each loop iteration with an optimal page size (number of records returned in one call) until the page contains a number of records that is less than the given page size.

### *Input*

***status:*** a string that describes the `Chargeback` status, which serves as the search criterion. See Table 7-3 for the values of the `Chargeback` enumeration.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

Table 7-3 `ChargebackStatus` Object Values

| Value | Description |
|---|---|
| `Challenged` | Vindicia has submitted rebuttal documents to your payment processor to dispute this chargeback. |
| `CollectionsNew` | An inactive status. |
| `CollectionsWon` | An inactive status. |
| `CollectionsLost` | An inactive status. |
| `Duplicate` | A duplicate chargeback has either been manually entered or received by Vindicia from the payment processor. Another chargeback in the queue exists with exactly the same information but is not marked duplicate. |
| `Expired` | The related documents or transaction details you reported were received too late by Vindicia to dispute this chargeback. |
| `Incomplete` | Vindicia has received chargeback information from the payment processor but does not have the original transaction details from you. |
| `Legitimate` | A valid chargeback because the original transaction was truly fraudulent. Vindicia does not represent or dispute legitimate transactions. |
| `Lost` | Vindicia challenged this chargeback but lost the case. |
| `New` | The first chargeback received by Vindicia, which is in the process of deciding how to pursue on your behalf. |
| `NewSecondChargeback` | A second chargeback has been received against a transaction that was initially charged back, disputed, and won. |

Table 7-3 `ChargebackStatus` Object Values *(Continued)*

| Value | Description |
|---|---|
| `Pass` | Even though all the documentation is available, Vindicia will not dispute this chargeback because of one or more of the following reasons:<br>The chargeback is less than US$5.<br>Not enough evidence exists for a dispute.<br>Regulations do not allow Vindicia to respond.<br>Vindicia does not recommend taking the dispute to arbitration. |
| `Retrieval` | An incoming retrieval or ticket request. |
| `Responded` | Vindicia has responded to the retrieval or ticket request. |
| *`Represented`* | *As a result of Vindicia's intervention, the chargeback was reversed in your favor. However, the customer or issuing bank is continuing the dispute by issuing a second chargeback. (This status is not in use.)* |
| `Won` | Vindicia challenged this chargeback, which has been reversed in your favor. |

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***chargebacks:*** an array of one or more `Chargeback` objects whose status matches the input.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load chargebacks by status` ***input-status*** `: No match.`<br>• `Unable to load chargebacks by status` ***input-status: error-description.***<br>• `Must specify a status to load by!` |

## *Example*

```
$cb = new Chargeback();
$page = 0;
$pageSize = 50;

do {
    $ret = $cb->fetchByStatus('Won', $page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedChargebacks = $ret['chargebacks'];
        if ($fetchedChargebacks != null) {
            $count = sizeof($fetchedChargebacks);
            foreach ($fetchedChargebacks as $chargeback) {

                 // process a fetched chargeback here …

                 $transactionId =
                 $chargeback->getMerchantTransactionId();
                 $amount = $chargeback->getAmount();
            }
            $page++;
        }
    }
} while ($count > 0);
```

## fetchByStatusSince

The `fetchByStatusSince` method returns one or more `Chargeback` objects whose statuses match the input and have changed since the specified timestamp. This call is similar the `fetchByStatus()` (see the preceding section), except that, with this call, you can restrict the retrieved chargebacks to a time window during which they changed to the status specified in the input.

Make this call periodically to, for example, retrieve the chargebacks that you have won so as to adjust your revenue statistics accordingly. Be sure to record the time you previously made this call and specify that time in the input for your next call.

### *Input*

***status:*** a `ChargebackStatus` value. See Table 7-3: `ChargebackStatus` Object Values.

***timestamp:*** the date and time on or after which the status of the `Chargeback` objects changed to ***status***.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page. Value must be greater than 0.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***chargebacks:*** an array of one or more `Chargeback` objects whose status changed since the timestamp specified in the input.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$cb = new Chargeback();
$page = 0;
$pageSize = 50;
// Assume a function is available that returned timestamp when
// we last made this call
$since = getLastCallTimestamp();
do {
    $ret = $cb->fetchByStatusSince('Won', $since, $page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedChargebacks = $ret['chargebacks'];
        if ($fetchedChargebacks != null) {
            $count = sizeof($fetchedChargebacks);
            foreach ($fetchedChargebacks as $chargeback) {
                 // process a fetched chargeback here …
                 $transactionId = $chargeback->getMerchantTransactionId();
                 $amount = $chargeback->getAmount();
            }
            $page++;
        }
    }
} while ($count > 0);
```

## fetchByVid

The `fetchByVid` method returns a `Chargeback` object whose VID matches the input, that is, it enables you to retrieve a `Chargeback` object by its VID.

When Vindicia adds a `Chargeback` object to its database by downloading the information from your payment processor or through your calling `update()` to create a `Chargeback` object, Vindicia assigns the object a unique identifier called VID. That VID is in the `Chargeback` object returned to you when you make calls to fetch chargebacks.

This call is useful for retrieving a specific chargeback because a `Chargeback` object does not have any other unique identifiers. Since there can be multiple chargebacks against one transaction, you cannot uniquely identify a chargeback with its associated `Transaction` object's ID, reference number, or case number.

### *Input*

***vid:*** the `Chargeback` object's Vindicia identifier, which serves as the search criterion.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***chargeback:*** the `Chargeback` object whose VID matches the input.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load chargebacks by VID` ***input-vid:*** `No match.`<br>• `Unable to load chargebacks by VID` ***input-vid: error-description.***<br>• `Must specify VID to load by!` |

### *Example*

```
$accountVid = 'MyVindiciaAccountVID';

// Create a SOAP caller object
$cb = new Chargeback();
$cbVID = "a209408014a33fec3dcd4a3339d78efc33603bfe";

// now load a chargeback object by VID
$response = $cb->fetchByVid($cbVid);
if($response['returnCode'] == 200) {
    $fetchedCb = $response['data']->chargeback;
}
else {
    // The call was unsuccessful
    print "Return code: " . $response['returnCode'] . "\n";
    print "Return string: " . $response['returnString'] . "\n";
}
```

## fetchDelta

The `fetchDelta` method is similar to `fetchDeltaSince`, except that `fetchDelta` does not require a timestamp as a parameter. CashBox keeps track of when you last called this method and returns the `Chargeback` objects whose statuses have changed since then. If you have never called this method before, CashBox returns all your chargebacks since January 1, 1970 (“epoch”).

This method is useful for periodically fetching the chargebacks with status changes or those that are newly added to the Vindicia database if you have no facilities for recording the time window for which you retrieved the results before.

For paging, this method only requires that you specify the page size. As with `fetchDeltaSince`, you need not increment through page numbers because this call keeps a record of the items previously returned to you in the last call. When you make this call next time, the results will continue onward from the last position in the result set.

### Input

***pageSize:*** the number of records to display per page per call.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***chargebacks:*** an array of one or more `Chargeback` objects that are newly created or whose statuses have changed since you last called `fetchDelta`.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$cb = new Chargeback();
$pageSize = 50;

do {
    $ret = $cb->fetchDelta ($pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedChargebacks = $ret['chargebacks'];
        if ($fetchedChargebacks != null) {
            $count = sizeof($fetchedChargebacks);
            foreach ($fetchedChargebacks as $chargeback) {

                 // process a fetched chargeback here …
                 $status = $chargeback->getStatus();
                 $transactionId = $chargeback->getMerchantTransactionId();
                 $amount = $chargeback->getAmount();
            }
            $page++;
        }
    }
} while ($count > 0);

// quit when no more objects are retrieved
```

## fetchDeltaSince

You can retrieve chargebacks from Vindicia in either of these ways:

- Manually, by logging into the CashBox Portal and downloading a comma-separated values (CSV) file of the chargebacks for a certain date range
- Programmatically, by making the `fetchDeltaSince` call, which returns one or more `Chargeback` objects whose statuses have changed since the specified timestamp

To always retrieve your chargebacks programmatically, call `fetchDeltaSince` periodically. The periodicity depends on your transaction and chargeback volume. Keep in mind that lag time usually exists between the time the customer calls the bank to charge back a transaction and the time the chargeback is downloaded from your payment processor and added to the Vindicia database.

Many merchants examine the statuses of their chargebacks from the information thus retrieved and, in some cases, use them as the basis on which to forbid or allow transactions initiated by certain customers or certain credit-card accounts. Each retrieved `Chargeback` object contains the ID of the original transaction that was charged back. You can retrieve the corresponding transaction and customer account with that ID by making the calls available for the `Transaction` object.

This method supports paging to limit the number of records returned per call. Occasionally, returning a large number of records in one call swamps buffers and might cause a failure. Vindicia recommends that you call this method in a loop, incrementing the page for each loop iteration with an optimal page size (number of records returned in one call) until the page contains a number of records that is less than the given page size.

### *Input*

***timestamp:*** the date and time on or after which a chargeback changed its status.

***endTimestamp:*** a timestamp that specifies the date and time before which a chargeback changed its status. If `null`, CashBox applies only ***timeStamp*** as the search criterion.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***chargebacks:*** an array of one or more `Chargeback` objects whose status has changed since ***timestamp (***and before ***endTimestamp***, if specified).

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

## Example

```
$cb = new Chargeback();
$page = 0;
$pageSize = 50;

// Here we want to fetch chargebacks that have changed in status or
// have been added since the last time we ran this call. Assume we have
// a function available to us that gives us the timestamp for the
// last time we ran this call

$since = getLastCallTime();
do {
    $ret = $cb->fetchDeltaSince($since, null, $page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedChargebacks = $ret['chargebacks'];
        if ($fetchedChargebacks != null) {
            $count = sizeof($fetchedChargebacks);
            foreach ($fetchedChargebacks as $chargeback) {

                // process a fetched chargeback here …
                $status = $chargeback->getStatus();
                $transactionId = $chargeback->getMerchantTransactionId();
                $amount = $chargeback->getAmount();
            }
            $page++;
        }
    }
} while ($count > 0);
```

## report

The `report` method reports a batch of `Chargeback` objects to ChargeGuard. This method is rarely used, because Vindicia usually retrieves chargebacks directly from the bank or payment processor on the merchant's behalf, and enters them into ChargeGuard. If your bank or payment processor does not allow Vindicia access to that information, you must retrieve the chargebacks yourself, and send the information to Vindicia by calling this method.

The data in this call is processed asynchronously. If the call succeeds, it means that CashBox has received the data and queued it for processing. Because CashBox processes chargebacks in the queue sequentially, and then adds them to the Vindicia database, a time lag exists between the time you report the chargebacks and the time they appear on the CashBox Portal.

An incomplete chargeback, or one that contains invalid data, might cause errors during processing, in which case CashBox might not add the chargeback to the database. Vindicia monitors its server logs for such errors and can, in some cases, fix them and reprocess the chargebacks. In other cases, a Vindicia representative might contact you for the correct data.

If you submit large amounts of data with this call, it might time out. Consider dividing the data into smaller batches and submitting them with separate calls, one batch at a time.

### *Input*

***chargebacks:*** an array of `Chargeback` objects to send to Vindicia.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | `Error saving transaction:` ***error-description.*** |

### *Example*

```
// create a chargeback object and populate it with data

$cb = new Chargeback();
$cb->setMerchantTransactionId("TX-2324");
$cb->setAmount(34.99);
$cb->setReferenceNumber("PTECH-42123");
$cb->setProcessorReceivedTimestamp('2009-11-11T22:34:32.265Z');

// Set other chargeback object fields here as available

...

// Create another chargeback to report

$cb2 = new Chargeback();
$cb2->setMerchantTransactionId("TX-2327");
$cb2->setAmount(19.99);
$cb2->setReferenceNumber("PTECH-42543");
$cb2->setProcessorReceivedTimestamp('2009-11-10T02:34:32.265Z');

$cb_soapcaller = new Chargeback();

// Make the SOAP call to report the chargebacks

$ret = $cb_soapcaller->report(array($cb, $cb2));
if ($ret['returnCode'] == 200) {
   log("Chargebacks submitted to Vindicia successfully at " . time() );
}
```

## update

The `update` method creates or updates a `Chargeback` object. This method is rarely used, because Vindicia usually creates and updates chargebacks by retrieving them directly from your payment processor, and updating their status during the dispute process.

You may also call `Chargeback.report()` to create one or more chargebacks in the Vindicia database. To create or update a single chargeback and immediately discover if the call succeeds or fails, call `update()`. The `report()` method processes data asynchronously, which means that even if you successfully submit a chargeback with a batch `report()` call but an error occurs during processing, you are not immediately aware of the error.

To create a `Chargeback` object, initialize the object and set the values for its data members, as appropriate, and then call `update()` to store the changes. When creating a new `Chargeback` object, do not set a value for `VID` because CashBox automatically generates that when you call `update()`. When updating an existing `Chargeback` object, identify it with its VID.

### *Input*

***chargeback:*** the `Chargeback` object to create or update. To update an object, identify it with its VID.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***chargeback:*** the `Chargeback` object that was created or updated.

***created:*** a Boolean flag that, if set to `true`, indicates that this method has created a new `Chargeback` object. A `false` setting indicates that `update` has updated an existing `Chargeback` object.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `200` | One of the following:<br>• `OK.`<br>• `Chargeback object was unchanged.` |
| `400` | One of the following:<br>• `Failed to save chargeback.`<br>• `Error saving chargeback disposition log entry:` ***error-description.*** |

### *Example*

```
// create a chargeback object and populate it with data

$cb = new Chargeback();
$cb->setMerchantTransactionId("TX-2324");
$cb->setAmount(34.99);
$cb->setReferenceNumber("PTECH-42123");
$cb->setProcessorReceivedTimestamp('2009-11-11T22:34:32.265Z');

// Set other chargeback object fields here as available

...

$ret = $cb->update();
if ($ret['returnCode'] == 200 && ($ret['created']) {
   log("Chargeback created with VID"
          . $ret['chargeback']->getVID() .
          " with Vindicia successfully at " . time() );
}
```

# 8 The `Entitlement` Object

An ***entitlement*** is the customer's right to access a product, as defined by their contractual agreement with a merchant. An `Entitlement` object (associated with an `Account` object) specifies whether a customer has the appropriate entitlement when the object is retrieved from the CashBox database. This object allows you to determine whether a customer can access a specific resource on your site at any given time.

CashBox uses several pieces of information to determine the content of an `Entitlement` object:

- The `merchantEntitlementId`, (the Entitlement Identifier), which is defined when creating new Entitlements for Products or Billing Plans.

  When creating Entitlements, use the `merchantEntitlementId` (the Entitlement Identifier) field to describe the entitlement conveyed. For example, to allow customers access to a Gold-Level subscription, create an Entitlement with `merchantEntitlementId`: `GoldAccess`.

  When creating `Product` and `BillingPlan` objects, specify the appropriate `merchantEntitlementId` in the object definition.
- CashBox calculates the `endTimestamp` of an Entitlement based on the AutoBill's Billing Plan. Until a payment attempt fails, or the AutoBill is stopped for any other reason, CashBox will assume that payments will continue to be made for the duration of the Billing Plan, and sets the Entitlement's `endTimestamp` according to its parameters. If the Billing Plan has a finite number of Billing Periods, the `endTimestamp` will be the termination date for the Billing Plan. If the Billing Plan has an infinite number of Billing Periods, the `endTimestamp` is `null`.
- The ***active*** flag on the `Entitlement` object defines whether the related entitlement is valid on the date you received the object from Vindicia. If the flag's value is `true`, it means that when CashBox constructed the `Entitlement` object, the customer was entitled to the access the object represents. To determine the duration, check the `endTimestamp` date.
- The `account` attribute of an `Entitlement` object specifies the customer to whom the object applies.

CashBox automatically grants entitlements upon successful creation of an AutoBill, and changes the end date only upon a payment failure or customer cancellation.

| **Note:** | If you are upgrading from CashBox 4.1 or previous, you must contact Vindicia Client Services to enable a merchant configuration setting which will allow Entitlements to work properly for CashBox 4.2 and greater. |
|---|---|

# 8.1 `Entitlement` Data Members

The following table lists and describes the data members of the `Entitlement` object.

Table 8-1 `Entitlement` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `account` | `Account` | The `Account` object with which this `Entitlement` object is associated.<br>See Section 1.2: Account Data Members. |
| `active` | Boolean | A Boolean flag that, if set to `true`, indicates that the `Entitlement` is currently active. |
| `autoBillVid` | string | The AutoBill VID associated with this `Entitlement`. |
| `description` | string | Your description for the `Entitlement`. |
| `endTimestamp` | `dateTime` | The date on which the Entitlement will expire, plus a grace period for the final billing transaction. (Blank for no end date.)<br>If CashBox returns this `Entitlement` object in the `active` status, you may assume that the object is active until this date (or the next failed billing attempt, if such occurs).<br>Re-fetch this `Entitlement` object to determine if it is still valid. If you call `fetchDeltaSince()` to retrieve `Entitlement` objects that might have changed but do not receive an update to this object, consider this Entitlement to be invalid after this timestamp. |
| `merchantAutoBillId` | string | Your AutoBill ID associated with this `Entitlement`. |
| `merchantEntitlementId` | string | An identifier for a specific privilege on your site. This ID, which has a special meaning in your application, specifies the resources to which a customer has access. Define this ID in the `merchantEntitlementId` field in `Product` or `BillingPlan` objects. |
| `merchantProductId` | string | Your Product ID associated with this `Entitlement`, if any. |
| `productVid` | string | The Product VID associated with this `Entitlement`, if any. |
| `source` | string | Indicates if this is a `Product`, `BillingPlan`, or `Account Entitlement`. |
| `startTimestamp` | `dateTime` | The time the entitlement begins. |

## 8.2 `Entitlement` Methods

The following table summarizes the methods for the `Entitlement` object.

Table 8-2 `Entitlement` Object Methods

| Method | Description |
|---|---|
| `fetchByAccount` | Returns one or more `Entitlement` objects for the `Account` object specified in the input. |
| `fetchByEntitlementIdAndAccount` | Returns the `Entitlement` object with the entitlement ID for the `Account` object specified in the input. |
| `fetchDeltaSince` | Returns one or more `Entitlement` objects that have changed since the specified timestamp. |

## fetchByAccount

The `fetchByAccount` method returns one or more `Entitlement` objects associated with the specified `Account` object. These `Entitlements` may be associated through an `AutoBill`, or directly with the `Account`.

Use this method to look up entitlements for a specific customer. Use the frequency of customer access, to determine how often to make this call. For example, if a customer on a monthly Billing Plan logs into your service several times each day, it's unnecessary and inefficient to make a call to CashBox to look up their entitlements for every login.

Instead, cache the entitlements obtained from this call locally. The `Entitlement` objects with the ***active*** flag set to `true` thus obtained and locally stored can be considered valid until the `endTimestamp` date.

You may cache `Entitlement` objects locally on your site with the database table shown below. Here, the columns customer_id and entitlement_id form a joint primary key.

| customer_id | entitlement_id | Last Update | Active Till … |
|---|---|---|---|
| Jdoe1970 | GoldAccessLevel1 | 2009-09-18 | 2009-10-13 |
| Jdoe1970 | VideoDownloadSpe-cial | 2009-09-18 | null |
| Jdoe1970 | LiveTechSupport | 2009-08-23 | 2009-09-01 |

For example, to check the entitlements for customer `Jdoe1970`, check if entries exist in the table for `Jdoe1970` and then follow these steps in your application logic:

- If entries exist, check if an entry exists in the table for the entitlement ID you need. If yes and if the `active_till` date is today or in the future, allow `Jdoe1970` access. If the `active_till` date is in the past, call `fetchByAccount()` or `fetchByEntitlementIdAndAccount()` and specify the related entitlement ID (`entitlement_id`).

  Afterwards, update the `Jdoe1970` table entries with the data in the `Entitlement` objects returned, and check the `active_till` date again. If it is not `null` and is in the future, allow `Jdoe1970` access.

- If no entries exist, call `fetchByAccount()` and add entries to the entitlement cache table with the data in the `Entitlement` objects returned. Next, check the entry for the entitlement ID you need for `Jdoe1970` and the `active_till` date. If that date is in the future, grant `Jdoe1970` access. If no entry exists or if the `active_till` date is `null` or in the past, `Jdoe1970` does not have that specific entitlement.

## Input

***account:*** the `Account` object for which to retrieve entitlements. Use the `merchantAccountId` or `VID` to identify the object.

***showAll:*** a Boolean flag that, if set to `true`, causes `fetchByAccount` to return all the `Entitlement` objects, including those that have expired. Otherwise, `fetchByAccount` returns only the active `Entitlement` objects.

***includeChildren:*** a Boolean flag that, if set to `true`, includes any children associated with this `Account`. If this flag is omitted, CashBox will interpret it as `false`, and constructs the query without looking at any child's account.

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***entitlements:*** an array of one or more `Entitlement` objects whose `Account` object matches the input.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 404 | Account not found. |

## Example

```
$account = new Account();
$acctId = 'xyz101';
$account->setMerchantAccountId($acctId);

// create the entitlement object to make the SOAP call
$entitlement = new Entitlement();
$showAll = true;

// fetch the records
$response = $entitlement->fetchByAccount($account, $showAll);
if ($response['returnCode'] == 200) {
   $fetchedEnts = $response['data']->entitlements;
   if ($fetchedEnts != null){
       foreach ($fetchedEnts as $ent) {
           $customer_id = $ent->getAccount()->getMerchantAccountId();
           $entitlement_id = $ent->getMerchantEntitlementId();
           $active = $ent->getActive();
           $active_till = null;
           if ($active) {
                $active_till = $ent->getEndTimestamp();
           }
           // use or locally store info obtained above
       }
   }
}
```

## fetchByEntitlementIdAndAccount

The behavior and use of the `fetchByEntitlementIdAndAccount` call are similar to the `fetchByAccount()` call. The only exception is that, instead of retrieving all the `Entitlement` objects for a specific customer, this method enables you to retrieve an `Entitlement` object with a specific entitlement ID for that customer. For details on how to interpret and store fetched `Entitlement` objects, see the `fetchByAccount` method.

### Input

***entitlementId:*** your entitlement ID (`merchantEntitlementId`), which serves as one of the two search criteria.

***account:*** the `Account` object, which serves as one of the two search criteria. Use the `merchantAccountId` or `VID` to identify the object.

***showAll:*** a Boolean flag, which, if `true`, shows all entitlements, including those that have expired. if `false` or `null`, returns only active entitlements.

***includeChildren:*** an optional Boolean flag that, if set to `true`, includes any children associated with this `Account`. If this flag is omitted, CashBox will interpret it as `false`, and constructs the query without looking at any child's account.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***entitlement:*** the `Entitlement` object with the specified entitlement ID (`merchantEntitlementId`) for the specified `Account` object.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Account not specified. |
| 404 | Account not found. |

### Example

```
$account = new Account();

$account->setMerchantAccountId('xyz101');

// create the entitlement object to make the SOAP call
$entitlement = new Entitlement();
$entitlement_id = 'ent_id_to_search_by';

// fetch the record
$response =
    $entitlement->fetchByEntitlementIdAndAccount($entitlement_id, $account);
if ($response['returnCode'] == 200) {
    $ent = $response['data']->entitlement;
    if ($ent != null) {
        $customer_id = $ent->getAccount()->getMerchantAccountId();
        $entitlement_id= $ent->getMerchantEntitlementId();
        $active = $ent->getActive();
        if ($active) {
            $active_till = $ent->getEndTimestamp();
        }
    // use or locally store info obtained above
    }
}
```

## fetchDeltaSince

The fetchDeltaSince call returns all the Entitlement objects that have changed since the specified timestamp. The change could be in the active status of an Entitlement, or in its endTimestamp if the entitlement is still active.

The purpose of this call differs from that of fetchByAccount() and fetchByEntitlementIdAndAccount(), which are used to look up the entitlements for a customer *while* they request access to a resource on your site. (fetchByAccount() and fetchByEntitlementIdAndAccount() often require that you make a request to the Vindicia servers during the customer's active session.)

To avoid making such a heavyweight call during a customer session, and to improve user experience, maintain a local cache of Entitlements in a table similar to the one shown in the fetchByAccount method for a faster lookup. Update that table periodically, or at a system quiescent time for all your customers by calling fetchDeltaSince().

Entitlements for an Account object may change for one of the following reasons:

- Your customer failed to pay their bill, and your grace period has been exhausted.
- You have created a new AutoBill object for an Account object.
- A cancellation for an AutoBill object with immediate disentitlement has occurred because either:
  - you have called AutoBill.cancel() or Account.stopAutoBilling() and set the flag for immediate disentitlement, or
  - Vindicia has received a chargeback from your payment processor against one of the transactions generated by the AutoBill object, and your profile configuration with Vindicia specifies that the customer be immediately disentitled in case of chargebacks.
- You have added or deleted entitlement IDs (merchantEntitlementIds) from a Product or BillingPlan object associated with an active AutoBill object.
- CashBox has postponed the end-date on an AutoBill object, as a result of a call that you made to delay the billing. See the delayBillingByDays() and delayBillingToDate() calls for AutoBill.

CashBox maintains a log of each event that can deactivate or extend an entitlement for all AutoBill and associated Account objects. When you call fetchDeltaSince(), CashBox constructs an Entitlement object from each log entry that has been added since the timestamp specified in the input, and includes it in the results returned to you. Thus, if an entitlement for a customer is changed several times during the fetchDeltaSince period, an Entitlement object that contains the same Account and Entitlement ID is in the result set for each of those changes. Because this method returns Entitlement objects in ascending order of the time when the log entries were made, in most cases you can determine the latest status of a customer's entitlement from the last Entitlement object with that ID in the result set. (In some cases, additional sorting logic is required to determine the active Entitlement with the latest end date.)

If you are using a database table, as described in the `fetchByAccount` method, to check the entitlements for a customer (for example, `Jdoe1970`), first check if entries exist in the table for `Jdoe1970` and then follow these steps in your application logic:

- If entries exist, check if one exists in the table for the entitlement ID you wish to look up. If it exists, and if the `active_till` date is today or in the future, allow `Jdoe1970` access. If the `active_till` date is in the past or is `null`, or if a row with the entitlement ID in question does not exist for `Jdoe1970`, `Jdoe1970` does not have access to the resources with that entitlement ID.
- If no entries exist, call `fetchByAccount()` for `Jdoe1970` and add the entries to the entitlement cache table with the data in the `Entitlement` objects returned. Next, check the entry for the entitlement ID you need to look up for `Jdoe1970` and the `active_till` date. If that date is in the future, grant `Jdoe1970` access. If no entry exists or if the `active_till` date is `null` or in the past, `Jdoe1970` does not have that specific entitlement.

The `fetchDeltaSince` method supports paging to limit the number of records returned per call. Returning a large number of records in one call may swamp buffers and might cause a failure. Vindicia recommends that you call this method in a loop, incrementing the page for each loop iteration with an optimal page size (number of records returned in one call) until the page contains a number of records that is less than the given page size.

## Input

***timestamp:*** the date and time after which to return the `Entitlement` objects that have changed.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

***endTimestamp:*** the time window's upper threshold by which to limit the search. If unspecified, this value defaults to the current time.

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***entitlements:*** an array of one or more `Entitlement` objects that have changed since ***timestamp***.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | `Invalid value or values of timestamp, and/or page, and/or page size.` |

## *Example*

```
$ent = new Entitlement();
$pg = 0;
$pageSize = 200;
$count = 0;
$endTimestamp = '2010-01-02T22:34:32.265Z';
$startTimestamp = '2010-01-01T22:34:32.265Z';

do {
    $ret = $ent->fetchDeltaSince($startTimestamp, $pg, $pageSize,
        $endTimestamp);
    $fetchedEnts = $ret['entitlements'];
    $count = 0;
    if ($fetchedEnts != null) {
        $count = count($fetchedEnts);
        foreach ($fetchedEnts as $ent) {
            $customer_id = $ent->getAccount()->getMerchantAccountId();
            $entitlement_id = $ent->getMerchantEntitlementId();
            $active = $ent->getActive();
            if($active == 1) {
                 $valid_till = $ent->endTimestamp();
            }
            // cache the data to your local database table here
        }
        $pg++;
    }
} while ($count > 0);
```

# 9 The GiftCard Object

The GiftCard object encapsulates information about a gift card offered by a merchant as a means of paying for a recurring subscription (AutoBill) or a one-time transaction. Payment with a gift card does not involve a monetary transaction. Instead, when you successfully redeem a gift card, CashBox adds credit to an Account or AutoBill. With the credit available to an Account, you can conduct a one-time transaction for that Account. Similarly, an AutoBill deducts credit available to it for every periodically recurring transaction it generates. The AutoBill offers entitlements to the subscriber as long as enough credit is available to sustain the offer. For more information, see the grantCredit method. For more information on how gift cards work within the CashBox system, see Chapter 12: Credit Grants and Gift Cards in the ***CashBox Programming Guide***.

With the redeemGiftCard() method of both the Account and AutoBill objects, you can redeem a gift card against those objects. For example, if you call redeemGiftCard() on an AutoBill object, the credit will be added to the AutoBill. See the redeemGiftCard, and redeemGiftCard methods.

CashBox determines how much credit to grant to an AutoBill or an Account by looking up a Product object. Create the Product object in advance in CashBox. The merchantProductId of this Product object should match the SKU (UPC) number returned by the gift card processor company (for example, InComm). The SKU/UPC number the processor returns when a gift card is redeemed is decided by a prior agreement between you and the gift card processor company. Before you start accepting gift cards from your customers, create a Product object in CashBox with a matching merchantProductId. When you create the Product, set its creditsGranted attribute to the amount of credit you want granted when the corresponding gift card is redeemed. See Section 13: The Product Object for more information.

As discussed in Section 12.2: Working with Gift Cards in the ***CashBox Programming Guide***, gift card redemption is a two-step process. In step 1, determine the status of the gift card by calling the statusInquiry() method, discussed below. If the status is Active, in the second step, redeem the card by calling redeemGiftCard() from the Account or AutoBill object.

# 9.1 GiftCard Data Members

The following table lists and describes the data members of the GiftCard object.

Table 9-1 GiftCard Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| *hashType* | *HashType* | *(This data member is not in use.)* |
| lastDigits | string | **Read-only**. Last four digits of a gift card's PIN. Do not populate this attribute; CashBox may populate this attribute when it returns a GiftCard object to you. For security, use this field for display to avoid displaying the entire PIN. |
| paymentProvider | string | Gift card processor company that CashBox should contact to check the status of a gift card and redeem the gift card. If left blank, this field defaults to InComm. **CashBox supports only gift cards redeemable by InComm, Inc.** |
| pin | string | Unique number associated with each gift card. A customer redeeming a gift card must give you this number. Populate this attribute in the GiftCard object when you check the status of the card for the first time. CashBox then creates a new record for this card in its system and assigns it a VID. For your subsequent calls that need to refer to this gift card, you need not populate the pin. Specifying only the VID will suffice. |
| *pinHash* | *string* | *(This data member is not in use.)* |
| pinLength | integer | **Read-only**. Number of characters or digits in the PIN of the gift card. Do not populate this attribute; CashBox may populate this attribute when it returns a GiftCard object to you. |
| product | Product | **Read-only**. Credit to add to the AutoBill or Account for which the card was redeemed, as specified by the Product object's creditsGranted attribute. CashBox populates this attribute in the GiftCard object it returns to you in response to a successful redeemGiftCard() call. The merchantProductId of this object matches the SKU/UPC returned by the gift card processor.<br>See Section 13.1: Product Data Members. |
| sku | string | **Read-only**. Unique ID (UPC) the gift card processor returns when CashBox redeems a specific type of gift card. Do not populate this attribute; CashBox may populate this attribute when it returns a GiftCard object to you when you call redeemGiftCard().<br>You must have previously created a Product object in CashBox with a merchantProductId matching each SKU you expect the processor to return, before redeeming gift cards. |

Table 9-1 `GiftCard` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `status` | `GiftCardStatus` | **Read-only**. Status of this gift card. CashBox populates this attribute in the `GiftCard` object returned to you when `GiftCard` is queried or changed as a result of calling `statusInquiry`, `redeemGiftCard`, or `reverse`.<br>See the `GiftCardStatus` Subobject. |
| `VID` | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `GiftCard` object, leave this field blank; it will be automatically populated by CashBox. |

# 9.2 GiftCard Subobjects

The GiftCard object has two subobjects:

- GiftCardStatus Subobject
- GiftCardStatusType Subobject

## GiftCardStatus Subobject

Describes the current status for a GiftCard by Activity.

Table 9-2 GiftCardStatus Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| *nameValues* | *NameValue-Pair[]* | *An array of name–value pairs.*<br>*(This data member is not in use.)* |
| providerResponseCode | string | Code that CashBox received from the gift card processor when it set the current status.<br>Use this code to determine why the processor did not authorize a certain gift card. |
| providerResponseMsg | string | Message string corresponding to the response code, if any, CashBox received from the gift card processor when it set the current status.<br>Use this string to determine why the processor did not authorize a certain gift card. |
| status | GiftCardStatusType | String describing the current status of a gift card. This string will be one of the values defined in the enumeration.<br>See the GiftCardStatusType Subobject for the status values returned when you execute statusInquiry, redeemGiftCard, or reverse. |
| timestamp | dateTime | The date and time when the GiftCard object acquired its current status. |

## `GiftCardStatusType` Subobject

Describes a list of `GiftCardStatus` types.

Table 9-3 `GiftCardStatusType` Object Enumeration Values

| Value | Description |
|---|---|
| `Active` | One of the following:<br>• You may redeem the `GiftCard` whose status you checked.<br>• An earlier call to reverse redemption of a `GiftCard` was successful and you can redeem the `GiftCard` again. |
| `Deactive` | One of the following:<br>• The gift card processor rejected the `GiftCard`.<br>• A call to redeem a `GiftCard` was unsuccessful.<br>• An attempt to reverse a redemption on a `GiftCard` was not authorized by the gift card processor. |
| `Redeemed` | Your call for redemption of a `GiftCard` was successful. |
| `RedemptionPending` | An earlier call to redeem the `GiftCard` did not yet complete. This is useful when there are two simultaneous attempts to redeem the same gift card, for example, via a multithreaded application. |
| *`Suspended`* | *(This status is not in use.)* |
| `Unknown` | CashBox cannot determine the status of the `GiftCard` for one of two reasons:<br>• It could not contact the processor.<br>• It could not interpret a response from the processor. |

## 9.3 `GiftCard` Methods

The following table lists and summarizes the methods for the `GiftCard` object.

Table 9-4 `GiftCard` Object Methods

| Method | Description |
|---|---|
| `reverse` | Reverses status of a `GiftCard` from a previous redemption attempt. |
| `statusInquiry` | Returns the latest status of a `GiftCard`<br>Use this method to determine whether a `GiftCard` can be redeemed. |

## reverse

The `reverse` method reverses a previous operation on a `GiftCard` (if the gift card processor allows it). Use `reverse` to reset the status of a gift card back to `Active` if a technical glitch occurred when you tried to redeem a gift card. `reverse` lets you retry the redemption. Do not use this method to undo the successful redemption of a gift card. This method does not automatically revoke credit from an `AutoBill` or `Account`, granted when the gift card was successfully redeemed.

### Input

***giftCard:*** the `GiftCard` object whose status you wish to reverse. Use the `VID` attribute to specify the `GiftCard` object.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***giftcard:*** the reversed `GiftCard` object, with updated status.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Failed to retrieve gift card` ***error-description.***<br>• `Reversal attempt failed` ***error-description.***<br>• `Reversal attempt rejected by GiftCard Processor.` |

### Example

```
$gc = new GiftCard();

// set the gift card VID. obtained when checking the gift card status

$gc->setVID($gcVID);

// Now make the SOAP API call to reverse the redemption

$response = $gc->reverse();

if ($response->['returnCode'] == 200) {

    // Also make sure the status of the gift card is 'Active'
    $updatedGc = $response['data']->giftcard;
    if ($updatedGc->getStatus()->getStatus() == 'Active') {
        print "Gift card is now redeemable \n";
    }
}
else {
    // Error while reversing the card
    print "Return code: " . $response['returnCode'];
    print " Return string: ";
    print $response['returnString'] . "\n";
}
```

## statusInquiry

The `statusInquiry` method causes CashBox to check with the gift card processor to learn the latest status of an input `GiftCard`. CashBox populates the `status` attribute in the `GiftCard` object it returns in response. Call this method before redeeming a gift card. If the status is `Active`, the gift card is redeemable.

### Input

***giftCard:*** the `GiftCard` object for which you want a status check. If this is a new gift card, be certain to specify the `pin` attribute. If this is an existing `GiftCard` object, you may specify only the `VID` attribute.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***giftcard:*** the `GiftCard` object requested, with an updated status.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Gift Card could not be saved prior to status. Status Inquiry attempt failed ***error-description.*** |

### Example

```
$gc = new GiftCard();

   // set the PIN provided by the customer
$gc->setPin('683092298403');
$gc->setPaymentProvider('InComm');

// Now make API call to check the status of the gift card

$response = $gc->statusInquiry();
if($response['returnCode'] == 200) {
   // The API call is successful. Now check the
   // status in the updated GiftCard object returned by this call

   $updatedGc = $response['data']->giftcard;
   $status = $updatedGc->getStatus();

   // the status thus obtained is an object of type GiftCardStatus
   // Now check if it indicates gift card is redeemable

   if ($status->getStatus() == 'Active') {

      // The gift card is redeemable, so retrieve its VID
      // so that we can reference it just by VID when we redeem it

      $gcVID = $updatedGc->getVID();
   }
   else {
      // Gift card is not redeemable. Inform the customer here
      // You may want to include the response received from the gift
      // card processor

      $responseCode = $status->getProviderResponseCode();
      $responseMsg = $status->getProviderResponseMessage();
   }
}
```

# 10 The NameValuePair Object

The NameValuePair object is referenced by several CashBox objects, and is used to hold attributes not otherwise supported in the object. This object is used to store a list of names, which are associated with text string values. These name-value pairs may be used to store custom data for your own, internal tracking purposes, or to store CashBox specific data, used for defined CashBox purposes.

> **Note:** CashBox allows only one value per name per object. NameValuePair objects may have several values associated with each name, but only one value may be used for a given name when assigning name-value pairs to an individual CashBox object.

For some objects, such as the PaymentMethod and Transaction objects, CashBox automatically generates certain name-value pairs, designated with vin: as the name's prefix. These pairs are listed and defined in the nameValue data member table for the specific object.

Cashbox also provides several pre-defined name-value pairs for use within CashBox. For these pairs, CashBox populates the name; you populate the value. These pairs include:

vin:Division: This name-value pair may be populated in an AutoBill, Transaction, or PaymentMethod (for purposes of validation) object. If used in conjunction with the divisionName name-value pair in your CashBox setup, it sends Transactions associated with these objects to the specified division (ID) at the processor.

vin:Division may be used to route Transactions to different payment processors, or to different merchant IDs configured at your payment processor in cases where you are not already routing by currency. The value you pass must match a value that has been configured in your merchant configuration in CashBox. Work with your Vindicia Client Services representative to configure this option.

vin:MandateFlag and vin:MandateVersion: When creating an AutoBill with EDD as the Payment Method, use vin:MandateFlag and vin:MandateVersion to associate a mandate document with the AutoBill. For example, set vin:Mandate flag to true, and vin:MandateVersion to 1.02 to associate a mandate document version 1.02 with the AutoBill.

vin:MandateBankName: The Bank Name for the EDD Payment Method (required only in the Netherlands).

## 10.1 `NameValuePair` Data Members

The following table lists and describes the data members of the `NameValuePair` object.

Table 10-1 `NameValue` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `name` | string | The `name` for the name/value pair. |
| `value` | string | The `value` for the name/value pair. |

## 10.2 NameValuePair Methods

The methods for NameValuePair are fetchNameValueNames and fetchNameValueTypes, which allow you to fetch the array of names for any given object.

### fetchNameValueNames

fetchNameValueNames accepts one parameter consisting of a type name, which must be one of the strings that fetchNameValueTypes returns. The fetchNameValueNames method returns an array of strings consisting of a list of distinct names from among the name/value pairs that the calling merchant has associated with objects of the given type.

#### *Input*

***type:*** the type of object for which the Names should be returned. This may be any of the CashBox objects that reference the NameValuePair object, and includes: Account, AutoBill, BillingPlan, CurrencyAmount, PaymentMethod, Product, TimeInterval, Transaction, or WebSession.

#### *Output*

***return:*** an object of type Return that indicates the success or failure of the call.

***names:*** an array of distinct names from the NameValuePair object associated with the input object.

#### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• Invalid Type.<br>• Failed to retrieve name/value names. |

#### *Example*

```
$nvp = new NameValuePair();
$response = $nvp->fetchNameValueNames('Account');

if ($response['returnCode'] == 200) {
    $names = $response['names];
    foreach ($names as $name) {
        print "$name\n";
    }
}
else {
    print "Error: " . $response['returnString'] . "\n";
}
```

## fetchNameValueTypes

`fetchNameValueTypes` takes no input parameters and returns a ***types*** list, which is an array of strings. Each string represents the name of a client-accessible type that supports name/value pairs.

Object ***types*** may include: `Account`, `AutoBill`, `BillingPlan`, `CurrencyAmount`, `PaymentMethod`, `Product`, `TimeInterval`, `Transaction`, or `WebSession`.

### *Input*

This method accepts no input parameters.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***types:*** an array of strings representing the types of objects that support name-value pairs.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$nvp = new NameValuePair();
$response = $nvp->fetchNameValueTypes();

if ($response['returnCode'] == 200) {
    $names = $response['types];
    foreach ($types as $type) {
        print "$type\n";
    }
}
else {
    print "Error: " . $response['returnString'] . "\n";
}
```

# 11 The PaymentMethod Object

The PaymentMethod object defines a customer's method of paying for your product or service. This is an umbrella object that encapsulates the subobjects that specify the details of various payment types, such as credit card, electronic check, and PayPal. An instance of the PaymentMethod object refers to only one payment type. When creating an instance, specify the payment type to which this object refers by populating the type attribute and then adding the related details. For example, for a credit-card payment, add details such as the card number and its expiration date in the corresponding subobject.

Although this object offers methods to independently create a new payment method in CashBox, to validate payment, and so forth, you might create PaymentMethod objects indirectly through Account, AutoBill, or Transaction objects, as follows:

- When creating an Account object, you can specify multiple PaymentMethod objects owned by the account in the paymentMethods attribute of the Account object.
- When creating an AutoBill object, you can specify a payment method for the rebilling transactions generated by the AutoBill object in its paymentMethod attribute. Otherwise, the AutoBill rebill transactions use the payment method available with the account.
- When creating a Transaction object, you can specify the sourcePaymentMethod attribute to define the means by which this transaction will be paid.

In the last two cases, CashBox creates the PaymentMethod object and associates it with the underlying Account object. For example, if you specify a PaymentMethod object in a Transaction object's sourcePaymentMethod attribute, CashBox attaches the PaymentMethod object to the Account object on the Transaction object. You can turn off this behavior by setting the ***active*** flag on the PaymentMethod object to false.

## 11.1 PaymentMethod Data Members

The following table lists and describes the data members of the PaymentMethod object.

Table 11-1 PaymentMethod Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| accountHolder-Name | string | The name of the account holder. |
| active | Boolean | A Boolean flag that, if set to true, causes CashBox to include this PaymentMethod object in the list of payment methods for the associated Account object, if any. |
| billingAddress | Address | The customer's billing address for this payment method only. This field is required if this payment method refers to a credit card and you want to conduct address-verification operations through AVS while validating the payment method. |
| boleto | Boleto | A subobject that specifies the details of a Boleto Bancário payment in Latin America. You must populate this attribute if you set the type attribute (described later in this table) to Boleto.<br>See the Boleto Subobject. |
| carrierBilling | CarrierBilling | A subobject that specifies the details of a Carrier Billing Payment Method. You must populate this data member if you set the type attribute to CarrierBilling. |
| creditCard | CreditCard | A subobject that specifies the details of a credit card. You must populate this attribute if you set the type attribute (described later in this table) to CreditCard.<br>See the CreditCard Subobject. |
| currency | string | The ISO 4217 currency code (see www.xe.com/iso4217.htm) to use for validating this payment method. The default is USD. Often, CashBox validates a payment method by only authorizing a transaction that uses the method for a small amount of this currency.<br>If this PaymentMethod object represents an EDD payment (that is, the type is set to DirectDebit), the currency must be one of the EDD-supported currencies, such as EUR for Euro. CashBox uses this currency while validating the payment method. |
| customerDe-scription | string | **Optional.** The customer's description for this payment method. |
| customerSpeci-fiedType | string | A customer-specified arbitrary string that describes the payment method type.<br>This field is optional for most credit cards, but required for the following card types, which must be specified exactly as listed:<br>• Switch<br>• Solo<br>• Dankort<br>• Laser, and<br>• CarteBleue. |

Table 11-1 PaymentMethod Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| directDebit | DirectDebit | A subobject that contains the details of the EDD payment. You must specify this attribute if you set the type attribute to DirectDebit.<br>See the DirectDebit Subobject. |
| ecp | ECP | A subobject that specifies the details of an electronic-check payment. You must populate this attribute if you set the type attribute to ECP.<br>See the ECP Subobject. |
| hostedPage | HostedPage | A subobject that contains the details of a payment accepted or applied using payment provider billing pages.<br>**Note:** Your customer's Account must exist before any Hosted Page related call references that Account.<br>See the HostedPage Subobject. |
| merchantAcceptedPayment | MerchantAcceptedPayment | A subobject that specifies the merchant's (optional) unique ID for this payment method. This is a free-form, unique string of 1024 or fewer bytes.<br>See the MerchantAcceptedPayment Subobject. |
| merchantPaymentMethodId | string | Your unique identifier for this PaymentMethod object. Once you've created this object, you may refer to it with this identifier. |
| nameValues | NameValuePair[] | **Optional.** An array of name–value pairs that provides additional information on the PaymentMethod object, as follows:<br>A name–value pair with the Name: CVN. The value for CVN is the security code on a credit card (the CVV2 code for Visa or the CVC code for MasterCard), for example, 111. This name–value pair is **required** if you want to run security code checks, such as CVV checks for Visa, on credit cards.<br>A name–value pair with Name: issueNumber. The value for issueNumber is the issue number on the customer's Switch or Solo card.<br>A name–value pair with Name: startDate. The value for startDate is the start date on a customer's Switch or Solo credit card with a date format of MMYY.<br>See Section 10: The NameValuePair Object. |
| paypal | PayPal | A subobject that specifies the details of a PayPal payment. You must populate this attribute if you set the type attribute to PayPal.<br>See the PayPal Subobject. |
| sortOrder | integer | The index into the paymentMethods array at which the PaymentMethod object is to be inserted if this object is associated with an Account object. (See the Account object's paymentMethods data member in Section 1.2: Account Data Members).<br>If no value is specified, CashBox will add the PaymentMethod at the beginning of the array, making it the default Payment Method for the Account.<br>If a value is specified, and a PaymentMethod already exists at that index, CashBox will insert the new PaymentMethod at the position indicated, and move the others down the array. |

Table 11-1 `PaymentMethod` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `token` | `Token` | An object that specifies the details of a token-based payment. You must populate this attribute if you set the type attribute to `Token`.<br>See Section 17.1: Token Data Members. |
| `type` | `PaymentMethod-Type` | **Required.** A string of the CashBox enumerated data type that defines the type of this payment method. Depending on this string, you must also populate the corresponding subobject in the appropriate attribute. For example, if you set the value of this data member to `CreditCard`, populate the `creditCard` data member with a `CreditCard` object that contains the card details.<br>See the `PaymentMethodType` Subobject. |
| `VID` | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `PaymentMethod` object, leave this field blank; it will be automatically populated by CashBox. |

## 11.2 PaymentMethod Subobjects

The PaymentMethod object has several subobjects:

- Boleto Subobject
- CarrierBilling Subobject
- CreditCard Subobject
- DirectDebit Subobject
- ECP Subobject
- HostedPage Subobject
- MerchantAcceptedPayment Subobject
- PaymentMethodType Subobject
- PayPal Subobject
- PhoneNumber Subobject
- PriceCriteria Subobject

### Boleto Subobject

Lists details for a Boleto Bancario payment.

Table 11-2 Boleto Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| fiscalNumber | string | The fiscal number that appears on the customer's Boleto Bancário payment slip. This number, formally called Casadastro de Pessoas, is formatted in a specific pattern (modulo 11).<br>**Note:** fiscalNumber is associated with a customer, not a payment method, and is analogous to a U.S. social security number. Treat fiscalNumber as Personally Identifiable Information (PII). |

## CarrierBilling Subobject

Lists details for a Mobile Carrier payment.

Table 11-3 CarrierBilling Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| countryCode | string | ISO 3166-1 alpha-2 Country Code for the customer's location. |
| currency | string | ISO 4217 Currency Code for either the static_price_inc_salestax, or the dynamic_target_price. (For dynamic pricing, the customer currency will be determined by the customer region/countryCode.) |
| encodedPhone-Number | string | The (read only) payment provider-encoded phone number used in the Transaction. |
| paymentProvid-er | PaymentPro-vider | The payment provider selected for the Transaction. (CashBox currently supports BOKU.)<br>See Section 12.1: PaymentProvider Data Members. |
| phoneNumber | PhoneNumber | **Optional.** The customer phone number used for the payment.<br>See the PhoneNumber Subobject. |
| priceCriteria | PriceCriteria | PriceCriteria are used when stipulating dynamic pricing for a Transaction. Note that priceCriteria has no meaning (and will be ignored) when creating a new PaymentMethod for an Account. Therefore only include this subobject with the PaymentMethod when processing a Carrier Billing-funded Transaction.<br>See the PriceCriteria Subobject. |

## `CreditCard` Subobject

Lists details for a Credit Card.

Table 11-4 `CreditCard` Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| `account` | string | The credit card's account number. Be certain to enter the number **in full** if you are using the associated payment method for CashBox one-time or recurring Transactions. When fully specified, this number must fulfill the Luhn check criterion. **Note:** CashBox partially masks the account number (for example, 444444XXXXXX1111) when returning this object to you in response to a call.<br>If this object is associated with a `Transaction` object that is reported directly to Vindicia (for example, if you are a ChargeGuard customer and report Transactions you process outside of CashBox), you might choose to omit this value or mask it partially for security. In that case, specify one of the following:<br>An encrypted value of the credit-card account number in the `accountHash` field (see the item below), or<br>The BIN (the first six digits of the credit-card number) and the last four digits of that number in the `bin` and `lastDigits` fields (see the items below). |
| `accountLength` | int | The length (number of digits) of the full account number. For example, for a Visa credit card, set the value to `16`. Specify this string only if you are **not** specifying the full account number for security reasons. |
| `bin` | string | The BIN, which is the first six digits of the full account number. Specify this string only if you are **not** specifying the full account number or its hash in the `accountHash` field for security when reporting transactions to Vindicia for ChargeGuard. You need not specify this field if the associated payment method is for a Transaction processed through CashBox. |
| `extendedCardAttributes` | `ExtendedCardAttributes` | Enhanced `auth` response details returned from Payment Provider.<br>See the `ExtendedCardAttributes` Subobject. |
| `expirationDate` | string | The `CreditCard` expiration date in YYYYMM format, where YYYY is the four-digit year and MM is the two-digit month. For example, the string for July 2007 is 200707. |

Table 11-4 `CreditCard` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `hashType` | HashType | The type of hash algorithm used if you specify the `accountHash` field. The allowed value is sha1, as CashBox only supports SHA1 hashing. Do not specify this field if the associated payment method is for a one-time or recurring Transaction processed through CashBox, as CashBox will automatically default to SHA1. |
| `lastDigits` | string | The truncated last part of the full credit-card account number, typically the last four or five digits of that number. Specify this string only if you are **not** specifying the full account number or its hash in the `accountHash` field for security when reporting transactions to Vindicia for ChargeGuard. |
| `VID` | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `CreditCard` object, leave this field blank; it will be automatically populated by CashBox. |

## DirectDebit Subobject

Lists details for a Direct Debit account.

Table 11-5 DirectDebit Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| account | string | The number of the customer's bank account from which to deduct payment. To use the associated PaymentMethod object for one-time or recurring transactions, you **must** specify the full account number. CashBox partially masks the account number (for example, 444444XXXXXX1111) when returning this object to you in response to an API call.<br>For security, if this object is associated with a Transaction object that is only reported to Vindicia (for example, if you are a ChargeGuard customer and report your transactions to Vindicia), you might choose to omit this value or mask it partially, such as by specifying an encrypted value of the account number in the accountHash field. |
| accountHash | string | A hash of the full account number, usually obtained through the Secure Hash Algorithm (SHA1). For numeric accounts, delete the nonnumeric characters before hashing. For calibration, the test number 1111111111111111 generates an SHA1 hash of 747417f2206148a3118d0f3adf20b5e4139baac.<br>Specify this string only if you are **not** specifying the full account number for security reasons when reporting transactions to Vindicia for ChargeGuard.<br>You need not specify this field when using the associated payment method in a Transaction processed through CashBox. |
| accountLength | int | The number of digits in the full account number.<br>Specify this attribute only if you are **not** specifying the full account number but are specifying the accountHash field only for security reasons when reporting transactions to Vindicia for ChargeGuard.<br>You need not specify this field for the associated payment method for a Transaction processed through CashBox. |
| bankSortCode | string | The European bank sort code that identifies the bank that houses the customer's bank account whose number is specified in the account field. This code is similar to the bank routing number in the United States.<br>You **must** specify this field for the associated payment method for a Transaction processed through CashBox. However, you may leave this field blank for bank accounts in the Netherlands or Belgium, if the countryCode attribute is set to NL or BE. |

Table 11-5 `DirectDebit` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `countryCode` | string | The ISO-3166-1 two-letter code for the country in which the related bank account is located. This code must match the country code specified in the `PaymentMethod` object's billing address. Valid values are `AT` (Austria), `DE` (Germany), and `NL` (the Netherlands).<br>You **must** specify this field for the associated payment method for a Transaction processed through CashBox. |
| `hashType` | `HashType` | The type of hashing algorithm used if you specify a value for the `accountHash` field. The allowed values are `sha1` and `md5`. Currently, only SHA1 hashing is supported on the server side.<br>You need not specify this field for the associated payment method for a Transaction processed through CashBox. |
| *`lastDigits`* | *string* | *(This data member is not in use.)* |
| *`ribCode`* | *string* | *(This data member is not in use.)* |

## ECP Subobject

Lists details for an ECP account.

Table 11-6 ECP Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| account | string | The full bank account number for this payment. Be certain to enter this number in full if you are using the associated payment method for CashBox Transactions.<br>**Note:** CashBox does not validate ECP accounts algorithmically, and partially masks the account number when returning it in response to your call. |
| accountHash | string | A hash of the full account number. Specify this string only if you are not specifying the full account number when reporting a Transaction for ChargeGuard. |
| accountLength | integer | The length (number of digits) of the full account number. Specify this string only if you are not specifying the full account number when reporting a Transaction for ChargeGuard. |
| accountType | AccountType | The type of bank account that issues this electronic check. The allowed values are ConsumerChecking, ConsumerSavings, and CorporateChecking. |
| allowedTransactionType | ECPTransactionType | The enumerated Transaction types allowed for ECP- or ACH-based Transactions that use this PaymentMethod object. The allowed values are All, Inbound, Outbound, InboundOutbound, Transfer, and NA. The default is All. |
| hashType | HashType | The type of hash algorithm used if you specify the accountHash field. The allowed values are sha1 and md5. CashBox supports SHA1 hashing only. You need not specify this field if the associated payment method is for a Transaction processed through CashBox. |
| lastDigits | string | The truncated last part of the full account number, typically the last four or five digits of that number. Specify this string only if you are **not** specifying the full account number or its hash in the accountHash field for security when reporting transactions to Vindicia for ChargeGuard. You need not specify this field if the associated payment method is for a one-time or recurring transaction processed through CashBox. |
| routingNumber | string | The bank routing number for an ACH or ECP account. Be certain to enter the correct number if you are using the associated payment method for CashBox Transactions. |

## ExtendedCardAttributes Subobject

This object is read-only, and lists auth response details returned from your Payment Provider.

Table 11-7 ExtendedCardAttributes Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| Affluent | int | Possible values: Y (true), N (false), U (undefined).<br>Applicable Processors: CPT, Litle. |
| Card-Description | string | The returned description for the card.<br>Applicable Processors: Litle, MeS. |
| CommercialCard | int | Possible values: Y (true), N (false), U (undefined).<br>Applicable Processors: CPT, Litle. |
| ConsumerCard | int | Possible values: Y (true), N (false), U (undefined).<br>Applicable Processor: Litle. |
| CountryOf-Issuance | string | Possible values: USA, etc.<br>Applicable Processors: CPT, Litle. |
| CreditCard | int | Possible values: Y (true), N (false), U (undefined).<br>Applicable Processors: Litle, MeS. |
| DebitCard | int | Possible values: Y (true), N (false), U (undefined).<br>Applicable Processors: Litle, MeS. |
| DurbinRegulat-ed | int | Possible values: Y (true), N (false), U (undefined).<br>Applicable Processor: CPT. |
| GiftCard | int | Possible values: Y (true), N (false), U (undefined).<br>Applicable Processors: Litle, MeS. |
| HealthcareCard | int | Possible values: Y (true), N (false), U (undefined).<br>Applicable Processors: CPT, Litle, MeS. |
| MassAffluent | int | Possible values: Y (true), N (false), U (undefined).<br>Applicable Processor: Litle. |
| PayrollCard | int | Possible values: Y (true), N (false), U (undefined).<br>Applicable Processors: CPT, Litle. |
| PINlessDebit-Card | int | Possible values: Y (true), N (false), U (undefined).<br>Applicable Processor: CPT. |
| PrepaidCard | int | Possible values: Y (true), N (false), U (undefined).<br>Applicable Processors: CPT, Litle, MeS. |
| Reloadable | int | Possible values: Y (true), N (false), U (undefined).<br>Applicable Processor: Litle. |

Table 11-7 `ExtendedCardAttributes` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `Signature-DebitCard` | int | Possible values: `Y` (true), `N` (false), `U` (undefined).<br>Applicable Processor: CPT. |
| `Virtual-AccountNumber` | int | Possible values: `Y` (true), `N` (false), `U` (undefined).<br>Applicable Processor: Litle. |

## `HostedPage` Subobject

Lists details for a HostedPage account.

| **Note:** | The customer's Account must exist before any Hosted Page related call references that Account. |
|---|---|

Table 11-8 `HostedPage` Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| `countryCode` | string | The ISO 3166 ( alpha-2 ) country code for customer's location,<br>**Note:** The combination country+processorPaymentMethodId+merchantId must be set at GlobalCollect |
| `language` | string | **Optional.** The ISO 639-1 language matrix code for the payment pages. |
| `paymentProvider` | `PaymentProvider` | The payment provider selected for the Transaction. (CashBox supports GlobalCollect.)<br>See Section 12.1: PaymentProvider Data Members. |

Table 11-8 `HostedPage` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `processor-PaymentMetho-dId` | string | The payment method to use for the Transaction. (These values correspond to GlobalCollect's payment product ID.)<br>CashBox supports the following values:<br>Moneybookers: 843<br>Paysafecard: 830<br>Ukash: 1400<br>Direct Debit (Germany): 702<br>Recurring Direct Debit (Germany): 712<br>Direct Debit (Austria): 703<br>Recurring Direct Debit (Austria): 713<br>Direct Debit (Netherlands): 701<br>Recurring Direct Debit (Netherlands): 711<br>Direct Debit (Spain): 709<br>Recurring Direct Debit (Spain): 719<br>PayPal: 840<br>iDEAL: 809<br>Sofortuberweisung: 836<br>Yandex: 849<br>Webmoney: 841<br>CashU: 845<br>Alipay: 861 |
| `returnUrl` | string | **Optional.** The URL to which you would like customers to be redirected after they have successfully completed the HostedPage transaction.<br>(This is often your confirmation page.) |

## `MerchantAcceptedPayment` Subobject

Lists details for a payment entered manually by a merchant.

Table 11-9 `MerchantAcceptedPayment` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `account` | string | The full account number. |
| `accountHash` | string | A hash of the full account number. Any non-numeric characters should be removed prior to hashing. If the account number is provided, this may be left blank and the hash will be calculated by CashBox. The exact length and format of this string may depend upon the hash algorithm chosen. |

Table 11-9 `MerchantAcceptedPayment` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `accountLength` | string | Length of the total account number. If the full account number is submitted, this field may be left blank, and CashBox will calculate it. |
| `amount` | decimal | The amount paid by a customer. This value must be 0 for PaymentMethods attached to AutoBills.<br>**Note:** `MerchantAcceptedPayment` PaymentMethods may be attached to AutoBills to indicate that the customer should be invoiced (rather than automatically charged). |
| `currency` | string | The ISO 4217 currency code for the payment. Currency or token must be specified, and must match the currency for charges contained in the invoice/AutoBill. |
| `hashType` | HashType | The algorithm used to hash the account number. If this value is not provided, CashBox will use assume SHA1. |
| `lastDigits` | string | The last part of the account number for display purposes, generally the last four digits. If the account field is provided, this may be left blank and will be filled in by CashBox. |
| `note` | string | An optional memo regarding the payment made. |
| `paymentId` | string | The ID of the payment accepted by the merchant. |
| `paymentType` | string | The type of payment accepted by the merchant. |
| `timestamp` | `dateTime` | The time that payment occurred. |
| `token` | `Token` | The Token associated with the amount (if this is a Token-based AutoBill).<br>See Section 17.1: Token Data Members. |

## `PaymentMethodType` Subobject

Describes the type of PaymentMethod.

**Note:** CashBox does not support partial payments data for the Merchant Accepted Payment `paymentMethodType`.

CashBox does not support the `CarrierBilling` or `Boleto` Payment Method Type with `AutoBill.migrate`.

Table 11-10 `PaymentMethodType` Object Values

| Value | Description |
|---|---|
| `Boleto` | The payment method is Boleto Bancário. |
| `CarrierBilling` | The payment method is Carrier Billing. |
| `CreditCard` | The payment method is credit card. |
| `DirectDebit` | The payment method is direct debit. CashBox supports direct debit payment methods in the Netherlands, Germany, and Austria. |
| `ECP` | The payment method is electronic check through the ACH network. |
| `HostedPage` | The payment method is HostedPage.<br>**Note:** The customer's Account must exist before any Hosted Page related call references that Account. |
| `MerchantAcceptedPayment` | The payment is manually entered by the merchant. |
| `PayPal` | The payment method is PayPal. |
| `Token` | The payment method is Tokens. |

## `PayPal` Subobject

Lists details for a PayPal account.

Table 11-11 `PayPal` Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| `cancelUrl` | string | The URL to which you would like to redirect customers if PayPal indicates failure after they have completed the payment process on the PayPal site. |
| *`hashType`* | *`HashType`* | *(This data member is not in use.)* |
| *`password`* | *string* | *(This data member is not in use.)* |
| *`passwordHash`* | *string* | *(This data member is not in use.)* |
| `payerId` | string | Unique PayPal customer account identification number in PayPal ExpressCheckout. |
| `returnUrl` | string | The URL to which you would like customers to be redirected after they have successfully completed payment transactions on the PayPal site.<br>(This is often your confirmation page, on which the customer confirms the order and payment or the billing agreement.) |
| `paypalEmail` | string | Email used in PayPal ExpressCheckout (read-only). (CashBox automatically populates this field with the customer email addressed used in the PayPal Transaction.) |
| `referenceId` | string | This data member maps to the PayPal field "REFERENCEID" which is the Billing Agreement ID or Reference Transaction ID associated with a PayPal Billing Agreement.<br>**Note:** If you enter a value for this data member, set `requestReferenceId` to `false`. |
| `requestReferenceId` | Boolean | When processing the initial Transaction for an AutoBill, ask PayPal for a Reference ID that can be used in the future for recurring billing. This works only if you have been previously approved for Reference Transactions by PayPal. |

**Note:** PayPal has one email address, `payerId`, that identifies the PayPal account of the customer.

## `PhoneNumber` Subobject

The `PhoneNumber` object is used to store customer phone number information, for use in Carrier Billing. This object is optional. Information contained within it is not required to be passed to the payment provider at this time.

For more information, see Section 6.3: Using Carrier Billing for One-Time Transactions in the ***CashBox Programming Guide***.

The `PhoneNumber` object describes a customer phone number used for Carrier Billing.

Table 11-12 `PhoneNumber` Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| `areaCode` | string | **Required.** The area code segment of the phone number. |
| `countryCode` | string | The Country Code segment of the phone number. |
| `extension` | string | The phone number's extension. |
| `localNumber` | string | **Required.** The local number (excluding extension). |
| `phoneType` | PhoneType | The type of phone. |
| `rawInput` | string | Raw, unfiltered data input by customer. |

## PriceCriteria Subobject

The PriceCriteria object is used to define dynamic pricing for a CarrierBilling Transaction. (Because mobile payments may only be processed for fixed values in a given country, pricing may be defined as Static, or Dynamic. The PriceCriteria subobject allows you to define your pricing structure.)

Note that priceCriteria has no meaning (and will be ignored) when creating a new PaymentMethod for an Account. Therefore, include this subobject with the PaymentMethod only when processing a CarrierBilling-funded Transaction.

Table 11-13 PriceCriteria Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| countryCode | string | ISO 3166-1 alpha-2 countryCode for customer location. This value will override the CarrierBilling object's countryCode data member. |
| currency | string | ISO 4217 Currency Code for either the staticPriceIncSalesTax, or the dynamicTargetPrice. (For dynamic pricing, the customer currency will be determined by the customer region/countryCode.) This value will override the CarrierBilling object's currency data member. |
| description | string | A description for the Price Criteria. |
| dynamicDeviation | int | The % deviation (+/- 1000) from the target value that is acceptable as a price point selection. |
| dynamicMatch | int | The % deviation (+/- 1000) from the target value that is classified as an exact match. |
| dynamicPriceMode | DynamicPriceMode | Defines which price point element is matched by the dynamic pricing algorithm.<br>DynamicPriceMode may be one of three types:<br>Price: The target value will be matched to "price-inc-salestax" values in the payment provider's price point matrix.<br>PayoutGross: The target value will be matched to "gross-payout" values in the payment provider's price point matrix.<br>PayoutNet: The target value will be matched to "net-payout" values in the payment provider's price point matrix. |
| dynamicTargetPrice | decimal | The target price in the specified currency for dynamic pricing. |
| fwdUrl | string | Overrides both the successful transaction forward-to URL, and the failed transaction forward-to URL. |
| merchantServiceIdentifier | string | Your service identifier for the payment provider. |

Table 11-13 `PriceCriteria` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `paymentProvider` | `PaymentProvider` | `PaymentProvider` selected for the Transaction. This value will override the `CarrierBilling` object's `paymentProvider` data member. (CashBox currently supports BOKU as a `CarrierBilling` payment provider.)<br>See Section 12.1: PaymentProvider Data Members. |
| `pricePointDeviationPolicy` | `PricePointDeviationPolicy` | The allowed price deviation policy for `CarrierBilling` payments using dynamic price selection.<br>`PricePointDeviationPolicy` may be one of three values:<br>`HiPreferred`: A solution higher than the target value will be favored over a lower solution.<br>`HiOnly`: Only solutions higher than the target value will be returned.<br>`LowPreferred`: A solution lower than the target value will be favored over a higher solution.<br>`LowOnly`: Only solutions lower than the target value will be returned.<br>`NearestNoPreference`: The closest solution to the target value will be selected. |
| `staticPriceIncSalesTax` | decimal | The price including tax (the amount your customer will pay). Used with Transactions with static "exact match" pricing. |
| `staticSelectionRowRef` | int | The row number identifier in the static product/service price matrix. |
| `subMerchantIdentifier` | string | The sub-merchant identifier for the Transaction. |

## 11.3 PaymentMethod Methods

The following table summarizes the methods for the PaymentMethod object.

Table 11-14 PaymentMethod Object Methods

| Method | Description |
|---|---|
| fetchByAccount | Returns one or more PaymentMethod objects whose Account object matches the input. |
| fetchByMerchantPayment-MethodId | Returns a PaymentMethod object whose merchantPayment-MethodId matches the input. |
| fetchByVid | Returns a PaymentMethod object whose VID matches the input. |
| fetchByWebSessionVid | Returns a PaymentMethod object whose WebSessionVid matches the input. |
| update | Creates or updates a PaymentMethod object. |
| validate | Validates but does not store a PaymentMethod object. |

## fetchByAccount

The `fetchByAccount` method returns one or more `PaymentMethod` objects whose `Account` object matches the input. You can, for example, call this method to retrieve the payment methods a customer has used before, present them to the customer, and ask them to choose one for a new product or subscription purchase.

### Input

***account:*** the `Account` object that serves as the search criterion. Use the `merchantAccountId` or `VID` to identify the object.

***includeChildren:*** an optional Boolean flag that, if set to `true`, includes any children associated with this `Account`. If this flag is omitted, CashBox will interpret it as `false`, and constructs the query without looking at any child's account.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***paymentMethods:*** an array of one or more active `PaymentMethod` objects associated with the `Account` object specified in the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | `Account parameter is required.` |
| `404` | One of the following:<br>• `No PaymentMethods found for account.`<br>• `Account not found.` |

### Example

```
$merchantId = '12345';

// Create a payment method object to make the call
$paymentMethod = new PaymentMethod();

// Create an account object to search the payment methods by
$account = new Account();
$account->setMerchantAccountId('abc101');

$response = $paymentMethod->fetchByAccount($account);
if($response['returnCode'] == 200) {
    $fetchedPms = $response['data']->paymentMethods;

    if($fetchedPms != null) {

        foreach ($fetchedPms as $pm) {

            // process a fetched payment method object here
            $accountHolder = $pm->getAccountHolderName();
            if ($pm->getType() == "CreditCard") {
                 $cc = $pm->getCreditCard();
                 // process other credit card attributes here
            }
        }
    }
}
```

## fetchByMerchantPaymentMethodId

The `fetchByMerchantPaymentMethodId` method returns a `PaymentMethod` object whose `merchantPaymentMethodId` (assigned by you) matches the input.

### Input

***paymentMethodId:*** the payment method ID (`merchantPaymentMethodId`), which serves as the search criterion.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***paymentMethod:*** the `PaymentMethod` object whose `merchantPaymentMethodId` matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Missing required parameter paymentMethodId. |
| 404 | Unable to find requested PaymentMethod: ***error-description.*** |

### Example

```
$merchantId = '12345';

// Create a payment method object to make the call
$paymentMethod = new PaymentMethod();

$pmId = 'PM34922012';

$response = $paymentMethod->fetchByMerchantPaymentMethodId($pmId);

if($response['returnCode'] == 200) {
    $fetchedPm = $response['data']->paymentMethod;

    if($fetchedPm != null) {

        // process the fetched payment method object here
        $accountHolder = $fetchedPm->getAccountHolderName();
        if ($fetchedPm->getType() == "CreditCard") {
            $cc = $fetchedPm->getCreditCard();
            // process other credit card attributes here
        }
        else if($fetchedPm->getType() == "ECP") {
            $ecp = $fetchedPm->getEcp();
            // process other ecp attributes here
        }
    }
}
```

## fetchByVid

The `fetchByVid` method returns a `PaymentMethod` object whose VID matches the input.

VID is Vindicia's unique identifier for a `PaymentMethod` object. While creating a `PaymentMethod` object, do not specify a VID for it yourself. When CashBox receives a `PaymentMethod` object in a call, such as `PaymentMethod.update()` or `Transaction.AuthCapture()`, if no VID or `merchantPaymentMethodId` exists inside the object, CashBox creates a new `PaymentMethod` object and assigns it a VID. Retrieve this VID from the `PaymentMethod` object CashBox returns to you in response to your call. Then, you may identify the object with the VID, and retrieve it by calling this method.

### *Input*

***vid:*** the `PaymentMethod` object's Vindicia identifier, which serves as the search criterion.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***paymentMethod:*** the `PaymentMethod` object whose VID matches the input.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | `Missing required parameter 'vid'.` |
| `404` | One of the following:<br>• `Unable to find requested PaymentMethod:` ***error-description.***<br>• `Unable to find requested PaymentMethod: No matches.` |

### *Example*

```
$merchantId = '12345';

// Create a payment method object to make the call
$paymentMethod = new PaymentMethod();

$vid = '6d46cb877cc9b0a458d61e0771e740ad8b531ec9';

$response = $paymentMethod->fetchByVid($vid);

if($response['returnCode'] == 200) {

    $fetchedPm = $response['data']->paymentMethod;
    if($fetchedPm != null) {

        // process the fetched payment method object here
        $accountHolder = $fetchedPm->getAccountHolderName();
        if ($fetchedPm->getType() == "CreditCard") {
            $cc = $fetchedPm->getCreditCard();
            // process other credit card attributes here
        }
        else if($fetchedPm->getType() == "ECP") {
            $ecp = $fetchedPm->getEcp();
            // process other ecp attributes here
        }
    }
}
```

## fetchByWebSessionVid

Use Vindicia's Hosted Order Automation (HOA) to create CashBox objects that contain sensitive payment information, such as credit-card account numbers. Store credit card numbers directly on Vindicia's servers after your customers have submitted their data through a specially designed Web order form accessed from your server. Because HOA bypasses your server altogether at form submission, you need not comply with PCI requirements. See Chapter 13: Hosted Order Automation in the ***CashBox Programming Guide*** for details on HOA.

You must create a `WebSession` object on Vindicia's servers before serving an order form to your customer to track the form's submission to Vindicia. (For details, see Section 19: The WebSession Object.) You may then call the `fetchByWebSessionVid` method to retrieve the `PaymentMethod` object created by HOA when a customer submits an order form, which results in a one-time or recurring bill.

The `WebSession` object's VID serves as the tracking ID for the Web session, from serving the order form to a customer, to returning a success or failure page to that same customer. Use the `WebSession` object to program the success page (see the `WebSession` object's `returnURL` attribute), to which HOA redirects the customer's browser after successfully processing the data in the order form. On your success page, the `WebSession` object's VID is available to you because HOA passes it during the redirection. In turn, you may pass that VID as the input parameter to this call and retrieve the `PaymentMethod` object created by HOA. Finally, extract the contents of the `PaymentMethod` object and include them, as appropriate, in the success page to be returned to the customer.

### *Input*

***vid:*** the `WebSession` object's Vindicia unique identifier for tracking the submission of the order form.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***paymentMethod:*** a `PaymentMethod` object, created by HOA as a result of an order form submitted by a customer.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Missing required parameter 'vid'. |
| 404 | Unable to find requested PaymentMethod: No matches. |

### *Example*

```
// To use the fetchByWebSessionVid call on a success web page

$webSessionVid = …; //passed in by redirected page

$soap = new WebSession($soapLogin, $soapPwd);

$response = $soap->fetchByVID($webSessionVid);

if ($response['returnCode'] == 200) {

    $fetchedWs = $response['data']->session;

    // check if the CashBox API call made by HOA was successful
    $retCode = $fetchedWs->apiReturn->returnCode;
    if ($retCode == 200) {

        // Assuming HOA created a PaymentMethod object, fetch it

        $soapPm = new PaymentMethod($soapLogin, $soapPwd);
        $resp = $soapPm->fetchByWebSessionVid($webSessionVid);
        if ($resp['returnCode'] == 200) {
            $createdPm = $resp['data']->paymentMethod;

            // Get PaymentMethod contents here to be included in
            // HTML returned to the customer.
        }
        else {
            // Return error message to customer
        }
    }
    else {
        // return failure page to customer
    }
}
else {
    // Return error message to the customer
}
```

## update

The `update` method creates or updates a `PaymentMethod` object. To encapsulate a specific payment method for a customer, you must specify the payment type in the object's `type` attribute, and then populate the payment details (specific to the payment type) in a `PaymentMethodType`-specific subobject. For example, if you set `type` to `PayPal`, construct a `PayPal` object and set it in the `PaymentMethod` object's `PayPal` attribute.

> **Note:** The customer's Account must exist before any Hosted Page related call references that Account.

This call supports a flag that may be set to validate the payment method. The validation process varies according to the payment method type. For example, for credit-card-based payment methods, validation proceeds by authorizing a transaction for US$1 (or for an amount in the currency you specified on the `PaymentMethod` object) with your payment processor. That transaction is not captured so the customer is not charged. However, CashBox does not support validation for the `PayPal` payment method type.

In case of the credit-card payment method, you can screen the card for fraud risk when creating the `PaymentMethod` object by specifying a chargeback probability score (also called risk score) that is acceptable to you. CashBox scores the payment method for fraud risk by examining the billing address, the BIN (the first six digits of the card's account number), the previous chargebacks on transactions conducted with this card, and other criteria. For details, see the `score` method. If CashBox evaluates the risk score for the payment method to be higher than your acceptable score, the creation process fails.

If validate is set to `true`, and validation fails, the `PaymentMethod` object is not created or updated.

The following table describes the validation process for the various payment methods.

Table 11-15 Validation Process by Payment Method Type

| Payment Method Type | Validation Process |
|---|---|
| Credit card | The account number must meet the Luhn check criterion and the payment processor must authorize a transaction for a small amount (one currency unit or, if the currency is not specified on the `PaymentMethod` object, US$1). CashBox sends this transaction to the payment processor for authorization only and does not capture it so the customer is not charged. These transactions, whose success status is `AuthorizedForValidation`, are displayed on the CashBox Portal. |
| Direct debit | First, CashBox internally validates the account number (`account`) and bank sort code (`bankSortCode`). The rules that apply depend on the country specified in the `DirectDebit` object. If internal validation succeeds, CashBox contacts the payment processor (currently, Chase Paymentech only) and conducts an auth operation, with no capture, on a transaction that uses the EDD payment method for a small amount, such as one unit of the currency specified on the `PaymentMethod` object.<br>The payment processor's initial response to the `auth` call is based on the verification that the account number and the bank sort code do not match any numbers in the negative file (blacklist) maintained by the processor. In that case, CashBox considers the payment method valid. |
| Electronic check (ECP) | CashBox supports ECP if your payment processor is Chase Paymentech or Litle.<br>For Chase, CashBox validates the ECP payment method by sending the LO verification code to Chase Paymentech, which verifies that the bank-account and routing numbers are valid and that they are not in Chase Paymentech's negative file.<br>(You may also perform a VO validation, which is more costly and involves more thorough checks. Work with Vindicia Client Services to add it to your CashBox configuration.)<br>For Litle, ECP (Litle "eCheck") data will be validated (verifying that the routing number is correctly formatted and that it exists in the Fed database) by CashBox. For `auth` and `authCapture` requests (whether performed directly by the merchant, or automatically by the CashBox rebilling system) an additional verification procedure is performed. This verification compares the ECP account information against a 3rd party database to determine if the account is associated with activities such as fraud, over drafts, or other items determined to be risk factors. If this verification procedure returns negative information, the `auth` or `authCapture` request will be rejected. |
| Merchant Accepted | CashBox does not validate Merchant Accepted payments. |

Table 11-15 Validation Process by Payment Method Type *(Continued)*

| Payment Method Type | Validation Process |
|---|---|
| PayPal | The `PaymentMethod` object methods do not support validation for PayPal. Instead, if you specify PayPal as the payment method while creating an `AutoBill` object, CashBox authorizes a transaction for a small amount (US $1 if the currency is USD). Such an authorization requires that the customer log in to his or her account on the PayPal site and agree to the terms and conditions of recurring billing (reference transaction). That process serves as validation of the PayPal payment method. |
| Token | Validation of the token payment method can occur only if that method is associated with an `Account` object. In that case, CashBox validates by ensuring that the token type (ID) has been previously defined and added to the `Account` object. |

## Input

***paymentMethod:*** the `PaymentMethod` object to create or update. In case of an update, you can identify this object with either its VID or your payment method ID (`merchantPaymentMethodId`).

***validate:*** a Boolean flag that, if set to `true`, causes this method to validate the `PaymentMethod` object first before creating or updating.

When ***validate*** is `true`, the AVS and CVN policies (or, in their absence, the default evaluation policy) are used to determine the status of the validation. If validation fails, the `PaymentMethod` is not updated.

For more detail on AVS and CVN Return Codes, please work with your Vindicia Client Services representative.

***minChargebackProbability:*** a number between 0 and 100 by which you specify your fraud risk score tolerance level. A chargeback probability (also called the risk-screening score or risk score) of 100 indicates that CashBox is 100% certain that a transaction is fraudulent and will result in a chargeback. Specify your acceptable threshold for chargeback possibility with this parameter. If the score evaluates to be more than your tolerance level, the `update` call will fail.

For risk score evaluation, you must specify the ***sourceIp*** parameter, described below, and full billing address containing city, state (district), and country for the payment method.

***replaceOnAllAutoBills:*** a Boolean flag that, if set to `true`, causes this method to propagate the updates to an existing payment method to all the `AutoBill` objects. This operation works only for those payment methods that are already associated with an `Account` object. The default is false, meaning that this method does not update any `AutoBill` objects.

***sourceIp:*** the customer IP address from which the customer specified details for this payment method. It must be specified if you want CashBox to evaluate risk score for this payment method, that is, if you specify `minChargebackProbability` to be less than 100.

***replaceOnAllChildAutoBills:*** a Boolean flag that, if set to `true`, the update will propagate to the AutoBills belonging to children of this account. `If replaceOnAllAutoBills` is set to `false`, this flag is ignored. If `replaceOnAllAutoBills` is set to `true` and `replaceOnAllChildAutoBills` is set to `true`, this will affect only the parent account.

***ignoreAvsPolicy:*** a Boolean flag that, if set to `true`, will override the AVS policy, and update the `paymentMethod`, regardless of the AVS return code. If set to `false` or `null`, (and if ***validate*** is set to `true`) the AVS return code will be used to determine whether to update the `paymentMethod`.

***ignoreCvnPolicy:*** an optional Boolean flag that, if set to `true`, will override the CVN policy, and update the `paymentMethod`, regardless of the CVN return code. If set to `false` or `null`, (and if ***validate*** is set to `true`) the CVN return code will be used to determine whether to update the `paymentMethod`.

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***paymentMethod:*** the `PaymentMethod` object that was created or updated. If the object was newly created, this output contains the object's Vindicia-assigned ID in the `VID` attribute. CashBox masks the account numbers in this object.

***created:*** a Boolean flag that, if set to `true`, indicates that this method has created a new `PaymentMethod` object. A `false` setting indicates that update has updated an existing `PaymentMethod` object, which occurs if a `PaymentMethod` object with the `merchantPaymentMethodId` or `VID` value specified in the input already exists in the Vindicia database.

***validated:*** a Boolean flag that, if set to `true`, indicates that the `update` method has successfully validated the underlying payment method. This is meaningful only if you turned the input ***validate*** flag on.

***score:*** the fraud risk score evaluated by CashBox for this payment method. If you specified `minChargebackProbability` of less than 100, CashBox evaluates the fraud risk score for this payment method.

***scoreCodes:*** an array of code numbers and corresponding explanatory text that explains the score evaluated by CashBox.

***authStatus:*** a `TransactionStatus` object containing information received from the payment processor for the underlying validation transaction. This is available only if you chose to validate the payment method.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `261` | `All active AutoBills were updated. AutoBills which are both expired and Suspended cannot be updated.` |
| `400` | One of the following:<br>• ***Error-description.*** (Returned if CashBox cannot map a `PaymentMethod` object that is passed into a database record.)<br>• `Data validation error Failed to create Payment-Type-Specific Payment Record: Credit Card conversion failed: Credit Card failed Luhn check.`<br>• `Unable to save payment method:` ***error-description*** `.` |
| `402` | `Unable to authorize card.` |
| `407` | `AVS policy evaluation failed.` |
| `408` | `CVN policy evaluation failed.` |
| `409` | `AVS and CVN policy evaluations failed.` |
| `410` | `AVS and CVN policy evaluations could not be performed.` |
| `501` | ***Error-description.*** |

### *Example*

```
// To create a credit card based payment method and validate it.

// Create a payment method object to make the call
$paymentMethod = new PaymentMethod();

$paymentMethod->setType('CreditCard');
$paymentMethod->setAccountHolderName('Jane Doe');
$paymentMethod->setCustomerSpecifiedType('Visa');
$paymentMethod->setCurrency('USD');
$paymentMethod->setActive(true);

$cc = new CreditCard();
$cc->setAccount('411111111111111');
$cc->setExpirationDate('201208');

$paymentMethod->setCreditCard($cc);

// not setting merchantPaymentMethodId. We can use the
// VID returned after creation as unique id for the payment method

$validate = true;
$minChargebackProbability = 100; // not evaluating risk score
$replaceOnAutoBills = false; // just creating the payment method, not
          // attached to an account yet
$ip = null; // not evaluating risk score
$response = $paymentMethod->update($validate,
          $minChargebackProbability,
          $replaceOnAutoBills, $ip);

if($response['returnCode'] == 200 && $response['created']) {
   $retPm = $response['data']->paymentMethod;
   print('Payment method successfully created with VID'
          . $retPm->getVID());
}

else if($response['returnCode'] == 402) {
   print('Payment method is invalid');

}
// check response from the payment processor
$validationTxStatus = $response['authStatus'];
if ($validationTxStatus != null) {
   $creditCardStatus =
       $validationTxStatus->getCreditCardStatus();
   if ($creditCardStatus != null) {
       $authCode = $creditCardStatus->getAuthCode();
       $avsCode = $creditCardStatus->getAuthCode();
       print "Card rejected with code " . $authCode . "\n";
       print "Address verification code " . $avsCode . "\n";
   }
}
```

## validate

The `validate` method validates a `PaymentMethod` object. You call this method on an appropriately populated `PaymentMethod` object. The validation process varies according to the payment method type. See the `update` method for the validation process in the context of the `validate` parameter being passed to the `update()` call.

This call only validates the `PaymentMethod` object but does not create, update, or store the data in CashBox. To create or update the data, call `update()` on the object after validation.

This method considers the Luhn check, the authorization return, and the (merchant defined) active AVS and CVN policy when formulating the validated result.

For more detail on AVS and CVN Return Codes, please work with your Vindicia Client Services representative.

### *Input*

***paymentMethod:*** the `PaymentMethod` to validate.

***sourceIp:*** the customer IP address from which the customer specified details for this payment method. It must be specified if CashBox is to evaluate risk score for this payment method, that is, if you specify `minChargebackProbability` to be less than 100.

***minChargebackProbability:*** a number between 0 and 100 by which you specify your fraud risk score tolerance level. A probability of 100 indicates that CashBox is 100% certain that a transaction is fraudulent and will result in a chargeback. Specify your acceptable threshold for chargeback possibility with this parameter. If the score evaluates to be more than your tolerance level, CashBox will not validate the payment method with your payment processor, saving you the cost of obtaining validation for potentially fraudulent payment methods.

> For risk score evaluation, you must specify the ***sourceIp*** parameter, described below, and the full billing address containing city, state (district), and country for the payment method.

***ignoreAvsPolicy:*** a Boolean flag that, if set to `true`, will override the AVS policy, and update the `paymentMethod`, regardless of the AVS return code. If set to `false` or `null`, (and if ***validatePaymentMethod*** is set to `true`) the AVS return code will be used to determine whether to update the `paymentMethod`.

***ignoreCvnPolicy:*** an optional Boolean flag that, if set to `true`, will override the CVN policy, and update the `paymentMethod`, regardless of the CVN return code. If set to `false` or `null`, (and if ***validatePaymentMethod*** is set to `true`) the CVN return code will be used to determine whether to update the `paymentMethod`.

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***authStatus:*** a `TransactionStatus` object containing information received from the payment processor for the underlying validation transaction processed by your payment processor. If you have enabled risk scoring and if the score evaluates to be more than your tolerance threshold specified in the ***minChargebackProbability*** input parameter, CashBox will not populate this output parameter.

***validated:*** a Boolean flag that, if set to `true`, indicates that this method has successfully validated the `PaymentMethod` object. A `false` setting indicates that the validation failed.

***avsCvnPolicyEvaluationDetails:*** an object of type `AvsCvnPolicyStatus`, and contains two fields, `returnCode` and `returnString`, which pertain to the outcome of the AVS/CVN policy evaluation.

(**Note:** All other methods affected by the AVS/CVN policy return their `returnCode` and `returnString` in the `Return` object from the method.)

***score:*** the fraud risk score evaluated by CashBox for this payment method. If you specified `minChargebackProbability` of less than 100, CashBox will evaluate the risk score for this payment method.

***scoreCodes:*** an array of code numbers and corresponding explanatory text that explains the score evaluated by CashBox

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `201` | `VS/CVN policy not evaluated.`<br>Returned to indicate that the `AutoBill` was created, but the AVS policy was not evaluated, due to a lack of response from the Payment Processor. |
| `400` | One of the following:<br>• `Invalid parameters:` ***error-description.***<br>• ***Error-description.***<br>Returned if CashBox encounters a general error while mapping the object to a CashBox database object. |
| `407` | `AVS policy evaluation failed.` |
| `408` | `CVN policy evaluation failed.` |
| `409` | `AVS and CVN policy evaluations failed.` |
| `410` | `AVS and CVN policy evaluations could not be performed.` |
| `501` | `Validation not implemented for` ***payment-method-type*** `accounts.` |

## *Example*

```
// To validate a credit card based payment method

// Create a payment method object to make the call
$paymentMethod = new PaymentMethod();

$paymentMethod->setType('CreditCard');
$paymentMethod->setAccountHolderName('Jane Doe');
$paymentMethod->setCustomerSpecifiedType('Visa');
$paymentMethod->setCurrency('USD');
$paymentMethod->setActive(true);

$cc = new CreditCard();
$cc->setAccount('411111111111111');
$cc->setExpirationDate('201208');

$paymentMethod->setCreditCard($cc);

// customer's ip address not necessary since we
// do not want to do risk scoring

$sourceIp = null;

// risk score threshold set to 100 since we
// do not want to do risk scoring

$minChargebackProbability = 100;

$response =
   $paymentMethod->validate($sourceIp, $minChargebackProbability);

if($response['returnCode'] == 200) {
   if($response['validated']) {
      print('Payment method is valid');
      // get AVS code
      $txStatus = $response['authStatus'];
      $avsCode = $txStatus->creditCardStatus->avsCode;
      // examine AVS return code here
   }
   else {
      print('Payment method is invalid');
   }
}
else {
   print('Error encountered during validation');
}
```

# 12 The `PaymentProvider` Object

The `PaymentProvider` object serves as a wrapper to contain static information required by a payment provider for payment processing.

## 12.1 PaymentProvider Data Members

The following table lists and describes the data members of the PaymentProvider object.

Table 12-1 PaymentProvider Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| authCurrency-Override | string | The currencies for which authorization currency may be overridden by USD. |
| auth-ExpirationDays | int | The number of days before the payment provider expires authorizations. |
| disputeAddress | Address | The payment provider's dispute address.<br>See Section 3.1: Address Data Members. |
| disputeEmail | string | The payment provider's email address for disputes. |
| disputeUri | anyURI | The payment provider's URI for disputes. |
| name | string | The name of the provider. |
| nameValues | NameValuePair | An optional array of name-value pairs to associate with the payment provider.<br>See Section 10: The NameValuePair Object. |

## 12.2 `PaymentProvider` Methods

The following table summarizes the methods for the `PaymentProvider` object.

Table 12-2 `PaymentProvider` Object Methods

| Method | Description |
|---|---|
| `dataRequest` | Performs a generic query on the `PaymentProvider` object. |
| `fetchByName` | Loads a `PaymentProvider` object by `name`. |

## dataRequest

The `dataRequest` method performs a generic query on a `PaymentProvider` object.

**Note:** CashBox currently supports BOKU for this method.

For more `dataRequest` examples, please see Section 6.3.3: Using CashBox to query BOKU in the ***CashBox Programming Guide***.

### *Input*

***paymentProvider:*** the `PaymentProvider` against which the query will be performed.

***requestType:*** the type of query to be performed. CashBox currently supports the BOKU price and service-price calls.

***requestArguments:*** an array of name/value pairs used to construct the query.

**Note:** The following price/service-price parameters are not allowed (Vindicia will include authentication information for the query): ***merchant-id***, ***password***, ***sig***, and ***timestamp***.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***paymentProvider:*** the `PaymentProvider` object against which the query was performed.

***request:*** the formatted query input in payment provider-native format.

***response:*** the formatted query output in payment provider-native format.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$provider = new paymentProvider();
$rc = $provider->dataRequest('price',
    [
    NameValuePair->new(name => 'reference-currency',
           value => 'USD'),
    NameValuePair->new(name => 'service-id',
           value => '140ba94f2c24e44b5cb85730')
    ]
);
```

## fetchByName

The `fetchByName` method fetches a `PaymentProvider` object by name.

### Input

***name:*** the name of the `PaymentProvider` object.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***paymentProvider:*** the `PaymentProvider` object requested.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$providerName = 'GiganticPicklesRuS';
// Create a SOAP caller object
$provider = new PaymentProvider();
$response = $provider->fetchByName($providerName);
if($response['returnCode'] == 200)
{
   $fetchedProvider = $response['data']->paymentProvider;
   // process fetched paymentProvider here
}
```

# 13 The `Product` Object

A `Product` object represents a product or service available for purchase on your site. `Product` objects contain a description of the product, its entitlements, and a default price.

A `Product` may be a single item, or may be a bundled collection of multiple products. For example, a `Product` may be a monthly magazine subscription, or may include a monthly subscription, a new customer gift, and a one-time purchase.

Use `Product` objects to:

- Define a product, including its default price, and its associated Entitlements. (See also Section 3.1: Creating Products in the ***CashBox Programming Guide***.)
- Define a bundled product, using pre-existing Product objects to create a new group of products, made available with its own Entitlements and default price. (See also Section 3.2: Creating Bundled Products in the ***CashBox Programming Guide***.)
- Create a Tokens for Cash system, in which customers may purchase Products which grant their Account Token credits. Tokens may be used as currency in proprietary transaction systems (such as the purchase of a sword in an online game), or may be used to allocate minutes in time-based transactions (for use in phone contracts, or website access). (See also Chapter 10: Working with Tokens in the ***CashBox Programming Guide***.)

# 13.1 `Product` Data Members

The following table lists and describes the data members of the `Product` object.

Table 13-1 `Product` Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| `billingStatement-Identifier` | string | **Optional**. The transaction description on the customer's billing statement that is sent by the bank when the customer is charged through this `Product` object. This field's value and format are set by your payment processor; consult with Vindicia Client Services before setting the value.<br>If GlobalCollect, Chase Paymentech, MeS, or Litle is your payment processor, see Appendix A: Custom Billing Statement Identifier Requirements in the ***CashBox Programming Guide***.<br>**Note:** If this identifier is also defined in a `BillingPlan` object associated with the `AutoBill` object for this `Product` object, the billing statement identifier on `BillingPlan` takes precedence. |
| `bundledProducts` | `Product` | Zero or more products "bundled" or grouped with this Product. |
| `creditGranted` | `Credit` | The credit(s) to be granted upon purchase of this Product.<br>See the `Credit` Subobject. |
| `defaultBillingPlan` | `BillingPlan` | **Optional.** Recurring pricing is governed by this attribute if a billing plan is not explicitly associated with the `AutoBill` object for this `Product` object. |
| `defaultRatePlan` | `RatePlan` | **Optional.** A default Rate Plan for the Product. |
| `descriptions` | `ProductDescription` | **Optional.** Zero or more language/product description pairs.<br>**Note:** In the absence of a product description, the `merchant-ProductId` will be used. |
| `endOfLifeTimestamp` | `dateTime` | **Optional.** A timestamp that specifies the expiration date for this `Product` object. Use this attribute to filter your product list, and present only those products with future expiration dates as currently available for subscription.<br>(This attribute is for your information only, and does not affect CashBox operations.) |

Table 13-1 `Product` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `merchant-EntitlementIds` | `MerchantEntitle-mentId` | An array of identifiers that determine the customer's entitlements. Use these IDs within your application to grant access to products or services. These IDs are returned to you inside `Entitlement` objects along with the dates until which they are valid for a given customer. The status of a customer's `AutoBill` object with which this product is associated determines the date until which the `Entitlement` objects are valid.<br>**Note:** `Entitlements` are available to customers through `Product`, `BillingPlan`, and `Account` objects. When adding `Products` to an `AutoBill`, `Entitlements` are cumulative, unless otherwise defined.<br>See Section 8.1: Entitlement Data Members. |
| `merchantProductId` | string | Your unique identifier for the product. If you track your products internally by SKU, use the SKU as your `merchantProductId`, to allow you to map your local records to CashBox Transactions that have this `Product` as a line item. |
| `nameValues` | `NameValuePair` | **Optional.** An array of name–value pairs, each of which enables you to add additional product information, not included in this `Product` object's other attributes.<br>See Section 10: The NameValuePair Object. |
| `prices` | `ProductPrice` | An array of `ProductPrice` objects, one per currency (or `Token`) code.<br>See the `ProductPrice` Subobject. |
| `status` | `ProductStatus` | An enumerated string value that describes the current status of the `Product` object. See the `ProductStatus` Subobject for the values.<br>For example, use this value to determine whether to make a `Product` object available for subscription purchase.<br>(This attribute is for your information only, and does not affect CashBox operations.)<br>See the `ProductStatus` Subobject. |
| `taxClassification` | string | A string that defines your tax classification for this `Product`. |
| `VID` | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `Product` object, leave this field blank; it will be automatically populated by CashBox. |

# 13.2 `Product` Subobjects

The `Product` object has several subobjects:

- `ProductDescription` Subobject
- `ProductPrice` Subobject
- `ProductStatus` Subobject

## `ProductDescription` Subobject

Defines a language/product description pair.

Table 13-2 `ProductDescription` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `description` | string | A description of this product written in *language*. A free-form string of less than 256 characters. |
| `language` | string | The language in which the product description is written. |

## `ProductPrice` Subobject

Lists a currency and/or Token value for the product.

Table 13-3 `ProductPrice` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `amount` | decimal | Value of the currency amount. |
| `currency` | string | ISO 4217 currency code to be used for this `ProductPrice` Amount. Defaults to USD. |
| `token` | `Token` | Details of pricing using tokens. |

## `ProductStatus` Subobject

Defines whether the product is `Active` or `Suspended`. `Suspended` products may not be renewed through `AutoBills`.

Table 13-4 `ProductStatus` Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| `Active` | string | `Product` is currently active (available to the customer). |
| `Suspended` | string | `Product` is inactive (unavailable to the customer), a state that cannot be renewed. Customers must start a new purchase process and reorder a suspended product as a brand-new billing plan. |

## 13.3 `Product` Methods

The following table summarizes the methods for the `Product` object.

Table 13-5 `Product` Object Methods

| Method | Description |
|---|---|
| `fetchAll` | Returns all the `Product` objects. |
| `fetchByAccount` | Returns one or more `Product` objects whose `Account` object matches the input. |
| `fetchByMerchantEntitlementId` | Returns all the `Product` objects whose entitlement ID assigned by you (`merchantEntitlementId`) matches the input. |
| `fetchByMerchantProductId` | Returns the `Product` object whose `merchantProductId` matches the input. |
| `fetchByVid` | Returns a `Product` object whose VID matches the input. |
| `update` | Creates or updates a `Product` object. |

## fetchAll

The `fetchAll` method returns all the `Product` objects.

This method supports paging to limit the number of records returned per call. Returning a large number of records in one call may swamp buffers, and might cause a failure. Vindicia recommends that you call this method in a loop, incrementing the page for each loop iteration with an optimal page size (number of records returned in one call) until the page contains a number of records that is less than the given page size.

### Input

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***products:*** an array of returned `Product` objects.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 404 | No products found for merchant. |

### Example

```
$product = new Product();
$page = 0;
$pageSize = 10;
do {
    $ret = $product->fetchAll($page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedProducts = $ret['products'];
        if ($fetchedProducts != null) {
            $count = sizeof($fetchedProducts);
            foreach ($fetchedProducts as $prod) {

                 // process a fetched product here …
                 $page++;
            }
        }
    } while ($count > 0);
}
```

## fetchByAccount

The `fetchByAccount` method returns one or more `Product` objects to which the `Account` object specified in the input is subscribed. That is, this method returns all the `Product` objects that are associated with the `AutoBill` objects that are also associated with the specified `Account` object.

### Input

***account:*** the `Account` object that serves as the search criterion. Use the `merchantAccountId` or `VID` to identify the object.

***includeChildren:*** an optional Boolean flag that, if set to `true`, includes all children associated with this `Account`. If this flag is omitted, CashBox will interpret it as `false`, and will not include children in the query.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***products:*** an array of one or more `Product` objects associated with the `AutoBill` objects that are also associated with the `Account` object specified in the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `404` | One of the following:<br>• `Unable to load account to search by:` ***error-description.***<br>• `Unable to load account by: No matches.` |

### Example

```
// Create a SOAP caller product object
$prod = new Product();

// Create an Account object to fetch products by
$acct = new Account();
$acct->setMerchantAccountId('jdoe101');

$response = $prod->fetchByAccount($acct);

if($response['returnCode'] == 200) {
    $fetchedProducts = $response['data']->products;

    // process fetched products here
    if ($fetchedProducts != null) {
        foreach ($fetchedProducts as $fetchedProd) {
            // process a fetched product here
        }
    }
}
```

## fetchByMerchantEntitlementId

The `fetchByMerchantEntitlementId` method returns one or more `Product` objects whose entitlement ID assigned by you (`merchantEntitlementId`) matches the input. For example, call this method in response to a customer request for a list of all your products that offer a certain privilege on your site.

### Input

***merchantEntitlementId:*** your entitlement ID (`merchantEntitlementId`), which serves as the search criterion.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***products:*** an array of one or more `Product` objects whose entitlement ID assigned by you (`merchantEntitlementId`) matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | `Must specify entitlement id.` |
| `404` | `Could not load product for entitlement id` ***input-merchant-EntitlementId.*** |

### Example

```
// Create a SOAP caller product object
$prod = new Product();

$response = $prod->fetchByMerchantEntitlementId
          ('PremiumVideoContentAccess');

if($response['returnCode'] == 200) {
   $fetchedProducts = $response['data']->products;

   // process fetched products here
   if ($fetchedProducts != null) {
      foreach ($fetchedProducts as $fetchedProd) {
         // process a fetched product here
      }
   }
}
```

## fetchByMerchantProductId

The `fetchByMerchantProductId` method returns the `Product` object whose product ID assigned by you (`merchantProductId`) matches the input.

### Input

***merchantProductId:*** your product ID (`merchantProductId`), which serves as the search criterion.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***product:*** the `Product` object whose `merchantProductId` matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load product by sku` ***input-merchantProductId:*** `No match.`<br>• `Unable to load product by merchantProductId` ***input-merchantProductId: error-description.***<br>• `Must specify merchantProductId to load by!` |

### Example

```
$prodMerchantId = '5w3320dj';

// Create a SOAP caller product object
$prod = new Product();

$response = $prod->fetchByMerchantProductId($prodMerchantId);

if($response['returnCode'] == 200) {
    $fetchedProduct = $response['data']->product;
    // process fetched product here
}
```

## fetchByVid

The `fetchByVid` method returns a `Product` object whose VID matches the input.

### Input

***vid:*** the `Product` object's Vindicia unique identifier, which serves as the search criterion.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***product:*** the `Product` object whose VID matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load product by VID` ***input-vid: error-description.***<br>• `Must specify VID to load by!` |
| `404` | `Unable to load product by VID` ***input-vid:*** `No match.` |

### Example

```
$prodVid = '079e770ca81ab5f4cd40a2dec6d4c72832ce8dd0';

// Create a SOAP caller product object
$prod = new Product();

$response = $prod->fetchByVid($prodVid);

if($response['returnCode'] == 200) {
    $fetchedProduct = $response['data']->product;
    // process fetched product here
}
```

## update

The `update` method creates or updates a `Product` object.

To create a `Product` object, initialize the object and set the values for its data members, as appropriate, and then call the `update()` method to store the changes. During the process, do not set a value for `VID` because CashBox automatically generates that when you call `update()`. When updating an existing `Product` object, identify it with its VID or your product ID (`merchantProductId`).

Because products are typically stable company offerings, and are updated or created only rarely, `Products` are usually created using the CashBox Portal, rather than the API.

### *Input*

***product:*** the `Product` object to create or update. Identify this object using either its VID or your product ID (`merchantProductId`).

***duplicateBehavior:*** an enumerated string that is currently **not** supported by CashBox.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***product:*** the `Product` object that was created or updated.

***created:*** a Boolean flag that, if set to `true`, indicates that this method has created a new `Product` object. A `false` setting indicates that `update` has updated an existing `Product` object.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• ***Error-description.***<br>• `CashBox encountered a general error while mapping the input Product object to the CashBox database.`<br>• `Unable to save product:` ***error-description.***<br>• `Unable to retrieve saved product:` ***error-description.*** |

### *Example*

```
// to create a new product object

$product = new Product();

// Identify the product by your unique identifier, etc.
$product->setMerchantProductId('gold12345');
$product->setStatus('Active');
$product->setDescription('Premium Video Access');

$meId = new MerchantEntitlementId();
$meId->setId('PremiumVideoAccess2010');
$meId->setDescription('Premium video access privilege for 2010');

$product->setMerchantEntitlementIds(array($meId));

$response = $product->update(DuplicateBehavior::SucceedIgnore);

if($response['returnCode'] == 200 && $response['created']) {
   $createdProduct = $response['data']->product;
   print "Created product with VID " . $createdProduct->getVID();
}
```

# 14 The `RatePlan` Object

The `RatePlan` object defines the logic by which the pricing structure for Rated Products will be determined.

# 14.1 RatePlan Data Members

The following table lists and describes the data members of the RatePlan object.

Table 14-1 RatePlan Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| description | string | **Optional.** A description for the Rate Plan. |
| hasEvent-Recorded | Boolean | Read-only field that indicates whether or not this RatePlan has had any Events recorded against it. |
| includedUnits | decimal | The number of Rated Units automatically included with each Billing Cycle. |
| maximumFee | RatePlanPrice | An array of Prices for the maximum charge for this plan, per Billing Cycle. If this field is defined, customers will never be charged more than this amount per Billing Cycle, regardless of their reported use.<br>The RatePlanPrice object is an (amount, currency) pair used in a RatePlan, and contains two members:<br>amount: the number of currency units.<br>currency: the ISO 4217 currency code to be used for this Fee. |
| merchantRate-PlanId | string | **Required.** Your unique ID for this Rate Plan. |
| minimumFee | RatePlanPrice | An array of Prices for the minimum charge for this plan, per Billing Cycle. If this field is defined, customers will be charged at least this amount per Billing Cycle, regardless of their use.<br>The RatePlanPrice object is an (amount, currency) pair used in a RatePlan, and contains two members:<br>amount: the number of currency units.<br>currency: the ISO 4217 currency code to be used for this Fee. |

Table 14-1 `RatePlan` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `multiplyRated-UnitsBy` | `MultiplyRated-UnitsBy` | The calculation method by which this `RatePlan` will determine the price to bill for a Billing Cycle.<br>The `MultiplyRatedUnitsBy` object contains two values:<br>`EachRespectiveTier`: multiplies the number of `Events` for the Billing Cycle by the `ratePrice` for the Tier in which they occurred.<br>`HighestApplicableTier`: multiplies the total number of `Events` for the Billing Cycle by the price for the highest Tier in which any reported Event occurred.<br>`EachRespectiveTier` calculates the charge by Tier use. That is, if a Tiered Plan is defined as $2 for 0-9 units, and $1 for 10-100 units, a customer who uses 15 units will be charged $2*9 + $1*6 = $24.<br>For the same use, `HighestApplicableTier` would calculate the charge by multiplying 15 units by $1, for a total charge of $15. |
| `nameValues` | `NameValuePair` | **Optional.** An array of name-value pair items specific to this `RatePlan`.<br>See Section 10: The NameValuePair Object. |
| `ratedUnit` | `RatedUnit` | Required. The names for the `Unit` included in this Rate Plan.<br>The `RatedUnit` object contains two data members:<br>`nameSingular` (string): defines the singular name for the Unit, as displayed in CashBox pages, reports, and customer emails.<br>`namePlural` (string): defines the plural name for the Unit. |
| `ratePlanModel` | `RatePlanModel` | **Required.** Defines the mode of use for the `RatePlan`.<br>The `RatePlanModel` object contains one of two values:<br>`UsageBased`: calculates the fee per Billing Cycle based on the number of Rated Units consumed during the Billing Cycle.<br>`LicenseBased`: calculates the fee per Billing Cycle based on a defined number of licenses per Billing Cycle.<br>**Note:** If no new Events are reported for a Billing Cycle, `LicenseBased` `AutoBillItems` will repeat the previous Billing Cycle's Use level; `UsageBased` `AutoBillItems` will be reset to zero. |

Table 14-1 `RatePlan` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `rounding-Decimals` | integer | Defines the rounding logic for returned Unit values.<br>Enter the decimal place to which you wish returned values to be rounded. Positive numbers round to the right of the decimal point; negative numbers round to the left of the decimal point. For example, given a return of 346.26961:<br>**0:** rounds to the nearest integer. (346)<br>**2:** rounds to the nearest hundredth. (346.27)<br>**-2:** rounds to the nearest hundred. (300) |
| `status` | `RatePlanStatus` | Defines the status of the `RatePlan`:<br>`Active`: the Rate Plan is available for use.<br>`Suspended`: the Rate Plan is not available for use. |
| `tier` | `RatePlanTier` | An array of pricing levels used in the Rate Plan.<br>See the `RatePlanTier` Subobject. |
| `VID` | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `RatePlan` object, leave this field blank; it will be automatically populated by CashBox. |

# 14.2 RatePlan Subobjects

The RatePlan object has three subobjects:

- Event Subobject
- RatedUnitSummary Subobject
- RatePlanTier Subobject

## Event Subobject

Defines a single reported Rate Plan Event, including the timestamp and billing status, and the Account, AutoBill, AutoBillItem, or Product with which the Event is associated.

Events are associated with AutoBillItems, in that a single AutoBillItem may contain an array of Events, but each Event is contained in only one AutoBillItem.

When defining an Event, associate it with a single, unique AutoBillItem. The AutoBillItem may be identified using any combination of the following objects' identifiers: Account, AutoBill, AutoBillItem, or Product. CashBox requires that at least one of the following three data members be specified: Account, AutoBill, or AutoBillItem.

When an Event object is returned, CashBox will populate both the VID and the ID for each of the four objects listed above, from the information contained in the database for the specified AutoBillItem.

If, when reporting an Event, more than one AutoBillItem fits the description, CashBox will return an error.

Table 14-2 Event Object Values

| Value | Data Type | Description |
|---|---|---|
| accountVid | string | Lists the VID for the Account associated with the Event. (Returned for Event fetches.) |
| amount | decimal | The number of Rated Units (as defined by the Rate Plan) used by this Event . |
| autoBillItem-Vid | string | Vindicia's unique name (VID) for the AutoBillItem.<br>Either merchantAutoBillItemId or autoBill-ItemVid must be defined for each Event. |
| autoBillVid | string | Vindicia's unique name (VID) for the AutoBill . |
| billedStatus | BilledStatus | A read-only object of type BilledStatus, which describes whether the Event has been billed, and which includes one of two values:<br>Billed: the Event has been included in a Billing Statement.<br>Unbilled: the Event has not yet been included in a Billing Statement. |

Table 14-2 Event Object Values *(Continued)*

| Value | Data Type | Description |
|---|---|---|
| `dateReceived` | dateTime | A read-only field which lists the date/time that CashBox received the Event. |
| `description` | string | **Optional.** A description of the `Event`. |
| `merchantAccountId` | string | Lists the `merchantAccountId` associated with the `Event`. (Returned for `Event` fetches.) |
| `merchantAutoBillId` | string | Your unique ID for the `AutoBill` associated with the `Event`. |
| `merchantAutoBillItemId` | string | Your unique ID for the `AutoBillItem` associated with the `Event`.<br>Either `merchantAutoBillItemId` or `autoBillItemVid` must be defined for each `Event`. |
| `merchantProductId` | string | Your unique ID for the `Product` associated with the `Event`. |
| `merchantEventId` | string | **Optional.** Your unique ID for the `Event`. Each `merchantEventId` must be unique. If omitted, CashBox will automatically populate this field. |
| `nameValues` | `NameValuePair` | **Optional.** An array of name-value pair items specific to this `Event`.<br>See Section 10: The NameValuePair Object. |
| `productVid` | string | Vindicia's unique name (VID) for the `Product`. |
| `eventDate` | dateTime | The date/time that the `Event` was (or will be) considered billable. By default, this field is populated with the date/time from `dateReceived`. Enter a different date if necessary. |
| *`recordMethod`* | *`EventRecordMethod`* | *(This data member is not in use.)* |
| `VID` | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `Event` object, leave this field blank; it will be automatically populated by CashBox. |

## RatedUnitSummary Subobject

Provides a (temporary) summary of Event charges for a single rated AutoBillItem, including related information about the Item to which these charges refer.

> **Note:** This object does not have an associated VID because it is temporary, created specifically for the fetch call that requests it. The RatedUnitSummary object is not permanently written to the database, and therefore does not require a VID.

Table 14-3 RatedUnitSummary Object Values

| Value | Data Type | Description |
|---|---|---|
| accountVid | string | Vindicia's unique identifier for the Account.<br>(Returned for Event fetches.) |
| autoBillItem-Vid | string | Vindicia's unique identifier for the AutoBillItem. |
| autoBillVid | string | Vindicia's unique identifier for the AutoBill. |
| currentTier | string | The Rate Plan Tier to which the summary refers. (The top Tier for which an Event is recorded at the moment of the query.) |
| currentTotal-RatedUnitsBill | decimal | The total current charge for the Billing Cycle, in the currency specified on the AutoBill. |
| eventCount | int | The number of Events included in this summary for this AutoBillItem. |
| merchantAc-countId | string | Your unique identifier for the Account.<br>This is a read-only field. (Returned for RatedUnit-Summary fetches.) |
| merchantAuto-BillId | string | Your unique identifier for the AutoBill. |
| merchantAuto-BillItemId | string | Your unique identifier for the AutoBillItem. |
| merchantPro-ductId | string | Your unique identifier for the Product. |
| merchantRate-PlanId | string | Your unique identifier for the RatePlan. |
| productVid | string | Vindicia's unique identifier for the Product. |
| ratedUnit | ratedUnit | The name of the Rated Unit for which the summary is returned. |

Table 14-3 `RatedUnitSummary` Object Values *(Continued)*

| Value | Data Type | Description |
|---|---|---|
| `ratedUnitTotal` | decimal | The total number of all Rated Units included in this summary. |
| `ratePlanVid` | string | Vindicia's unique identifier for the `RatePlan`. |

## `RatePlanTier` Subobject

The `RatePlanTier` object describes a single Tier of a `RatePlan`, including its price, whether to charge by individual Unit or by stepped Tier Price, and the lower limit of the Tier.

Table 14-4 `RatePlanTier` Object Values

| Value | Data Type | Description |
|---|---|---|
| `name` | string | **Required.** The descriptive name for the Tier. |
| `ratePrice` | `RatePlanPrice` | **Required.** An array of `RatePlanPrice` objects, which define the Price (or prices) for this Tier (one price for each currency used).<br>The `RatePlanPrice` object is an (amount, currency) pair, which contains two data members:<br>`amount`**:** the number of currency units.<br>`currency`: the ISO 4217 currency code to be used for this `ratePrice`. |
| `chargeCustomer` | `ChargeCustomer` | An object of type `ChargeCustomer`, which may be one of two types:<br>`FlatFee`: charges the customer a defined price per Tier.<br>`PerUnit`: charges the customer a defined price per Rated Unit.<br>`FlatFee` defines a stepped pricing structure, in which the customer is charged the `ratePrice` per Tier.<br>`PerUnit` defines a graduated pricing structure, in which the customer is charged the number of units accessed, multiplied by the `ratePrice` per Tier. |
| `beginsAtLevel` | decimal | The number of Units at which this Tier's pricing structure takes effect. The number of Units defined for each Tier runs from the minimum value of the Tier, to one `Unit` less than the minimum value of the next higher Tier.<br>Typically the first tier would have `beginsAtLevel` = 1. The final, highest tier is unbounded (infinite). |

## 14.3 RatePlan Methods

The following table summarizes the methods for the RatePlan object.

Table 14-5 RatePlan Object Methods

| Method | Description |
|---|---|
| deductEvent | Deducts Events from the unbilled Unit balance. |
| fetchAll | Fetches all existing Rate Plans. |
| fetchByMerchantRatePlanId | Fetches an existing RatePlan by its merchantRatePlanId. |
| fetchByVid | Fetches an existing RatePlan by its VID. |
| fetchEventById | Fetches an Event by its merchantEventId. |
| fetchEventByVid | Fetches an Event by its VID. |
| fetchEvents | Fetches all Events by the specified Account, AutoBill, RatePlan, or Product.<br>If none of these are specified, fetches all Events. |
| fetchUnbilledEvents | Returns unbilled Events for the input AutoBill, Account, Product, or RatePlan.<br>If none of these are specified, fetches all unbilled Events. |
| fetchUnbilledRatedUnits-Total | Returns an array of RatedUnitSummary objects, broken out by AutoBillItem. |
| recordEvent | Records Events against a defined Account, AutoBill, or AutoBillItem. |
| reverseEvent | Reverses one or more existing unbilled Events. |
| *update* | *(Vindicia best practices recommendation is to use the CashBox GUI interface, rather than the API, to create or update a RatePlan.)* |

## deductEvent

The `deductEvent` method reduces a customer's unbilled Event balance.

Use this method to pass in a number of Events to subtract from a customer's balance. To pass in a specific, existing Event, use `reverseEvent`.

> **Note:** This method may not be used against a billed `Event`.

### *Input*

***event:*** an array of `Event` objects.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$rateplan = new RatePlan;
$event = new Event;

$event->setMerchantEventId('rating_129');
$event->setMerchantAutoBillId('ab_715');
$event->setAmount(2);

$response = $rateplan->deductEvent(array($event));
// check $response
```

## fetchAll

The `fetchAll` method returns all available `RatePlan` objects.

### Input

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize***: the number of records to display per page per call. This value must be greater than 0.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***ratePlans:*** an array of returned `RatePlan` objects.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Must specify page and pageSize! |

### Example

```
$rp = new RatePlan();
$page = 0;
$pageSize = 10;
do {
    $ret = $rp->fetchAll($page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedPlans = $ret['ratePlans'];
        $count = sizeof($fetchedPlans);
        foreach ($fetchedPlans as $plan) {
            // process a fetched plan here …
        }
    $page++;
    }
} while ($count > 0);
```

## fetchByMerchantRatePlanId

The `fetchByMerchantRatePlanId` method fetches an existing `RatePlan` by its `merchantRatePlanId`.

### Input

***merchantRatePlanId:*** your Rate Plan ID (`merchantRatePlanId`), which serves as the search criterion. (Optional.)

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***ratePlan:*** the specified `RatePlan`. (Optional.)

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$rateplan = new RatePlan;
$response = $rateplan->fetchByMerchantRatePlanId('rp_46');
if ($response['returnCode'] == 200) {
   $fetchedRatePlan = $response['data']->ratePlan;
   // process fetched RatePlan here
}
```

## fetchByVid

The `fetchByVid` method fetches an existing `RatePlan` by its `VID`.

***Input***

***vid:*** the Vindicia ID for the `RatePlan` you wish to fetch.

***Output***

***return:*** an object of type `Return` that indicates the success or failure of the call.

***ratePlan:*** the returned `RatePlan`. (Optional.)

***Returns***

This method returns the codes listed in Table 1: Standard Return Codes.

***Example***

```
$rateplan = new RatePlan;
$response =
          $rateplan->fetchByVid('2a99928a749ac05cdc8041aee3cacedcb4b6962e');
if ($response['returnCode'] == 200) {
   fetchedRatePlan = $response['data']->ratePlan;
   // process fetched RatePlan here
}
// check $response
```

## fetchEventById

The `fetchEventById` method returns the `Event` for the input `merchantEventId`.

### *Input*

***merchantEventId:*** your Event ID (`merchantEventId`), which serves as the search criterion.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***event:*** the returned `Event`. (Optional.)

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$event = new Event;
$response = $event->fetchEventById('rating_129');

if ($response['returnCode'] == 200) {
    $fetchedEvent = $response['data']->event;
    // process fetched event here
}
```

## fetchEventByVid

The `fetchEventByVid` method returns the `Event` for the input `VID`.

### *Input*

***vid:*** the `Event`'s VID, which serves as the search criterion.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***event:*** the returned `Event`. (Optional.)

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$event = new Event;

$response = $event->fetchEventByVid(
    'a5cfac6ef6da4a3b49a89011e98d5a9731104c63');

if ($response['returnCode'] == 200) {
    $fetchedEvent = $response['data']->event;
       // process fetched event here
}
```

## fetchEvents

The `fetchEvents` method returns all `Events` for the specified `Account`, `AutoBill`, `Product`, or `RatePlan`.

If no input parameters are specified, this call will return the first 50 of ***ALL*** `Events` in your CashBox system. (Default ***pageSize*** is 50.)

### *Input*

***account:*** the `Account` for which Events should be fetched.

***autobill:*** the `AutoBill` for which Events should be fetched.

***product:*** the `Product` for which Events should be fetched.

***ratePlan:*** the `RatePlan` for which Events should be fetched. (Optional.)

***startTimestamp:*** the starting timestamp (lower limit) for the range of `Events` you wish to retrieve.

***endTimestamp:*** the ending timestamp (upper limit) for the range of `Events` you wish to retrieve.

***page:*** (optional) the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** (optional) the number of records to display per page per call. This value must be greater than 0.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***event:*** an array of `Event` objects that match the input constraints.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
// For a specific product on a specific AutoBill
// fetch all Events on all RatePlans
// within a specific date range.

$rateplan = new RatePlan;
$response = $rateplan->fetchEvents(
   null,         # account
   $myAutoBill,  #
   $myProduct,   #
   null,         # ratePlan
   '2012-03-01', # start
   '2012-03-31', # end
   0,            # page
   50,           # pageSize
);
if ($response['returnCode'] == 200) {
   $events = $response->['data']->event;
   foreach ($events as $ev) {
      print $ev->amount;
      print $ev->description;
      print $ev->eventDate;
      print $ev->billedStatus;
      print $ev->VID;
   }
}
```

## fetchUnbilledEvents

The fetchUnbilledEvents method returns the Events for the specified Account, AutoBill, Product, RatePlan, or combination thereof, for which the Account has not yet been billed.

This method returns an array of Events. For example, if you specify the AccountVid for the query, your return will be an array of Events, one for each rated AutoBillItem listed for the Account.

If no input parameters are specified, this call will return the first 50 of ***ALL*** Events in your CashBox system. (Default ***pageSize*** is 50.)

### *Input*

***account:*** the Account for which Events should be fetched.

***autobill:*** the AutoBill for which Events should be fetched.

***product:*** the Product for which Events should be fetched.

***ratePlan:*** the RatePlan for which Events should be fetched.

***startTimestamp:*** the starting timestamp (lower limit) for the range of Events you wish to retrieve.

***endTimestamp:*** the ending timestamp (upper limit) for the range of Events you wish to retrieve.

***page:*** (optional) the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** (optional) the number of records to display per page per call. This value must be greater than 0.

### *Output*

***return:*** an object of type Return that indicates the success or failure of the call.

***event:*** the array of specified Event objects.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
// For a specific product on a specific AutoBill
// fetch all Unbilled Events on all RatePlans
// within a specific date range.

$rateplan = new RatePlan;
$response = $rateplan->fetchUnbilledEvents(
   null,         # account
   $myAutoBill,  #
   $myProduct,   #
   null,         # ratePlan
   '2012-03-01', # start
   '2012-03-31', # end
   0,            # page
   50,           # pageSize
);
if ($response['returnCode'] == 200) {
   $events = $response->['data']->event;
   foreach ($events as $ev) {
      print $ev->amount;
      print $ev->description;
      print $ev->eventDate;
      print $ev->billedStatus;
      print $ev->VID;
   }
}
```

## fetchUnbilledRatedUnitsTotal

The `fetchUnbilledRatedUnitsTotal` method returns the total number and currency value for the specified unbilled Events.

> **Note:** If no input parameters are specified, this method will return the total for all unbilled `Events` in your CashBox system. Specifying any of the input parameters is additive, in that you may specify any combination of listed parameters to narrow your return.

### *Input*

***account:*** the `Account` for which `Events` should be fetched.

***autobill:*** the `AutoBill` for which `Events` should be fetched.

***product:*** the `Product` for which `Events` should be fetched.

***ratePlan:*** the `RatePlan` for which `Events` should be fetched.

***startTimestamp:*** the starting timestamp (lower limit) for the range of `Events` you wish to retrieve. (Optional.)

***endTimestamp:*** the ending timestamp (upper limit) for the range of `Events` you wish to retrieve. (Optional.)

***page:*** (optional) the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** (optional) the number of records to display per page per call. This value must be greater than 0.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***ratedUnitSummary:*** the array of specified `RatedUnitSummary` objects, broken out by `AutoBillItem`.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
// For a specific product on a specific AutoBill
// fetch the array of ratedUnitSummary objects
// for all RatePlans within a specific date range.

$rateplan = new RatePlan;
$response = $rateplan->fetchUnbilledRatedUnitsTotal(
   null,         # account
   $myAutoBill,  #
   $myProduct,   #
   null,         # ratePlan
   '2012-03-01', # start
   '2012-03-31', # end
   0,            # page
   50,           # pageSize
);
if ($response['returnCode'] == 200) {
   $summaries = $response->['data']->ratedUnitSummary;
   foreach ($summaries as $sum) {
      print $sum->ratedUnitTotal;
      print $sum->currentTotalRatedUnitsBill;
   }
}
```

## recordEvent

The `recordEvent` method records `Events` against a defined `Account`, `AutoBill`, or `AutoBillItem`.

`recordEvent` will return an error if you attempt to pass in a negative amount.

> **Note:** This method is a bulk interface, which allows up to 50 `Events` to be recorded in a single call.

### *Input*

***event:*** the array of `Event` objects that you wish to record.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$rateplan = new RatePlan;
$event = new Event;

$event->setMerchantEventId('rating_129');
$event->setMerchantAutoBillId('ab_715');
$event->setAmount(2);

$response = $rateplan->recordEvent(array($event));
// check $response
```

## reverseEvent

The `reverseEvent` method reverses an unbilled `Event`.

Use this method to reverse a specific `Event`. To simply subtract unbilled Units from a customer's balance, use `deductEvent`.

> **Note:** This method may not be used against a billed `Event`.

### *Input*

***event:*** the array of `Event` objects you wish to reverse.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$rateplan = new RatePlan;

$event = new Event;
$event->setMerchantEventId('rating_129');

$response = $rateplan->reverseEvent(array($event));
// check $response
```

# 15 The `Refund` Object

The `Refund` object encapsulates the data on the funds that you returned to a customer for a previously conducted Transaction, in which the customer paid you for a product or service.

The `Refund` object in CashBox may be generated in one of two ways:

- A `Refund` may be issued through CashBox, to reverse a one-time or recurring Transaction, using either the CashBox API or Portal.
- A refund may be created to report a transaction that occurred outside the CashBox system, to allow Vindicia's ChargeGuard team to effectively dispute a chargeback against the transaction for which you issued a refund. In this case, the refund was issued (and the original transaction might have occurred) outside of CashBox.

# 15.1 Refund Data Members

The following table lists and describes the data members of the Refund object.

Table 15-1 Refund Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| amount | decimal | A decimal representation of a monetary amount for the refund. Even though amount is a financial unit, its actual value and meaning depend on the value you set in the currency data member. This amount must not exceed that on the Transaction for which you are issuing this refund. |
| credit | Credit | A credit(s) reversed as part of the refund. This is a read-only field.<br>See the Credit Subobject. |
| currency | string | The ISO 4217 currency code (see www.xe.com/iso4217.htm) for this transaction. |
| merchantRefundId | string | A string of a maximum of 255 characters that represents your unique identifier for this Refund object. For refunds issued through the CashBox Portal, CashBox automatically generates this ID, with a prefix provided Vindicia when CashBox was initially configured for your company. Vindicia recommends that you use a different prefix for this ID to avoid collision with CashBox-generated IDs. |
| note | string | An optional memo regarding the refund. |
| referenceString | string | The data returned from the payment processor, such as the latter's ID for the refund. This field is only for refunds that are processed outside of CashBox, and that are reported to Vindicia for chargeback processing only. For refunds processed through CashBox, leave this field blank. |
| timestamp | dateTime | A timestamp that specifies the date and time of the refund. For refunds processed through CashBox, leave this field blank. CashBox will fill it in the Refund object returned to you in response to a fetch call or the perform() call. |
| tokenAction | RefundTokenAction | The CashBox action for handling the Token grant when processing the refund. Specify this attribute when issuing a refund for a Transaction that granted Tokens to a customer's Account.<br>See the RefundTokenAction Subobject. |

Table 15-1 `Refund` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `transaction` | `Transaction` | The original Transaction to which this refund applies, which must have been successfully captured through CashBox.<br>To process this Refund through CashBox, populate this field with the VID or your transaction ID (`merchantTransactionId`) to identify the transaction.<br>If you are reporting the refund to Vindicia for chargeback processing only, and have already reported this transaction, identify it with the `merchantTransactionId`. If you have not yet reported the Transaction with this ID, CashBox creates a stub `Transaction` object that contains only the `merchantTransactionId` value, with the assumption that the Transaction information will be completed at a later date.<br>See Section 18.1: Transaction Data Members. |
| `VID` | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `Refund` object, leave this field blank; it will be automatically populated by CashBox. |

## 15.2 `Refund` Subobject

The `Refund` object has one subobject: the `RefundTokenAction` Subobject.

### `RefundTokenAction` Subobject

Describes the action taken on a Transaction refund, which caused a customer to be granted or to receive tokens.

Table 15-2 `RefundTokenAction` Object Values

| Value | Description |
|---|---|
| `CancelNegative-Balance` | Reverses the token grants made by the Transaction that is being refunded. If this action causes the Token balance to drop below zero, subsequent Transactions will fail until the balance is positive. |
| `CancelZeroBal-ance` | Reverses the Token grants made by the Transaction that is being refunded. If this action causes the token balances to drop below zero, CashBox sets the balance to zero. |
| `None` | Leaves the token grants made by the transaction that is being refunded as is, as if the Transaction had not been refunded. This value is the default. |

## 15.3 `Refund` Methods

The following table summarizes the methods for the `Refund` object.

Table 15-3 `Refund` Object Methods

| Method | Description |
|---|---|
| `fetchByAccount` | Returns one or more `Refund` objects that represent the refunds for the Transactions whose `Account` object matches the input. |
| `fetchByTransaction` | Returns one or more `Refund` objects that are associated with the `Transaction` object specified in the input. |
| `fetchByVid` | Returns a `Refund` object whose VID matches the input. |
| `fetchDeltaSince` | Returns one or more `Refund` objects whose timestamp falls on or after the timestamp specified in the input. |
| `perform` | Issues one or more refunds. |
| `report` | Reports the refunds to Vindicia for chargeback processing. |

## fetchByAccount

The `fetchByAccount` method returns one or more `Refund` objects that represent refunds made for Transactions whose `Account` object matches the input. Call this method for a list of all the refunds that have been issued to a certain customer.

### Input

***account:*** the `Account` object that serves as the search criterion. Use the `merchantAccountId` or `VID` to identify the object.

***includeChildren:*** an optional Boolean flag that, if set to `true`, includes any children associated with this `Account`. If `null` or `false`, CashBox will construct the query without including children accounts.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***refunds:*** an array of one or more `Refund` objects associated with the `Transaction` objects that are, in turn, associated with the `Account` object specified in the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load account to search by: No matches.`<br>• `No account specified to load refunds by!` |
| `404` | `Unable to load account to search by:` ***error-description.*** |

### Example

```
$account = new Account();
$account->setMerchantAccountId('jdoe101');

$refund = new Refund();
$response = $refund->fetchByAccount($account);
if($response['returnCode'] == 200) {
    $fetchedRefunds = $response['data']->refunds;

    // process fetched refunds here
    if ($fetchedRefunds != null) {
        foreach ($fetchedRefunds as $fetchedRef) {
            // process a fetched refund here
            print "Refund VID " . $fetchedRef->getVID();
            print "Refund amount ". $fetchedRef->getAmount();
            print "Refund timestamp ". $fetchedRef->getTimestamp();
        }
    }
}
```

## fetchByTransaction

The `fetchByTransaction` method returns one or more `Refund` objects associated with the `Transaction` object specified in the input.

With CashBox, you can issue multiple partial refunds against a `Transaction` as long as the amount of each refund is less than the `Transaction` amount, and the sum of all refunds does **not** exceed the `Transaction` amount.

If you are reporting refunds to Vindicia for chargeback processing only, multiple partial refunds may have been issued, and reported, against a single Transaction. Use this method to return all refunds listed against a specific Transaction.

### *Input*

***transaction:*** the `Transaction` object that serves as the search criterion. Identify this object with either its VID or your transaction ID (`merchantTransactionId`).

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***refunds:*** an array of one or more `Refund` objects associated with the `Transaction` object specified in the input.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | No transaction specified to load by! |
| 404 | Unable to load refund: No match for transaction. |

### *Example*

```
$txn = new Transaction();
$txn->setMerchantTransactionId('MRCH49229492');

$refund = new Refund();
$response = $refund->fetchByTransaction($txn);

if($response['returnCode'] == 200) {
    $fetchedRefunds = $response['data']->refunds;

    // process fetched refunds here
    if ($fetchedRefunds != null) {
        foreach ($fetchedRefunds as $fetchedRef) {
            // process a fetched refund here
            print "Refund VID " . $fetchedRef->getVID();
            print "Refund amount ". $fetchedRef->getAmount();
            print "Refund timestamp ". $fetchedRef->getTimestamp();
        }
    }
}
```

## fetchByVid

The `fetchByVid` method returns a `Refund` object whose VID matches the input.

The VID is assigned by CashBox when creating a new `Refund` object, in response to a refund issued with a `report()` or `perform()` call, or through the CashBox Portal. When constructing a `Refund` object to pass into a `report()` or `perform()` call, leave the `VID` field blank so that CashBox can assign the object a VID when it adds the object to the database. The VID is available in the `Refund` object returned to you.

### *Input*

***vid:*** the `Refund` object's Vindicia unique identifier, which serves as the search criterion.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***refund:*** the `Refund` object whose VID matches the input.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | `No VID specified to load refund by.` |
| `404` | One of the following:<br>• `Unable to load refund: No match for VID` ***input-vid.***<br>• `Unable to load refund by VID` ***input-vid: error-description.*** |

### *Example*

```
$vid='cdbdab93f509e2bf8c6d0e7918b0cee2e03cc175'

$refund = new Refund();

$response = $refund->fetchByVid($vid);

if($response['returnCode'] == 200) {
    $fetchedRef = $response['data']->refund;

    // process fetched refunds here
    if ($fetchedRef != null) {
        print "Refund VID " . $fetchedRef->getVID();
        print "Refund amount ". $fetchedRef->getAmount();
        print "Refund timestamp ". $fetchedRef->getTimestamp();
    }
}
```

## fetchDeltaSince

The `fetchDeltaSince` method returns one or more `Refund` objects whose timestamp falls on or after the timestamp specified in the input. Limit the number of objects returned by specifying an upper limit on the timestamp as well, using the ***endTimestamp*** parameter.

### Input

***timestamp:*** the search criterion for selecting `Refund` objects to be returned. The timestamps of those selected objects are less than or equal to this value.

***endTimestamp:*** the end-date and timestamp. `Refunds` with timestamps greater than this value will not be returned.

***paymentMethod:*** an optional constraint that, if included, restricts the return to only those `Refund` objects whose original Transactions were conducted with this payment method. Specify this parameter with either the `paymentMethod` VID or `merchantPaymentMethodId`.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***refunds:*** an array of one or more `Refund` objects whose timestamp falls on or after ***timestamp*** but before ***endTimestamp*** (if specified) in the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 404 | Unable to find payment method in database. |
| 400 | Must specify a timestamp to find refunds newer than … |

### Example

```
$refund = new Refund();
$pm = null;
$startTimeStamp = '2009-11-28T12:40:51-0800';
$endTimeStamp = '2009-12-28T12:40:50-0800';

$response = $refund->fetchDeltaSince($startTimestamp,$endTimestamp, $pm);

if($response['returnCode'] == 200) {
    $fetchedRefunds = $response['data']->refunds;

    // process fetched refunds here
    if ($fetchedRefunds != null) {
        foreach ($fetchedRefunds as $fetchedRef) {
            // process a fetched refund here
            print "Refund VID " . $fetchedRef->getVID();
            print "Refund amount ". $fetchedRef->getAmount();
            print "Refund timestamp ". $fetchedRef->getTimestamp();
        }
    }
}
```

# perform

The `perform` method enables you to issue one or more refunds for Transactions that were processed through CashBox. Not all CashBox Transactions are refundable. If CashBox cannot process some of the refunds in your input, you are informed through the return code in this call's `Return` object.

CashBox can refund a transaction only if it meets all of the following criteria:

- The transaction status is one of the following:
  - `Captured`.
  - `Refunded` (if a partial refund has occurred).
  - `Authorized`. The Transaction is scheduled for capture but is not yet captured with your payment processor. Refunding such a Transaction essentially cancels it.
  - `AuthorizedPending` or `DepositRetryPending` for ECP or Direct Debit-based transactions. Refunding such a Transaction essentially cancels it.
- The Transaction has an associated authorization response code.
- The Transaction was not paid through the Boleto Bancário payment method.
- The Transaction is not an outbound Transaction, conducted to pay a customer through the ECP-based payment method.
- The sum of all the Transaction's past refunds is less than the original Transaction amount.

Also note that you cannot grant partial refunds if:

- The Transaction used a `Token` payment method that resulted in the granting of tokens to a customer's Account.
- Your payment processor is GlobalCollect and the authorization code is `800`, which means that the Transaction has been captured but not yet settled.

CashBox processes refunds submitted through this call asynchronously with your payment processor in batches. Because CashBox ensures that the refunds submitted are indeed refundable when you call `perform()`, payment processors rarely reject refunds accepted by CashBox. To monitor refund status, log into the CashBox Portal and use the **Transaction Details** page, which displays the most up-to-date status of your refund-.

## *Input*

***refunds:*** an array of one or more `Refund` objects, each corresponding to a refund that you would like to process through CashBox. Because this call creates a `Refund` object in CashBox, leave the VID field blank. If CashBox accepts the refund for processing, it populates the VID field in the corresponding `Refund` object in the array of returned refunds.

## *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

If the return code is `200`, all the `Refund` objects in this array have Vindicia-assigned VIDs, indicating that CashBox has accepted these objects for processing. A return code of `206` indicates that only some of the `Refund` objects have been accepted by CashBox and have VIDs. The `Refund` objects without VIDs have been rejected by CashBox because they do not meet the criteria described above. Reasons for rejection are included in the `note` attribute of the `Refund` objects.

***refunds:*** an array of one or more `Refund` objects, which corresponds to your input array.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 206 | Some (or all) refunds failed; check VIDs, notes. |
| 404 | Cannot refund transaction: ***error-description***. |

## Example

```
$refundVid = 'MyVindiciaRefundVID';

// Create a refund object
$refund1 = new Refund();
$refund1->setMerchantRefundId('REF101');

$transaction1 = new Transaction();
// merchant ID of a successful transaction that we wish to refund
$transaction1->setMerchantTransactionId('TX101');

$refund1->setTransaction($transaction1);
$refund1->setAmount(5.99);
$refund1->setNote('Refunding due to customer complaint about outage');

// Create another refund object
$refund2 = new Refund();
$refund2->setMerchantRefundId('REF102');

$transaction2 = new Transaction();
// merchant ID of a successful transaction that we wish to refund
$transaction1->setMerchantTransactionId('TX102');

$refund2->setTransaction($transaction2);
$refund2->setAmount(10.99);
$refund2->setNote('Customer charged twice');
$soap_refund = new Refund();
$response = $soap_refund->perform(array($refund1, $refund2));
if($response['returnCode'] == 200) {
   print ("All refunds submitted successfully");
}
else if($response['returnCode'] == 206) {
   $resultRefunds = $response['data']->refunds;

   // process fetched refunds here
   if ($resultRefunds != null) {
      foreach ($resultRefunds as $resultRef) {
         // process a fetched refund here
         if($resultRef->getVID() != null) {
              print "Refund id "
                    . $resultRef->getMerchantRefundId()
                    . " submitted successfully";
         }
         else {
              print "Refund id "
                    . $resultRef->getMerchantRefundId()
                    . " was unsuccessful because "
                    . $resultRef->getNote();
         }
      }
   }
}
```

## report

Call the `report` method to report refunds that were issued outside of CashBox. Use this method to report ChargeGuard information to Vindicia for chargeback disputes. Unlike the `perform()` call, `report()` does not process refunds with your payment processor, but simply stores the `Refund` objects reported in the Vindicia database.

If the `Refund` object passed in this call refers to a Transaction that does not exist in the CashBox database, this call creates and stores the Transaction there. CashBox expects that, as a ChargeGuard customer, if you are reporting a refund on a transaction, you have previously reported that transaction to Vindicia. If you have not done so, however, this call creates a `Transaction` object in CashBox according to the information you include in the call.

| **Note** | For ChargeGuard customers: If a chargeback against a transaction exists, be sure to report the refund you issued for it. Doing so automatically means that you have won the chargeback. |
|---|---|

### *Input*

***refunds:*** an array of one or more `Refund` objects to report. Leave the `VID` attribute blank because CashBox will assign VIDs when creating the corresponding database records, and will return them to you with the `Refund` objects in the output.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***refunds:*** an array of one or more `Refund` objects. This array corresponds to your input array. If the return code is `200`, all `Refund` objects in this array have CashBox-assigned VIDs, because CashBox has created records in its database for each of those objects.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | `Unable to save refunds:` ***error-description.*** |

### *Example*

```
// to report a refund issued outside of CashBox

$refundVid = 'MyVindiciaRefundVID';

// Create a refund object
$refund1 = new Refund();
$refund1->setMerchantRefundId('REF101');

$transaction1 = new Transaction();
// merchant ID of a previously reported transaction
$transaction1->setMerchantTransactionId('TX101');

$refund1->setTransaction($transaction1);
$refund1->setAmount(5.99);
$refund1->setNote('Refunded due to service outage');
// Payment Processor's refund id when you processed
// this refund with it directly - if available
$refund1->setReferenceString('2033992');

// Create another refund object
$refund2 = new Refund();
$refund2->setMerchantRefundId('REF102');

$transaction2 = new Transaction();
// merchant ID of a previously reported transaction
$transaction1->setMerchantTransactionId('TX102');

$refund2->setTransaction($transaction2);
$refund2->setAmount(10.99);
$refund2->setNote('Customer did not receive delivery);

$soap_refund = new Refund();
$response = $soap_refund->report(array($refund1, $refund2));
if($response['returnCode'] == 200) {
   print ("All refunds submitted successfully");
}
```

# 16 The `SeasonSet` Object

A `SeasonSet` object allows you to create groups of time intervals, which may be used with Billing Plans to define both Billing Cycles, and Entitlement grants.

Season Sets are best described using the CashBox user interface, rather than the API.

# 16.1 SeasonSet Data Members

The SeasonSet object defines an array of seasons, with an identifier.

The following table lists and describes the data members of the SeasonSet object.

Table 16-1 Token Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| merchantSeasonSetID | string | Your unique ID for this SeasonSet. Free-form string 255 characters or fewer. |
| nameValues | NameValuePair | An array of name-value pairs to associate with the SeasonSet. |
| seasons | Season | An array of Seasons that make up the SeasonSet.<br>The Season object contains three values:<br>description: your description for the Season.<br>startDate: its start date.<br>endDate: its end date. |
| VID | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new SeasonSet object, leave this field blank; it will be automatically populated by CashBox. |

## 16.2 SeasonSet Methods

The following table lists and summarizes the methods for the SeasonSet object.

Table 16-2 SeasonSet Object Methods

| Method | Description |
|---|---|
| fetchAll | Returns all SeasonSets. |
| fetchAllInSeason | Returns all in season SeasonSets. |
| fetchAllOffSeason | Returns all off-season SeasonSets. |
| fetchByMerchantSeasonSetId | Returns the SeasonSet specified by the input Merchant ID. |
| fetchByVid | Returns the SeasonSet specified by the input VID. |
| fetchCurrentSeason | Returns the current Season for the input SeasonSet. |
| fetchNextSeason | Returns the next Season for the input SeasonSet. |
| isInSeason | Returns a Boolean flag, which indicates whether the input SeasonSet is in season. |
| update | Creates a new SeasonSet or updates an existing one. |

## fetchAll

The `fetchAll` method returns all existing `SeasonSets`.

### Input

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize***: the number of records to display per page per call. This value must be greater than 0.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***seasonSets:*** an array of returned `SeasonSet` objects.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$ss_factory = new SeasonSet();
$page = 0;
$pageSize = 10;
do {
    $ret = $ss_factory ->fetchAll($page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedSets = $ret['seasonSets'];
        $count = sizeof($fetchedSets);
        foreach ($fetchedSets as $set) {

            // process a fetched Season Set here …
        }
    $page++;
    }
} while ($count > 0);
```

## fetchAllInSeason

The `fetchAllInSeason` method returns all existing `SeasonSet` objects that are in season during the input ***nowDate***.

### *Input*

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize***: the number of records to display per page per call. This value must be greater than 0.

***nowDate:*** the (optional) date to query. (Defaults to ***today***.)

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***seasonSets:*** an array of returned `SeasonSet` objects.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$ss_factory = new SeasonSet();
$page = 0;
$pageSize = 10;
do {
    $ret = $ss_factory ->fetchAllInSeason($page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedSets = $ret['seasonSets'];
        $count = sizeof($fetchedSets);
        foreach ($fetchedSets as $set) {

            // process a fetched Season Set here …
        }
    $page++;
    }
} while ($count > 0);
```

## fetchAllOffSeason

The `fetchAllOffSeason` method returns all existing `SeasonSet` objects that are off-season during the input ***nowDate***.

### *Input*

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize***: the number of records to display per page per call. This value must be greater than 0.

***nowDate:*** the (optional) date to query. (Defaults to ***today***.)

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***seasonSets:*** an array of returned `SeasonSet` objects.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$ss_factory = new SeasonSet();
$page = 0;
$pageSize = 10;
do {
    $ret = $ss_factory ->fetchAllOffSeason($page, $pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedSets = $ret['seasonSets'];
        $count = sizeof($fetchedSets);
        foreach ($fetchedSets as $set) {

            // process a fetched Season Set here …
        }
    $page++;
    }
} while ($count > 0);
```

## fetchByMerchantSeasonSetId

The `fetchByMerchantSeasonSetId` method returns an existing `Season` object that matches the input `merchantSeasonSetId`.

***Input*** ***merchantSeasonSetId:*** the input SeasonSet ID.

***Output*** ***return:*** an object of type `Return` that indicates the success or failure of the call.

***seasonSet:*** the returned `SeasonSet` object.

***Returns*** This method returns the codes listed in Table 1: Standard Return Codes.

***Example***

```
$ss_factory = new SeasonSet();
$response - $ss_factory->fetchByMerchantSeasonSetId ('Summer Volleyball');

   // check $response

$volleyball seasonSet = $response['Season Set'];
```

## fetchByVid

The `fetchByVid` method returns an existing `SeasonSet` object that matches the input VID.

### Input

***vid:*** the Vindicia ID to query.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***seasonSet:*** the returned `SeasonSet` object.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$ss = new SeasonSet();
$response =
       $ss->fetchByVid('8367ae7148d071a4e25c24bef856f68f71ee03e3');

// check $response

$seasonSet = $response['seasonSet'];
print "got SeasonSet " . $seasonSet->name() . "\n";
```

## fetchCurrentSeason

The `fetchCurrentSeason` method returns the current `Season` for the input `SeasonSet`.

### *Input*

***seasonSet:*** the `SeasonSet` object to query.

***nowDate:*** the (optional) date to query. (Defaults to ***today***.)

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***season:*** the current `Season`, or `null` if not currently in season.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$ss = new SeasonSet();
$response = $ss->fetchCurrentSeason();

   // check $response

$season = $response['season'];
```

## fetchNextSeason

The `fetchNextSeason` method returns the next `Season` for the input `SeasonSet`.

### *Input*

***seasonSet:*** the `SeasonSet` object to query.

***nowDate:*** the (optional) date to query. (Defaults to ***today***.)

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***season:*** the next `Season`, or `null` if none exist.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$ss = new SeasonSet();
$response = $ss->fetchNextSeason();

   // check $response

$season = $response['season'];
```

## isInSeason

The `isInSeason` method returns a Boolean flag which indicates whether the input `SeasonSet` is in season.

> **Note:** This method will return all Season Sets which include a Season which is currently active. This method will not return any Season Sets which are currently off-season, even if that set includes a Season which will be active in the future.

### *Input*

***seasonSet:*** the `SeasonSet` object to query.

***nowDate:*** the (optional) date to query. (Defaults to ***today***.)

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***inSeason:*** `true` if the `SeasonSet` is in season; `false` if it is not.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
# Given $my_ss, which we want to ask about:
$ss_factory = new SeasonSet();
$response = $ss_factory->isInSeason($my_ss);

// check $response

if($response['inSeason']) {
   print "My Season Set has a Season that is in effect now.";
}

else {
   print "My Season Set has no Season that is in effect now.";
}
```

## update

The `update` method creates a new, or updates an existing `SeasonSet` object.

To create a `SeasonSet` object, initialize the object, set the values for its data members, and then call the `update` method to store the changes in the Vindicia database. Do ***not*** set a value for `VID`; CashBox automatically generates a VID when you call `update()`. When updating an existing `SeasonSet` object, identify it with either its VID or your SeasonSet ID (`merchantSeasonSetId`).

### *Input*

***seasonSet:*** the `SeasonSet` object to create or update. To update an existing `SeasonSet` object, identify it with either its VID or your SeasonSet ID (`merchantSeasonSetId`). If you specify a new value for `merchantSeasonSetId`, CashBox will create a new `SeasonSet`.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***seasonSet:*** the `SeasonSet` object that was created or updated.

***created:*** returns `true` if a new object was created; `false` if an existing object was updated.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

### *Example*

```
$summer_ss = new SeasonSet();
$summer_ss->setMerchantSeasonSetId('Summers');

$s2013 = new Season();
$s2013->setStartDate('2013-06-22');
$s2013->setEndDate('2013-09-22');

$s2014 = new Season();
$s2014->setStartDate('2014-06-21');
$s2014->setEndDate('2014-09-20');

$summer_ss->setSeasons(array($s2013, $s2014));

$ss_factory = new SeasonSet();
$response = $ss_factory->update($summer_ss);
// check $response
```

# 17 The Token Object

A Token object represents a metering or virtual-currency unit of a certain type, which is identified by the object's unique ID (merchantTokenId).

Token objects enable you to define a credit system in your application without conducting actual monetary transactions. For example, a cell-phone company can use a Token object to represent a one-minute phone call; an online game company can have a Token object represent a player's game time, and another Token object represent virtual goods. An airline might use a Token object to represent 1000 frequent-flier miles earned by a customer.

Token objects are meaningful when associated with Account objects. A certain number of Token objects of a certain type associated with an Account object define the customer's credit recognized by your application and allow the customer access to resources within the application.

With a TokenAmount object (see the Account object), you can couple a token type with a quantity, and then associate various token amounts with an Account. For example:

- While creating an Account object, populate its tokenBalances attribute with TokenAmount objects to grant Tokens of various types to the customer. The Account object supports incrementTokens() and decrementTokens() calls, which allow you to manipulate the quantities of token types. To grant or revoke tokens owned by an Account object, you may also conduct a token-based Transaction with the object's tokenTransaction() call. For more information, see Section 1: The Account Object.
- You may also define one or more TokenAmount objects on a Product object. When a customer acquires a product through an AutoBill instance, CashBox adds the Token amounts defined on the Product to the customer's Account. For more information, see Section 13: The Product Object.

Because Token objects are meaningful only when attached to Account objects, most of the token-related methods are defined on the Account object. The Token object itself offers methods only for creating new token types and for fetching tokens.

## 17.1 `Token` Data Members

The following table lists and describes the data members of the `Token` object.

Table 17-1 `Token` Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| `description` | string | **Optional.** A description of this token type in your application. |
| `merchantTokenId` | string | **Required.** Your unique identifier for this `Token` object. This ID is also referred to as the *token type*. For example, an airline might identify a `Token` object with the ID `FREQ_FLIER_MILES_2010` to denote the number of frequent-flier miles accumulated by a customer account in 2010.<br>A cell-phone company might use `ANYTIME_PHONE_MINUTES` to identify `Token` objects that specify a customer's balance of anytime minutes. |
| `VID` | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `Token` object, leave this field blank; it will be automatically populated by CashBox. |

## 17.2 `Token` Methods

The following table lists and summarizes the methods for the `Token` object.

Table 17-2 `Token` Object Methods

| Method | Description |
|---|---|
| `fetch` | Returns an existing `Token` object. |
| `update` | Creates or updates a `Token` object. |

## fetch

The `fetch` method returns an existing `Token` object that matches your token ID (`merchantTokenId`) or the VID for the object as specified in the input.

### Input

***token:*** the `Token` object that serves as the search criterion. Identify this object with either its VID or your token ID (`merchantTokenId`).

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***token:*** the returned `Token` object.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | No token specified to load! |

### Example

```
$soapCaller = new Token();

$tok = new Token();
$tok->setMerchantTokenId("ANY_TIME_PHONE_MINUTES");
$response = $tok->fetch();

if($response['returnCode'] == 200) {
    $fetchedToken = $response['data']->token;
    print "Fetched token with id: " .
           $fetchedToken->merchantTokenId . " and VID: ";
    print $fetchedToken->VID . " and description: " .
           $fetchedToken->description;
    print "\n";
}
```

## update

The `update` method creates or updates a `Token` object.

To create a `Token` object, initialize the object, set the values for its data members, and then call the `update` method to store the changes in the Vindicia database. Do **not** set a value for `VID`; CashBox automatically generates a VID when you call `update`. When updating an existing `Token` object, identify it with either its VID or your token ID (`merchantTokenId`).

### *Input*

***token:*** the `Token` object to create or update. To update an existing `Token` object, identify it with either its VID or your token ID (`merchantTokenId`). If you specify a new value for `merchantTokenId`, CashBox will create a new Token type.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***token:*** the updated or created `Token` object.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | `Unable to save token:` ***error-description.*** |
| `501` | ***Error-description.***<br>(Returned if the call cannot map the SOAP `Token` object to CashBox's database representation of the token.) |

### *Example*

```
$soapCaller = new Token();

$tok = new Token();
$tok->setMerchantTokenId("ANY_TIME_PHONE_MINUTES");
$tok->setDescription("Any time phone minutes for 2010");

// Make the SOAP call to create the token
$response = $tok->update();
if($response['returnCode'] == 200) {
   print "Created token type with id " .
          $tok->merchantTokenId . " and vid ";
   print $response['token']->VID . "\n";
}
```

# 18 The `Transaction` Object

The `Transaction` object encapsulates information about a financial transaction processed through CashBox. In addition to standard transaction content, such as customer information (`Account`), payment information (`PaymentMethod`), line items (`TransactionItem`), and amount, this object contains a rich set of attributes that support CashBox services.

A `Transaction` object might represent a financial transaction conducted through CashBox for recurring or one-time billing, or one conducted outside of CashBox, but reported to Vindicia for chargeback dispute through ChargeGuard. Please note that `Refund` objects, rather than `Transaction` objects encapsulate information on refunds to your customers.

CashBox processes `Transaction` objects with your payment processor and updates their status during the process. The `Transaction` object includes an array of `TransactionStatus` subobjects that form a log of statuses through the Transaction processing sequence.

When migrating a Transaction to CashBox, be certain to include the latest or final status information within the `Transaction` object (such as the reason code returned by your payment processor). The status cycle of a `Transaction` object and the reason codes will vary, depending on the payment method and your processor.

A `Transaction` object might also represent a potential, rather than a completed, financial transaction. For example, you may score a Transaction and screen it for fraud risk before moving funds through your payment processor. (For more information on risk screening, see Chapter 14: Common ChargeGuard Programming Tasks in the ***CashBox Programming Guide***.) If the scoring result reflects a high fraud probability, you might decide to abandon the Transaction, in which case the corresponding `Transaction` object remains in CashBox in the `New` status, which means that it was never processed.

When Vindicia downloads chargebacks from your payment processor for ChargeGuard, it matches them to your transactions in its database. If you have conducted one of those transactions outside of Vindicia but have not yet migrated it, CashBox creates a stub `Transaction` object in its database with the `Transaction` information in the chargeback that was downloaded. After you've reported that transaction to CashBox, the object is populated with the remaining information.

The following table lists and describes cases in which a Transaction object should be used.

| | |
|---|---|
| **Note:** | The "Transaction Types" listed in this table are not formal types, but simply general classifications. |

Table 18-1 Uses for the `Transaction` Object

| Transaction Type | Description | Initiated By | Transaction API Call |
|---|---|---|---|
| Migrated | Transactions generated by a billing system other than CashBox, that have been migrated into CashBox by either the `AutoBill.migrate` or `Transaction.migrate` calls. Once imported, these Transactions will behave as if they were Recurring/Real-time Transactions generated within CashBox. | You, the merchant | `AutoBill.migrate`<br>`Transaction.migrate` |
| Real-time (one-time) | A one-time purchase by a customer. CashBox authorizes this transaction with your payment processor in real time in response to your call. Depending on the call, the transaction may be captured with the payment processor later in batch mode. A captured transaction means that monies will be exchanged. This type of transaction goes through status changes until it is eventually captured. | You, the merchant | `Transaction.auth()`<br>`Transaction.capture()`<br>`Transaction.authCapture()` |
| Recurring billing | Periodic transactions generated by CashBox for an instance of an `AutoBill` object. These are recurring transactions for a customer's subscription, with the frequency, amount, and other content determined by the `Product` and `BillingPlan` objects in the `AutoBill` object. To create an `AutoBill` instance for a customer's subscription, either make an API call, or create the instance on the CashBox Portal. CashBox captures these transactions in batch mode. The status of the entitlements offered by the corresponding `AutoBill` object depends on whether the transaction is captured successfully. | CashBox | None. An active `AutoBill` object must exist. |
| Reported | A transaction conducted outside of CashBox, and reported to Vindicia for chargeback dispute. CashBox does not process this transaction with a payment processor, nor does the transaction go through changes in status. | You, the merchant | `Transaction.report()`<br>`Transaction.score()` |
| Stub | A transaction with minimal data. If you are a ChargeGuard customer, Vindicia downloads your chargebacks from your payment processor and matches them to their corresponding Transactions in the Vindicia database to capture all information available for chargeback disputes.<br>If the original transaction is not in the database (was conducted outside of CashBox but not yet migrated), CashBox creates a stub `Transaction` object that contains the minimal data obtained from the chargeback, and stores the object in the database.<br>Once you have migrated the transaction, CashBox will enter the missing details. | Indirectly by you, the merchant | None. |

Table 18-1 Uses for the `Transaction` Object *(Continued)*

| Transaction Type | Description | Initiated By | Transaction API Call |
|---|---|---|---|
| Validation | A transaction to validate a payment method. When you make an API call or perform a task on the CashBox Portal to validate a payment method, CashBox generates a Transaction that uses that payment method for an amount of one currency unit (US$1 if the payment method specifies USD as the currency), and authorizes it with your payment processor.<br>If the Transaction is authorized, the payment method is considered valid, and the transaction status becomes `AuthorizedForValidation`; if not, the status is `Cancelled`.<br>Not all payment methods may be validated this way. (See Section 11: The PaymentMethod Object for details.) Because CashBox only *authorizes* such a transaction with your processor but never *captures* it, the customer is not charged for the transaction. | CashBox | None. |

When creating and processing a `Transaction` object through CashBox, reporting it to Vindicia for ChargeGuard, or scoring it for risk screening, be sure to include all related information. The more detail you provide, the more effective Vindicia will be in disputing chargebacks on your behalf should they occur.

# 18.1 `Transaction` Data Members

The following table lists and describes the data members of the `Transaction` object.

Table 18-2 `Transaction` Object Data Members

| Data Member | Data Type | Description |
|---|---|---|
| `account` | `Account` | The `Account` object that represents the customer to which this `Transaction` object applies.<br>See Section 1.2: Account Data Members. |
| `amount` | decimal | **Required**. The monetary amount for this `Transaction` object, that is, the total cost of one or more line items purchased. When you process the transaction through CashBox by calling `auth` or `authCapture`, CashBox fills in this attribute based on the total of the line items (see the `TransactionItem` attributes) added to the transaction. When reporting the transaction to Vindicia, ensure that the transaction amount matches the total of the line items.<br>For CashBox-generated Transactions, this field is automatically generated. |
| `autoBillCycle` | int | The AutoBill Billing Cycle during which this Transaction occurred. |
| `billingPlanCycle` | `int` | The zero-based number of times the AutoBill has been billed for the current Billing Plan. One-time transactions will have the same value as the most recent recurring billing event, as determined by the AutoBill and its BillingPlan.<br>**Note:** This data member will increment for free cycles. |
| `billingStatementIdentifier` | string | The string that is displayed on a customer's billing statement. For one-time transactions, CashBox supports this value for only certain payment processors. Because this value and its format are constrained by your payment processor, consult with Vindicia Client Services before setting its value.<br>If GlobalCollect, MeS, Chase Paymentech or Litle is your payment processor, see Appendix A: Custom Billing Statement Identifier Requirements in the ***CashBox Programming Guide***. |
| `currency` | string | The ISO 4217 currency code (see www.xe.com/iso4217.htm) for this `Transaction` object. The default is USD. To determine the actual monetary value, set the values for both `amount` and `currency`. |
| `destPaymentMethod` | `PaymentMethod` | The payment method for deposits to a customer account for this `Transaction` object. This field is used to make outbound ECP payments or transfers.<br>See Section 11.1: PaymentMethod Data Members. |
| `divisionNumber` | string | The number of your division or group with your payment processor for this Transaction. Chase Paymentech refers to this number as the Division Number; Litle calls it the Report Group; MeS calls it the Profile ID. Do not specify this attribute for one-time transactions.<br>If you subscribe to ChargeGuard, complete this field when reporting transactions to CashBox. CashBox will use this value to match the Transaction to the appropriate chargeback from the payment processor. |

Table 18-2 `Transaction` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `ecpTransaction-Type` | `ECPTransaction-Type` | The ECP transaction mode for the `Transaction` object, for example, `Inbound` or `Outbound`. If this value is `Outbound` or `Transfer`, you must set a value for `destPaymentMethod`. Specify this attribute for ECP-based transactions only. |
| `merchantAffili-ateId` | string | **Optional.** Your unique identifier for the partner or affiliate who directed this `Transaction` object to you. Track this information if, for example, you pay a service fee to affiliates who generate business and revenue for you. To implement affiliate tracking, fill in this attribute when reporting or processing one-time Transactions through CashBox. For recurring transactions, CashBox fills in this attribute if it is specified in the corresponding `AutoBill` object. |
| `merchantAffili-ateSubId` | string | **Optional.** Your sub-ID for (and additional information on) the partner or affiliate who directed this Transaction to you. To implement affiliate tracking, fill in this attribute when reporting or processing one-time transactions through CashBox. For recurring `Transactions`, CashBox fills in this attribute if it is specified in the corresponding `AutoBill` object. |
| `merchantTrans-actionId` | string | Your unique identifier for this `Transaction` object. CashBox automatically generates this value for rebilling transactions with the prefix you specified during initial configuration. Vindicia recommends that this prefix differ from the one specified for *recurring* transactions.<br>For real-time transactions that you authorize or capture by making a call to CashBox, for example, with `Transaction.capture()`, you must fill in this attribute.<br>If you are reporting this transaction to Vindicia for ChargeGuard only, ensure that this ID matches the order number you sent to the payment processor. That way, ChargeGuard can match this transaction with a chargeback received for this transaction from the processor. |

Table 18-2 `Transaction` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `nameValues` | `NameValuePair[]` | **Optional.** An array of name–value pairs, which are useful in tracking the associated `AutoBill` object.<br>CashBox provides four name-value pairs for use with European Direct Debit (EDD) payment methods:<br>Use name `vin:MandateFlag` and value `1` to associate the EDD Payment Method with the AutoBill.<br>Use name `vin:MandateVersion` and value `1.0.1`, to associate a mandate document of version 1.0.1 with the object.<br>Use name `vin:MandateID` to pass the Mandate ID field of the EDD Extension record to Chase Paymentech.<br>Use name `vin:MandateApprovalDate` to pass the Signature Date field of the EDD Extension Record to Chase Paymentech.<br>**Note:** All name-value pairs included with the Transaction-generating AutoBill will be automatically copied to the resultant `Transaction`.<br>The following name-value pairs are automatically populated by Cash-Box for AutoBill-generated Transactions:<br>• `vin:AutoBillVID:` the VID of the AutoBill for which this Transaction was generated.<br>• `vin:ignoreCredits`: if set to `true`, specifies that `Transaction.auth`, `Transaction.capture`, and `Transaction.authCapture` calls ignore Credits available on an Account to pay for a one-time purchase. If set to `false`, these calls will use available credits for the Transaction. (This name-value pair enables customers to make purchases, without using available Credits to pay for them.)<br>• `vin:MerchantAutoBillIdentifier`**:** your unique ID for the AutoBill for which this Transaction was generated.<br>• `vin:RetryNumber`: the attempt number (in retry cycles) of the Transaction.<br>• `vin:Type:` the type of Transaction. CashBox will automatically populate this name-value pair with value = `modify` for Transactions which are the result of a `Transaction.modify` call.<br>See Section 10: The NameValuePair Object. |
| `note` | string | An optional description of the `Transaction` object. |
| `originalAmount` | decimal | In the event of a partial payment, this read-only field reflects the original amount of the Transaction, as a decimal value. |
| `paymentProcessor` | string | The payment processor for this `Transaction` object. This string will be available to you in the `Transaction` object CashBox returns to you in response to your call.<br>**Note:** If CashBox handles the billing, do not fill in this field. |
| `preferredNotificationLanguage` | string | The language (specified as an ISO language string) CashBox uses in email notifications when creating a real-time (one-time) `Transaction` (see the `authCapture` method), assuming that a template for this language and the email notification type have been uploaded to the Cash-Box database as part of your configuration. This value overrides any language setting in the `Account` object for this transaction. |

Table 18-2 `Transaction` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `previousMerchantTransactionId` | string | Your unique identifier for a previous transaction referenced by this `Transaction` object. |
| `salesTaxAddress` | `Address` | The corrected billing or shipping address CashBox uses to calculate sales tax for this `Transaction` object. CashBox fills it in automatically. This field is optional for migrated transactions.<br>**Note:** If CashBox calculates sales tax for you, leave this field empty.<br>See Section 3.1: Address Data Members. |
| `shippingAddress` | `Address` | **Optional.** The customer's shipping address for this `Transaction` object. For one-time transactions, CashBox uses this address first to calculate taxes, if any, that are to be added to this transaction's total.<br>See Section 3.1: Address Data Members. |
| `sourceIp` | string | **Optional.** The IP address from which this `Transaction` object originated. This attribute is required for reporting transactions for ChargeGuard, and for scoring transactions for risk screening. With this information, CashBox can pinpoint the geographical location at which the transaction was made. For CashBox-generated recurring transactions, this is the IP address specified on the corresponding `AutoBill` object. |
| `sourceMacAddress` | string | **Optional.** The Media Access Control (MAC) address of the customer computer or router, from which this `Transaction` object originated. This information can be useful in chargeback disputes. |
| `sourcePaymentMethod` | `PaymentMethod` | The payment method through which this `Transaction` object will deduct funds. CashBox uses this payment method for actual billing. For one-time transactions, except for outbound ECP-based, specify this attribute. If the payment method is not already attached to the account for this `Transaction` object, CashBox attaches it when saving this object in the database. To turn off this behavior, set the `PaymentMethod` object's `active` attribute to `false`.<br>For one-time transactions, if `shippingAddress` is not specified on the `Transaction` object, CashBox uses the billing address specified on the payment method for calculating taxes, if any.<br>For recurring transactions generated by CashBox, this attribute is the `PaymentMethod` object associated with the corresponding `AutoBill` object.<br>When reporting a transaction for ChargeGuard, partially mask the payment method's data members, for added security.<br>See Section 11.1: PaymentMethod Data Members. |
| `sourcePhoneNumber` | string | **Optional.** The phone number from which this `Transaction` object originated. This information may be useful in chargeback disputes. |

Table 18-2 `Transaction` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `statusLog` | `TransactionStatus()` | An array of the statuses this `Transaction` object has gone through, with the first entry being the most recent status. Each `TransactionStatus` object contains a CashBox enumerated status type (for example, `Authorized` or `Captured`) and the responses from the payment processor, depending on how CashBox processed the `Transaction`. You need not specify this attribute when creating `Transaction` objects for risk screening.<br>Because CashBox sets this value for real-time (one-time) transactions, leave this field empty when creating a `Transaction` object to be processed through CashBox. When your API call completes, CashBox returns to you the `Transaction` object with this attribute filled in. Be sure to examine this attribute in the returned object to verify that the transaction has been approved by the payment processor.<br>For recurring transactions, CashBox sets this attribute when capturing the transaction with the payment processor.<br>When reporting transactions for ChargeGuard, specify a value for this field. After status updates, report the transaction again and include the reason codes (`auth` codes or other return codes) received from the payment processor.<br>See the `TransactionStatus` Subobject. |
| `taxExemptions` | `TaxExemption` | An array of tax exemptions to be applied by CashBox to this `Transaction` object. Specify this attribute for one-time transactions for which CashBox calculates and adds applicable taxes, if any, and adjusts the total transaction amounts accordingly.<br>See the `TaxExemption` Subobject. |
| `timestamp` | `dateTime` | A timestamp that specifies the date and time of when this transaction occurred. CashBox sets this value for one-time and recurring transactions. Be sure to include this attribute in migrated transactions; otherwise, it defaults to the current time. |
| `transactionItems` | `TransactionItem` | A `TransactionItem` array that lists the line items that comprise this `Transaction` object. Each item is a separate data structure of type `TransactionItem`.<br>For migrated transactions, CashBox does not validate that the subitem amounts listed here add up to the total transaction amount (see the `amount` attribute).<br>For one-time transactions, CashBox adds the subitem amounts and sets this `Transaction` object's amount attribute. CashBox also adds applicable taxes, such as city tax and state tax, as subitems.<br>For CashBox-generated recurring transactions, this attribute consists of a `TransactionItem` that refers to the `Product` object on the corresponding `AutoBill` object and the applicable tax items.<br>To add sales tax when migrating transactions, include the tax as a line item here.<br>See the `TransactionItem` Subobject. |
| `userAgent` | string | **Optional.** Your customer's user agent from whom this Transaction originated. |

Table 18-2 `Transaction` Object Data Members *(Continued)*

| Data Member | Data Type | Description |
|---|---|---|
| `verification-Code` | string | The response from a payment verification system, for example, Visa (VbV) or MasterCard SecureCode, for this `Transaction` object. If you report transactions to Vindicia for ChargeGuard, populate this field with the information on the payment verification performed while conducting this transaction. |
| `VID` | string | Vindicia's unique identifier for this `Transaction` object. When creating a `Transaction` object, leave this field empty. Vindicia assigns it a VID when saving the object in the database and make the VID available in the `Transaction` object returned to you in response to your call. Afterwards, you can refer to the object by specifying either the VID or `merchantTransactionId`.<br>**Note:** In the absence of an existing VID or `merchantTransactionId`, Vindicia treats a `Transaction` object as a new object for any API call, and assigns the object a new VID. |

## 18.2 `Transaction` Subobjects

The `Transaction` object has several subobjects:

- `AVSMatchType` Subobject
- `MigrationTaxItem` Subobject
- `MigrationTransaction` Subobject
- `MigrationTransactionItem` Subobject
- `MigrationTransactionType` Subobject
- `TransactionItem` Subobject
- `TransactionStatus` Subobject
- `TransactionStatusBoleto` Subobject
- `TransactionStatusCreditCard` Subobject
- `TransactionStatusECP` Subobject
- `TransactionStatusHostedPage` Subobject
- `TransactionStatusPayPal` Subobject
- `TransactionStatusType` Subobject
- `TransactionValidationResponse` Subobject

### `AVSMatchType` Subobject

Defines the AVS Match type.

Table 18-3 `AVSMatchType` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `FullMatch` | string | The billing address from the customer matches the one on file with the bank. |
| `IssuerError` | string | The payment processor or card issuer has returned an error. For credit-card-based transactions, you may retrieve the payment processor's response code from the `creditCardStatus` attribute. |
| `NoMatch` | string | The billing address from the customer does not match the one on file with the bank. |
| `NoOpinion` | string | CashBox cannot classify a new AVS return code from the payment processor, and will update its database to classify this code for future transactions. For credit-card-based transactions, you may retrieve the payment processor's response code from the `creditCardStatus` attribute. |

Table 18-3 `AVSMatchType` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `NotSupported` | string | The AVS match type requested is not supported. |
| `PartialMatch` | string | The billing address from the customer partially matches the one on file with the bank. For credit-card-based transactions, you may retrieve the payment processor's actual response code from the `creditCardStatus` attribute. |

## `MigrationTaxItem` Subobject

Defines a tax line-item in a `MigrationTransactionItem`.

Table 18-4 `MigrationTaxItem` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `amount` | decimal | Tax amount in the currency of the overall transaction. |
| `jurisdiction` | string | Sales tax jurisdiction for the Transaction. |
| `name` | string | Sales tax name. |

## `MigrationTransaction` Subobject

Defines a `Transaction` migrated to CashBox from a different billing system.

Table 18-5 `MigrationTransaction` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `account` | `Account` | The `Account` associated with this `Transaction`. If this `migrationTransaction` is included in an `AutoBill` migration request, the Account on the `AutoBill` will be used instead of this field.<br>When calling `AutoBill.migrate`, the `Account` on the `AutoBill` passed in will be associated with all of the `Transactions` created. When calling `Transaction.migrate`, the `Account` on the `MigrationTransaction` object will be used. In both cases, you may create `Accounts` on the fly by passing in an `Account` that does not yet exist in CashBox. |
| `amount` | decimal | **Required.** The amount of the transaction, as a decimal. Must be non-negative, and add up to the total value of the all the associated `TransactionItems`. |

Table 18-5 `MigrationTransaction` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `autoBillCycle` | int | **Required.** The billing sequence number for the Transaction within the life of the AutoBill. (**Note:** The first CashBox billing = 0.) |
| `billingDate` | dateTime | **Required.** The AutoBill's Billing Plan Period start date/time associated with the Transaction. |
| `billingPlan-Cycle` | int | **Required.** The billing sequence for the Transaction within the specified Billing Plan. |
| `currency` | string | **Required.** The ISO 4217 currency code used for this Transaction. Defaults to USD if not specified. |
| `divisionNumber` | string | The division or group with which this Transaction should be associated with your payment processor. Chase Paymentech refers to this number as the Division Number; Litle calls it the Report Group; MeS calls it the Profile ID. |
| `merchantAffil-iateId` | string | **Optional.** Your ID (a free-form string of 128 characters or less) for the affiliate that submitted this `Transaction` object, if any. |
| `merchantAffil-iateSubId` | string | **Optional.** Your ID (a free-form string of 128 characters or less) for the sub-affiliate that submitted this `Transaction` object, if any. |
| `merchantBill-ingPlanId` | string | **Required.** Your unique identifier for the `Billing-Plan` associated with this Transaction. The Billing-Plan must exist within CashBox prior to migrating Transactions that reference it. This field is required for Transactions included in an `AutoBill.migrate` request.<br>For more information, see Section 5: The BillingPlan Object. |
| `merchantTrans-actionId` | string | **Optional.** Your unique identifier for the Transaction (a free-form string of 128 characters or less, with no validation) . If not specified, this field will be populated by CashBox.<br>**Note:** For PayPal transactions, this value must be the INVNUM or INVOICEID field sent to PayPal for the transaction. |
| `migration-Transaction-Items` | `Migration-Transaction-Item` | **Required.** An array of `MigrationTransaction-Items` included with the `Transaction`.<br>For more information, see the `Migration-TransactionItem` Subobject. |

Table 18-5 `MigrationTransaction` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `nameValues` | `NameValuePair` | An optional array of name-value pairs you wish to associate with the Transaction.<br>`Transactions` generated as a result of the `AutoBill.modify` call will include a name-value pair with name `vin:type` and value `modify`.<br>See Section 10: The NameValuePair Object. |
| `paymentMethod` | `PaymentMethod` | **Required.** The Payment Method (e.g., a credit card) used for this Transaction. |
| `paymentProcessor` | string | The payment processor for this Transaction.<br>Possible values include FDMS, GlobalCollect, InComm, Litle, MeS, Orbital, PayFlowPro, PayPal, Paymentech, and Other.<br><br>If the Payment Processor is not supported by CashBox, the `migrationTransaction` will be imported, but other actions (i.e. refunds) will not be supported. If a value is not provided for this field, then CashBox will attempt to deduce the Payment Processor from your routing rules. |
| `paymentProcessorTransactionId` | string | The identifier assigned to this Transaction by your Payment Processor. |
| `preferredNotificationLanguage` | string | **Optional.** The language (specified as an ISO language string) for CashBox to use in email notifications for this Transaction. This value overrides any language setting in the `Account` object for this transaction. |
| `retryNumber` | integer | **Optional.** 0-based index indicating the billing attempt for a given Billing Period. For example, if this is the first billing attempt for a given Billing Period, the value will be 0. If the first billing attempt fails, and a second Transaction is attempted for the same Billing Period, the value will be 1.<br>If the `migrationTransaction` is included in an `AutoBill.migrate` request, but `retryNumber` is not specified, this field will default to 0. |
| `salesTaxAddress` | `Address` | The address used to calculate sales tax on this transaction. This field should be included if you include taxes in the Transaction. |
| `shippingAddress` | `Address` | **Optional.** The shipping address for this `Transaction` object.<br>**Note:** While optional, this field is useful in resolving chargebacks.<br>See Section 3.1: Address Data Members. |

Table 18-5 `MigrationTransaction` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `sourceIp` | string | **Optional.** The IP address (in standard dotted-quad form) of the machine from which the customer requested the creation of this `Transaction`. This attribute is required if you wish to score the Transaction for risk screening. Some payment methods, such as European Direct Debit, also require this attribute. |
| `statusLog` | `Transaction-Status` | **Required.** A log of `TransactionStatus` entries (with accurate timestamps) associated with this Transaction. At least one `TransactionStatus` object with a timestamp and status set to `Cancelled`, `Captured` or `Settled` must be included with every `AutoBill.migrate` call.<br>For CreditCard and ECP `PaymentMethod` objects, enter the `avsCode` and `cvnCode` to help the CashBox Chargeback team fight Chargebacks.<br>See the `TransactionStatus` Subobject. |
| `taxExemptions` | `TaxExemption` | An array of Exemptions that apply to this Transaction. Multiple tax exemptions may be defined.<br>See the `TaxExemption` Subobject. |
| `taxInclusive` | Boolean | A Boolean flag which defines whether the price listed for the Transaction is inclusive or exclusive of tax. If `true`, CashBox treats the `Migration-TransactionItem` price value as inclusive of the tax amount(s) when calculating the total cost of the Transaction. If `false`, CashBox adds the tax amount(s) to the `MigrationTransactionItem price` value when calculating the total cost of the Transaction. |
| `type` | `Migration-Transaction-Type` | The Transaction type: credit, recurring or non-recurring. For `MigrationTransactions` included in an `AutoBill.Migrate` request this will default to `Recurring`. For `MigrationTransactions` included in a `Transaction.Migrate` request, this value will default to `NonRecurring`. |
| `verification-Code` | string | The response from your verification system for this transaction (for example: Verified by Visa (VbV) or MasterCard SecureCode). Populate this field with your most recent payment verification information. |

## MigrationTransactionItem Subobject

A line-item in a MigrationTransaction. All line-items added together should add up to the total Transaction amount.

Table 18-6 MigrationTransactionItem Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| itemType | Migration-Transaction-ItemType | The Migrated Transaction Item's type, which may be one of three values:<br>• Credit: a one-time charge (not necessarily associated with an AutoBillItem).<br>• NonRecurringCharge: an after tax Credit applied to a Transaction.<br>• RecurringCharge: a one-time charge (not necessarily associated with an AutoBillItem).<br><br>If unspecified the type will default to RecurringCharge. |
| merchantAuto-BillItemId | string | **Optional.** Your unique identifier for the AutoBillItem associated with this MigrationTransaction-Item. Use this data member to distinguish between two or more AutoBillItems for the same Product. |
| migrationTax-Items | MigrationTax-Item | **Optional.** An array of tax line-items in a MigrationTransactionItem.<br>The MigrationTaxItems subobject contains three data members:<br>• amount: Tax amount in the currency of the overall transaction. (Required decimal.)<br>• jurisdiction: Sales tax jurisdiction. (Optional string.)<br>• name: Sales tax name. (Optional string.) |
| name | string | **Required.** A description of the item, which should match an existing Product description field. This is a free-form string of 256 or fewer characters. |
| price | decimal | **Required.** The price of the item, in the currency of the overall transaction.<br>Currencies may not be mixed on a single Transaction. This value must be zero or positive. |
| servicePeriod-EndDate | dateTime | **Required.** The entitlement end date for this item (generally associated with an AutoBill item). |
| servicePeriod-StartDate | dateTime | **Required.** The entitlement start date for this item (generally associated with an AutoBill item).<br>If unspecified, defaults to the MigrationTrans-action's billingDate. |

Table 18-6 `MigrationTransactionItem` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `sku` | string | **Required.** Your unique identifier for the product or service purchased with this `MigrationTransactionItem`. This value should match the `merchantProductId` for an existing Product , but it is not required to do so.<br>This is a free-form string of 256 or fewer characters. |
| `taxClassification` | string | The Item's tax classification. |

## `MigrationTransactionType` Subobject

Defines the migrated Transaction's type.

Table 18-7 `MigrationTransactionType` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `nonRecurring` | string | A one-time charge (not necessarily associated with an `AutoBill`). |
| `recurring` | string | A recurring charge (associated with an `AutoBill`). |

## TransactionItem Subobject

A line-item in a Transaction. Line items may be goods sold, sales tax, or other charges or credits. All line-items added together should add up to the total Transaction amount.

Table 18-8 TransactionItem Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| autoBillItem-Vid | string | Vindicia's unique identifier for the associated AutoBillItem. |
| campaignCode | string | Campaign code redeemed on this Transaction. To apply a Campaign, use this field to pass in a valid Coupon or Promotion code.<br>**Note:** This data member will not be returned. |
| campaignId | string | **Read only.** The unique identifier for a Campaign applied to this Transaction. This is a read-only field returned by CashBox for informational purposes. Values sent in with a SOAP call will be ignored. |
| merchantAuto-BillItemId | string | Your identifier for the associated AutoBillItem. |
| name | string | A description of the item. For CashBox-generated rebill transactions in which this Transaction item is derived from a Product object used with an AutoBill object, this value maps to the Product object's description attribute.<br>For TransactionItems which reflect Campaign discounts, this data member will be populated by CashBox with the text "Discount for ***description***," where ***description*** is the description data member for the ProductDescription subobject of the Product receiving the discount. |
| price | decimal | The item price, denominated by the currency data member of this Transaction object. |
| quantity | int | The number of items sold. If migrating quantity does not make sense, such as for a sales-tax line item, set quantity to 1, not 0. |

Table 18-8 `TransactionItem` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `servicePeriodEndDate` | dateTime | The start date for the service provided by this `TransactionItem`.<br>For standard AutoBills, these dates will coincide with the Billing Plan's bill dates. For AutoBills which include Season Sets, or other variants, these dates might be the same for multiple Transactions.<br>Blank indicates that the entitlement has no end date, and is valid forever, or that the Transaction resulted from a `Transaction.auth`, `capture`, or `migrate` call, in which case this value has no meaning.<br>**Note:** Service period start and end dates may not coincide with Billing Dates. For example, with installment-like Billing Plans, the start and end dates of every transaction are the dates of the full installment period, regardless of when billing occurs. |
| `servicePeriodStartDate` | dateTime | The start date for the time period reflected by this `TransactionItem`. |
| `sku` | string | **Optional.** Your SKU or other tracking key for this item. For CashBox-generated rebill transactions in which this transaction item is derived from a `Product` object used with an `AutoBill` object, this value maps to the `Product` object's `merchantProductId` attribute.<br>For `TransactionItems` which reflect Campaign discounts, this data member will be populated by CashBox with the text "Discount for ***merchantProductId***," where ***merchantProductId*** is the `merchantProductId` data member for the `Product` object receiving the discount. |
| `tax` | `Tax()` | An array of `Tax` objects, which include the following data members:<br>• `jurisdiction:` (string) the TransactionItem sku for the tax.<br>• `name:` (string) the description for the tax.<br>• `amount:` (decimal) the amount for the tax. |
| `taxClassification` | string | A string that defines the tax classification for this `TransactionItem`. |

Table 18-8 `TransactionItem` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `taxType` | string | This data member will be automatically populated by CashBox with applied tax information for the `TransactionItem`.<br>Possible values include `Inclusive Sales`, `Exclusive Sales`, `Inclusive Use`, and `Exclusive Use`. |
| `tokens` | `TokenAmount()` | An array of `TokenAmount` objects granted to the Account on this Transaction for purchasing this item (if CashBox tokens are in use). Each object in the array specifies the quantity of a specific type of token. This is a read-only attribute when CashBox returns the `TransactionItem` object to you in response to a call.<br>See Section 17.1: Token Data Members. |

## `TransactionStatus` Subobject

Lists the current `Status` for the Transaction.

**Note:** This subobject is required for the `AutoBill` and `Transaction.migrate` calls. With these calls, you must record at least one `TransactionStatus` object, with the `timestamp` and a status of `Cancelled`, `Captured`, or `Settled`.

Table 18-9 `TransactionStatus` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `boletoStatus` | `Transaction-StatusBoleto` | The status of a Boleto Bancário-based transaction. This field is populated with the `uri` received for this Transaction (the URL your payment processor received in response to a presentment of the fiscal number for the Transaction).<br>**Note:** CashBox does not support the Boleto Payment method for migrated Transactions.<br>See the `TransactionStatusBoleto` Subobject. |
| `carrier-BillingStatus` | `Transaction-StatusCarrier-Billing` | The status for a Carrier Billing based Transaction.<br>This object contains two data members:<br>`authCode`: Result code for the requested action.<br>`buyUrl`: URL which (when sourced on a customer's browser) generates HTML elements to facilitate processing of a `CarrierBilling` payment.<br>**Note:** CashBox does not support the Carrier Billing payment method for migrated Transactions. |
| `creditCardStatus` | `Transaction-StatusCredit-Card` | The most recently returned status of the credit-card-based transaction.<br>For migrated Transactions, populate this field with the payment-processor-specific details, such as the authorization code.<br>When reporting transactions to Vindicia for ChargeGuard, specify this attribute to help Vindicia dispute chargebacks.<br>See the `TransactionStatusCreditCard` Subobject. |
| *`directDebit-Status`* | *`Transaction-StatusDirect-Debit`* | *(This data member is not in use.)* |
| `ecpStatus` | `Transaction-StatusECP` | The status of an ECP-based transaction.<br>For migrated Transactions, populate this field with the most recent status received from your Payment Processor.<br>When reporting transactions to Vindicia for ChargeGuard, specify this attribute, to help Vindicia dispute chargebacks.<br>See the `TransactionStatusECP` Subobject. |
| `fundingSource-Balance` | decimal | The outstanding available balance on the submitted `PaymentMethod`. |

Table 18-9 `TransactionStatus` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `hostedPageStatus` | `TransactionStatusHostedPage` | Status details for a HostedPage Transaction.<br>**Note:** The customer's Account must exist before any Hosted Page related call references that Account.<br>See the `TransactionStatusHostedPage` Subobject. |
| `paymentMethodType` | `PaymentMethodType` | **Optional.** The type of payment method for this `Transaction` object. Depending on this value, you must also populate other `TransactionStatus` data members. For example, if you set the value of this data member to `CreditCard`, you must also populate the `creditCard` data member with the appropriate information.<br>If no value is entered for this data member with a `Transaction` or `AutoBill.migrate` call, CashBox will automatically populate it based on the `MigrationTransaction` object's `paymentMethod` data member.<br>See the `PaymentMethodType` Subobject. |
| `payPalStatus` | `TransactionStatusPayPal` | The status of a PayPal-based transaction. For one-time transactions, included in this attribute is the URL you must present to your customer for a visit to PayPal's site to complete the transaction process.<br>See the `TransactionStatusPayPal` Subobject. |
| `status` | `TransactionStatusType` | An enumerated string that specifies the transaction status.<br>For transactions processed through CashBox, this status is Vindicia's interpretation of a specific reason code received from your payment processor. Reason codes from payment processors vary from processor to processor, and are numerous. For one-time transactions, check this value in the `Transaction` object returned to you in response to your call to ensure that the processor has authorized the transaction.<br>See the `TransactionStatusType` Subobject. |
| `timestamp` | `dateTime` | A timestamp that specifies the date and time of when the `Transaction` status changed.<br>Required for migrated `Transactions`. |
| `vinAVS` | `AVSMatchType` | For transactions processed by CashBox, this value is Vindicia's interpretation of the AVS code returned by your payment processor.<br>For this field to be valid, you must enable AVS with your payment processor.<br>See the `AVSMatchType` Subobject. |

## `TransactionStatusBoleto` Subobject

Defines the status for a Credit Card transaction.

Table 18-10 `TransactionStatusBoleto` Object Data Member

| Data Member | Data Type | Description |
|---|---|---|
| `uri` | string | The URL returned by the payment processor in response to a presentment of the fiscal number. Send this string to the customer for further processing of the related transaction that uses the Boleto Bancário payment method. |

## `TransactionStatusCreditCard` Subobject

Defines the status for a Credit Card transaction.

Table 18-11 `TransactionStatusCreditCard` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `authCode` | string | The reason code returned by the payment processor when this `Transaction` object is authorized, captured, or cancelled. |
| `avsCode` | string | The AVS code returned by the payment processor when authorizing this `Transaction` object for one-time and migrated transactions. To receive this code, enable AVS with the payment processor. |
| `cvnCode` | string | The response sent by the payment processor for verification of the security code (the three- or four-digit number on the front or back of a credit card) for one-time and migrated transactions. |

## `TransactionStatusECP` Subobject

Defines the status for a Boleto Bancario transaction.

Table 18-12 `TransactionStatusECP` Object Data Member

| Data Member | Data Type | Description |
|---|---|---|
| `authCode` | string | The reason code returned by the payment processor when this `Transaction` object is authorized, captured, or cancelled. |

## TransactionStatusHostedPage Subobject

Defines the status for a Hosted Page transaction.

Table 18-13 TransactionStatusHostedPage Object Data Member

| Data Member | Data Type | Description |
|---|---|---|
| authCode | string | The result code for the status update. |
| redirectUrl | string | The Hosted Pages URL to which your customer should be redirected to complete a HostedPage Transaction. |

## TransactionStatusPayPal Subobject

Defines the status for a PayPal transaction.

Table 18-14 TransactionStatusPayPal Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| authCode | string | The success or failure return code received from PayPal after authorization is finalized. |
| payerId | string | Unique PayPal customer account identification number in PayPal's ExpressCheckout |
| redirectUrl | string | The PayPal URL to which you must redirect your customer to complete a PayPal transaction. |
| token | string | The token issued by PayPal Express Checkout. This token means that PayPal has tentatively accepted the transaction, and is awaiting further customer action. The token and the corresponding transaction will remain valid for a limited amount of time, during which the customer must complete the payment process on the PayPal site. |

## `TransactionStatusType` Subobject

Defines the Transaction Status.

Table 18-15 `TransactionStatusType` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `AuthExpired` | string | The transaction was not captured and the auth has expired. The transaction must be re-authorized and then captured. |
| `Authorization-Pending` | string | A PayPal-based transaction in CashBox that is awaiting further action by the customer on the PayPal site.<br>Do not interpret this status as authorization of payment. When a transaction is in this status, you should have sent your customer the PayPal URL at which to complete the payment process. |
| `Authorized` | string | A transaction authorized by the payment processor. This status indicates that the payment processor has approved this transaction but that the customer has not yet been charged. The actual charge will occur after transaction capture. |
| `AuthorizedFor-Validation` | string | A CashBox-generated transaction that is authorized to validate a payment method but that will not be captured, nor is the customer charged. CashBox generates transactions for small amounts (such as $1) and authorizes them with a payment processor to ensure the validity of a payment method, most commonly a credit card. These transactions may be ignored. |
| `Authorized-Pending` | string | A transaction that has passed initial validation but that has not yet been fully processed. This status is primarily for PayPal, ECP, and Boleto payment-based transactions that are awaiting action from the bank or the customer. |
| `Cancelled` | string | A cancellation, such as a rejection by the payment processor prior to capture, possibly before authorization. You may examine the reason code returned by the payment processor in the corresponding `status` object, for example, the `creditCardStatus` attribute. You can also cancel a transaction that is not yet captured by calling `cancel()`. |
| `Captured` | string | A captured status, which indicates that the payment processor has charged the customer. A captured transaction means that the payment processor has accepted it and that money transfer will take place. For most successful transactions processed by CashBox, this is the terminal status. |
| `New` | string | A brand-new transaction to be processed through CashBox with no past status. This status is often transient and soon changes if normal processing of the transaction continues. |

Table 18-15 `TransactionStatusType` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `Pending` | string | An incomplete transaction or one that is awaiting additional data. This status is mostly for internal use by CashBox. |
| `Refunded` | string | A CashBox-issued partial or full refund for this `Transaction` object. |
| `Settled` | string | A settled transaction. After the money transfer initiated by a captured transaction succeeds, the transaction is considered settled. Set this status when reporting a transaction to Vindicia for ChargeGuard. For transactions processed through CashBox, CashBox never sets this status because settlement is between you and the card-issuing bank, and is outside the scope of Vindicia's service. |
| `Void` | string | When a merchant cancels an auth, it becomes Void in the system. |

## `TransactionValidationResponse` Subobject

Returned from the `Transaction.migrate` call, this object describes a specific validation issue with a submitted transaction.

Table 18-16 `TransactionValidationResponse` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `code` | string | **Required.** A numerical code indicating the type of issue that was encountered.<br>Specific codes are listed below. |
| `description` | string | **Required.** A human readable description of the issue encountered. |
| `merchantTransactionId` | string | **Required.** Your unique ID for the submitted transaction. |

Table 18-17 `TransactionValidationResponse` Return Codes

| Return Code | Description |
|---|---|
| `200` | `The call succeeded.` |
| `400` | Your call failed, which could be due to an authentication failure or a CashBox failure to find any objects that match your input.<br>400 may also be one of the following:<br>• `Billing has already been attempted for Transaction ID` ***merchantTransactionId***.<br>• `Failed to deserialize Transaction.`<br>• `Invalid Arguments - No transaction object.` |
| `403` | `The Vindicia server cannot authenticate your request.` |
| `404` | One of the following:<br>• `Unable to load transaction: no match for merchantTransactionId` ***merchantTransactionId***.<br>• `Unable to load transaction: no match for VID` ***vid***. |
| `405` | `Unable to save transaction.` |
| `500` | The Vindicia server encountered an internal error. That error could occur for various reasons, the most common being an incorrectly populated input object, especially when you are making the call from a client library whose language does not support strict data-type checking. For resolution, especially during the development phase, contact Vindicia Technical Support. |
| `503` | `A Vindicia back-end service, such as a database, is un-available. Retry your call later.` |

## 18.3 `Transaction` Methods

The following table summarizes the methods for the `Transaction` object.

Table 18-18 `Transaction` Object Methods

| Method | Description |
|---|---|
| `addressAndSalesTaxFromPayPalOrder` | Allows you to fetch the billing and shipping addresses from PayPal. |
| `auth` | Sends this `Transaction` object to the payment processor for pre-authorization. |
| `authCapture` | Authorizes and captures this `Transaction` object in one call. |
| `calculateSalesTax` | Calculates the sales tax for this `Transaction` object. |
| `cancel` | Cancels a batch of previously authorized but not yet captured `Transaction` objects. |
| `capture` | Captures a batch of previously authorized `Transaction` objects. |
| `fetchByAccount` | Returns one or more `Transaction` objects whose `Account` object matches the input. |
| `fetchByAutobill` | Returns all the `Transaction` objects for an `AutoBill` object. |
| `fetchByMerchantTransactionId` | Returns a `Transaction` object whose transaction ID assigned by you (`merchantTransactionId`) matches the input. |
| `fetchByPaymentMethod` | Returns all the `Transaction` objects whose payment method matches the input. Identify the payment method with its VID, your payment method ID, or the payment-method-specific string, such as a credit-card account number. |
| `fetchByVid` | Returns a `Transaction` object whose VID matches the input. |
| `fetchByWebSessionVid` | Returns a `Transaction` object whose `WebSession` VID matches the input. |
| `fetchDelta` | Returns the `Transaction` objects whose status has changed since the last `fetchDelta` call. |
| `fetchDeltaSince` | Returns the `Transaction` objects that have been modified since the specified timestamp. (An `endTimeStamp` may also be specified.) |
| *`finalizeBokuAuthCapture`* | *(This method is not in use.)* |
| `finalizeCustomerAction` | Completes Transaction processing after your customer finishes payment activities at the payment provider-hosted web pages and is redirected to your site. |
| `finalizePayPalAuth` | Informs CashBox about the final authorization status of a transaction paid for with a PayPal-based payment method. |
| `migrate` | Allows you to migrate Transactions from a previous billing system to CashBox. |

Table 18-18 `Transaction` Object Methods *(Continued)*

| Method | Description |
|---|---|
| *`report`* | *(This method is not in use. Use `Transaction.migrate` to report Transactions to CashBox that have been processed in other billing systems.)* |
| `score` | Evaluates the risk score or chargeback probability score for this `Transaction` object. |

## addressAndSalesTaxFromPayPalOrder

The `addressAndSalesTaxFromPayPalOrder` method allows you to fetch the billing and shipping addresses from PayPal, and apply tax to Transactions

This call will calculate taxes using the billing address obtained from PayPal, and is recommended for merchants who do not collect this address information from their customers.

Billing and shipping addresses are only applied to the current Transaction and will ***not*** be stored in CashBox for use in subsequent Transactions. If you wish these values to be stored for use in later one-time or recurring PayPal transactions, you must do so manually.

| | |
|---|---|
| **Note:** | You must be approved by PayPal, and your Seller Account enabled for the Billing Address feature, to use this method successfully. Once you have established this relationship with PayPal, please work with your Vindicia Client Services representative to enable the feature for your CashBox account.<br>The Shipping Address will be always returned by this call, even without completing these required steps for the Billing Address return. |

### *Input*

***payPalTransactionId:*** Vindicia's ID for the PayPal payment method validation `Transaction`, generated when you called **AutoBill.`update`**. Retrieve this ID from the value associated with the name: `vindicia_vid` in the name–value pairs attached to the redirect URL.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transaction:*** an object of type Transaction

***SalesTaxAddress:*** an object of type `Address` that describes the PayPal listed sales tax address for the Transaction.

***BillingAddress:*** an object of type `Address` that describes the PayPal listed billing address for the Transaction.

***ShippingAddress:*** an object of type `Address` that describes the PayPal listed shipping address for the Transaction.

***taxItems:*** an object of type `SalesTax` that describes the total amount for taxable items included with the Transaction.

***totalTax:*** the total amount of tax levied against the Transaction.

***subtotalAmount:*** the pre-taxed total for the Transaction.

***totalAmount:*** the post-tax total for the Transaction.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

## *Examples*

The following examples are for One-Time and Recurring Transactions. Both of these examples should be called on your PayPal Success page, after your Buyer has approved the Transaction.

> **Note:** These examples differ only in that Recurring Transactions require that a separate object be created for the AutoBill to use to call `finalizePayPalAuth`.

One-Time Transactions may use the same Transaction object for both Transaction.addressAndSalesTaxFromPayPalOrder and `finalizePayPalAuth`.

## *One-Time*

The following example demonstrates use of this method for a One-Time Transaction.

```
$transaction = new Transaction();

// Obtain the id of the PayPal transaction from the redirect URL.

$payPalTxId = $_GET['vindicia_vid'];

// For a successfully authorized PayPal transaction,
// set the success input parameter to true.

$success = true;

// Fetch the Billing and Shipping Addresses from PayPal,
// and apply Tax to the Transaction using the returned addresses.

$response =
   $transaction->addressAndSalesTaxFromPayPalOrder($payPalTxId);

// Update the PaymentMethod.billingAddress with the
// Billing Address returned by PayPal so it will be used
// to apply Tax to subsequent Transactions based on the Billing Address.

// (Optional.) Update Account.shippingAddress with
// the returned Shipping Address, so it will be used
// to apply Tax to subsequent Transactions
// (if there is not an existing Account.shippingAddress stored).

// To obtain Buyer confirmation of the modified
// Transaction amount, which now includes Tax,
// you must first interact with the Buyer in the User Interface,
// then, after Buyer approval, proceed to complete the Transaction.

// Finalize the Transaction

$response =
   $response->finalizePayPalAuth($payPalTxId, $success);

if($response['returnCode'] == 200) {
   $txId = $response['transaction']->getMerchantTransactionId();
   printLog "Transaction authorized: " . $txId;
}
```

### *Recurring*

The following example demonstrates use of this method for a Recurring (AutoBill) Transaction.

```
$autobill = new AutoBill();
$transaction = new Transaction();

// Obtain the id of the PayPal transaction from the redirect URL.

$payPalTxId = $_GET['vindicia_vid'];

// For a successfully authorized PayPal transaction,
// set the success input parameter to true.

$success = true;

// Fetch the Billing and Shipping Addresses from PayPal,
// and apply Tax to the Transaction using the returned addresses.

$response =
   $transaction->addressAndSalesTaxFromPayPalOrder($payPalTxId);

// Update the PaymentMethod.billingAddress with the
// Billing Address returned by PayPal so it will be used
// to apply Tax to subsequent Transactions based on the Billing Address.

// (Optional.) Update Account.shippingAddress with
// the returned Shipping Address, so it will be used
// to apply Tax to subsequent Transactions
// (if there is not an existing Account.shippingAddress stored).

// To obtain Buyer confirmation of the modified
// Transaction amount, which now includes Tax,
// you must first interact with the Buyer in the User Interface,
// then, after Buyer approval, proceed to complete the Transaction.

//Finalize the Transaction:

$response =
   $autobill->finalizePayPalAuth($payPalTxId, $success);

if($response['returnCode'] == 200) {
   $txId = $response['transaction']->getMerchantTransactionId();
   printLog "Transaction authorized: " . $txId;
}
```

## auth

The `auth` method sends a transaction to a payment processor for authorization before a capture operation. Call this method for one-time transactions when you want to bill a customer for a specific purchase. Used with the `capture()` call, this call is useful if the purchase involves shipping of physical goods. For such purchases and in some other situations, payment processors typically mandate that you not receive payment until you have shipped the goods to the customer. Before shipping or beginning the delivery, call `auth()` to determine the customer's ability to pay and, after shipment, call `capture()` to receive payment.

You may also call `auth()` to simply validate a payment method, because the call does not charge the customer. However, because `auth()` requires CashBox to call your payment processor on your behalf, a cost is involved. For each transaction authorized, the payment processor typically charges a fee as stipulated in your contract. To avoid this fee, Vindicia recommends that you prescreen transactions for fraud risk before authorizing them with your payment processor. You can do so by specifying an acceptable risk score (less than 100) in the `minChargebackProbability` parameter of this call. For details on fraud risk screening, see Chapter 14: Common ChargeGuard Programming Tasks in the ***CashBox Programming Guide*** and the `score` method.

**Note:** The chargeback risk score is evaluated first, and, if it fails, is returned first.

Note that this call only *authorizes* the transaction with your payment processor. The processor's approval, indicated by the `Authorized` status set in the `Transaction` object returned by this call, means that the payment processor will initiate a fund transfer when you make a call to *capture* the transaction. **Note:** the `Authorized` status does not mean that the customer will be charged for this transaction. If a transaction involves the shipment of goods, call `auth()` after receiving the order. The `Authorized` status indicates that the customer will be able to pay. After shipping the order, call `capture()` (typically in batch mode, to process multiple transactions authorized over a period of time) to charge the customers in question.

Calling `auth()` also enables you to further validate a transaction before its capture. For example, for credit-card-based one-time transactions, `auth()` returns a `Transaction` object that contains a `TransactionStatus` object, which not only indicates whether the payment processor has approved the transaction, but also includes the processor's responses to AVS (address verification) and CVN (credit-card security code) verifications, assuming that you have enabled those services with the processor. If the responses are not satisfactory to you, you can make a call to cancel the transaction and thus never capture it.

For more detail on AVS and CVN Return Codes, please work with your Vindicia Client Services representative.

The meaning of a transaction's authorization varies from payment method to payment method. For example:

- If you are conducting an ECP-based inbound transaction, the authorization returned by your payment processor in response to the `auth()` call means that the processor has only verified that the bank account and routing number specified by the customer on the payment method are not in the negative file ("blacklist") maintained by the processor. `auth()` does not guarantee that the customer has enough funds in their bank account to pay for the transaction.
- CashBox does not support the `auth()` call for one-time transactions whose payment method is Boleto Bancário.
- For PayPal-based transactions, the `auth()` call returns a PayPal URL in the `TransactionStatus` object, which you must present to your customer. The transaction is considered authorized only after the customers has visited the URL, and successfully completed the payment process required by PayPal.

The authorization that you obtain from your payment processor through the `auth()` call is usually valid for only a few days. To charge the customer and collect the funds associated with an authorized transaction, you must call `capture()` on it. For some payment processors, CashBox explicitly voids authorized transactions that have not been captured within a certain period of time.

The `auth()` call also adds applicable sales-tax line items to your `Transaction` before authorizing it and, if it is authorized, scheduling it for capture. For tax calculation, you must work with Vindicia Client Services to define and capture your tax nexus, that is, the state and local governments that can legally tax your sales. Also, be certain to indicate the appropriate tax classification on your `Transaction` items.

The `auth` call will handle a tax-based timeout, returning a `202` error if the tax calculation has timed out. Given this error, you may choose to abandon or cancel the `Transaction`. If you ignore this error, the related `capture` will recognize the failed timeout, and recalculate based on tax-inclusive prices.

**Note:** `Transaction.auth` allows you to set your own `minChargebackProbability` threshold, while `Transaction.authCapture` uses the built-in CashBox AVS/CVN policy evaluation. Use `Transaction.auth`, rather than `Transaction.authCapture`, only with compelling reason.

## *Input*

***transaction:*** the `Transaction` object for preauthorization. Identify this object using either its VID or your transaction ID (`merchantTransactionId`).

**Note:** `PaymentMethods` may not be duplicated for an `Account`. Passing in an existing credit card number and expiration date (in the `sourcePaymentMethod` for the `Transaction`) in an attempt to create a new `PaymentMethod` for an `Account` will return the pre-existing `PaymentMethod` instead.

***minChargebackProbability:*** a number between 0 and 100 by which you specify your fraud risk score tolerance level. A chargeback probability (also called the risk-screening score or risk score) of 100 indicates that CashBox is 100% certain that a transaction is fraudulent and will result in a chargeback. Specify your acceptable threshold for chargeback possibility with this parameter. If the score evaluates to be more than your tolerance level, the `auth` call will fail.

If you do not specify this parameter, it defaults to a value of 100, meaning no risk screening, in which case the `Transaction` is always acceptable to you (unless it fails). In order for Vindicia to successfully evaluate a transaction's risk score, the transaction must have certain minimum information, such as the IP address, billing city, state, and country. For details on Vindicia's risk-screening features, see Chapter 14: Common ChargeGuard Programming Tasks in the ***CashBox Programming Guide***.

***sendEmailNotification:*** a Boolean flag that, if set to `true`, triggers an email notification from CashBox to the `Account` object for this `Transaction` object. Use the `Transaction` data member `preferredNotificationLanguage` to set the language for the notification. (For more information, see Section 9.1: Setting the Preferred Language in the ***CashBox Programming Guide***.)

***campaignCode:*** the Coupon or Promotion Code used to obtain a discount on this `Transaction`. (This discount will be applied to all eligible Transaction items.)

***dryrun:*** a Boolean flag that, if set to `true`, will return the updated `Transaction`, without recording the result in the CashBox database. Use this method to compute the cost of a `Transaction` without committing to the change.

If the `Transaction` did not exist before, it will not exist afterward; if it did exist before, it will not change. (No payment method validations, authorizations or charges will be performed if ***dryrun*** is true.)

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transaction:*** the original `Transaction` object, with several attributes added by CashBox during processing, including the `Transaction`'s latest status, which will list the success or failure of the `auth`.

***score:*** the `Transaction` object's risk score, which represents the estimated probability that this transaction will result in a chargeback. This number ranges from 0 (best) to 100 (worst). It can also be -1, meaning that Vindicia has no opinion. (-1 indicates a transaction with no originating IP addresses, an incomplete addresses, or both. -2 indicates an error; retry later.)

If the score is not acceptable, contact the customer for more information and then re-call this method for a new score.

***scoreCodes:*** an array of `ScoreCode` objects that explain the score. Each object contains two attributes: `id` and `description`. See Table 18-21: Score Code Descriptions.

## Returns

If successful, the `auth()` call returns a `returnCode` value of `200` along with the transaction status in the first (and latest) entry in the `statusLog` array. A `200` code does not necessarily mean that your transaction has been approved by the payment processor. For example, if your processor denies the transaction, CashBox sets a status of `Cancelled` in the latest entry in the `statusLog` array in the returned `Transaction` object, but the return code still remains `200`.

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 202 | Taxes temporarily unavailable. |
| 400 | One of the following:<br>• Must specify line items in transaction to calculate sales tax for auth!<br>• Data validation error ***error-description.***<br>• Must specify transaction to authorize!<br>• Auth attempt failed to return a valid Transaction.<br>• Vindicia fault ***fault-code*** encountered.<br>• ***Internal-error-description.***<br>• Data validation error Failed to create Payment-Type-Specific Payment Record: Credit Card conversion failed: Credit Card failed Luhn check. |
| 407 | Failed AVS policy evaluation. |
| 408 | Failed CVN policy evaluation. |
| 402 | One of the following:<br>• Can't call auth on Boleto associated transaction. Please call authCapture!<br>• The transaction ID ***merchantTransactionId*** collides with reserved Vindicia namespace, which is: ***namespace.***<br>• Unable to create transaction ID consistent with reserved Vindicia namespace, which is: ***namespace.***<br>• No payment method found in transaction or account.<br>• Transaction already previously authorized! |
| 406 | Chargeback risk score is higher than minChargebackProbability, transaction not authorized.<br>(Vindicia saves the unauthorized transaction as a cancelled transaction, and returns a SOAP transaction object in $rc.) |

## *Example*

```
// to authorize a credit card-based transaction
// with risk screening enabled

$tx = new Transaction();
$tx->setAmount('9.90');
$tx->setCurrency('USD');
$tx->setMerchantTransactionId('txid-123456');
$tx->setSourceIp('189.201.45.7');

// Reference an existing account by its ID
$account = new Account();
$account->setMerchantAccountId('9876-5432');
$tx->setAccount($account);

// Different shipping address from Account?
$shippingAddress = new Address();
$shippingAddress->setName('Jane Doe');
$shippingAddress->setAddr1('44 Elm St.');
$shippingAddress->setAddr2('Apt 55');
```

```
$shippingAddress->setCity('San Mateo');
$shippingAddress->setDistrict('CA');
$shippingAddress->setPostalCode('94403');
$shippingAddress->setCountry('US');
$shippingAddress->setPhone('650-555-3444');
$shippingAddress->setFax('650-555-3445');

$tx->setShippingAddress($shippingAddress);

// The line items of the transaction
$tx_item = new TransactionItem();
$tx_item->setSku('sku-1234');
$tx_item->setName('Widget');
$tx_item->setPrice('3.30');
$tx_item->setQuantity('3');
$tx->setTransactionItems(array($tx_item));

$paymentMethod = new PaymentMethod();
$paymentMethod->setType('CreditCard');

// Populate rest of the payment method object here.
// Make sure payment method has full billing address
// in it or the risk screen will not work

…

$tx->setSourcePaymentMethod($paymentMethod);

// make the auth call here. We can tolerate a risk score below
// 70 and do not want to send an email notification to
// the customer
$response = $tx->auth(70, false);

if($response['returnCode']==200) {
    $ret_tx = $response['data']->transaction;

    if($ret_tx->statusLog[0]->status=='Authorized') {
        print "Transaction approved";
    }
    else if($ret_tx->statusLog[0]->status=='Cancelled') {
        print "Transaction not approved \n";
        print "Reason code is: ";
        print $ret_tx->statusLog[0]->creditCardStatus->authCode;
        print "\n";
    }
    else {
        print "Error: Unexpected transaction status\n";
    }
}
else if ($response['returnCode']==403) {
    print "Transaction cannot be processed due to high fraud potential\n";
}
else {
    print "Error while making call to Vindicia CashBox\n";
}
```

## authCapture

The `authCapture` method combines auth and capture functionality. It authorizes a transaction with your payment processor in real time, and schedules it for capture simultaneously. CashBox performs the capture with your payment processor in batch mode at periodic intervals. AVS and CVN policy settings determine whether or not the `authCapture` call will succeed.

> **Note:** For more information on AVS and CVN Return Codes, please work with your Vindicia Client Services representative.

The `authCapture` call also adds applicable sales-tax line items to your `Transaction` before authorizing it and, if it is authorized, scheduling it for capture. Work with Vindicia Client Services to define which state and local governments can legally tax your sales. Be certain to indicate the appropriate tax classification on your transaction items.

The `authCapture` call will handle a tax-based timeout, returning a `202` error if the tax calculation has timed out. Given this error, the automatic capture is postponed by a configurable delay which defaults to one hour, during which you may explicitly cancel the `Transaction`. If you ignore this error, the related capture will recognize the failed timeout, and recalculate based on tax-inclusive prices.

This call is used to process one-time (real-time) transactions through CashBox. Call `auth()` to preauthorize a customer's order before shipment and, after shipment, call `capture()` to capture the transaction. If the order does not involve shipment of physical goods, you may call `authCapture` to both authorize and capture the transaction.

This call returns the `Transaction` object with a `TransactionStatus` object (first entry in the array in the `statusLog` attribute) populated with results of the real-time authorization obtained from your payment processor. If the authorization result is positive (`Authorized` status), CashBox schedules the transaction for capture. Otherwise, CashBox sets the status to `Cancelled`.

By default, `authCapture` examines the AVS and CVN return codes, issued by your payment processor in response to the `auth` call, to determine whether to process the call. To ignore the CashBox evaluation of the AVS/CVN return code, and process the `Transaction` regardless of their result, set the `ignoreAvsPolicy` and `ignoreCvnPolicy` flags to true.

If there is a policy failure, the `capture` will be aborted.

> **Note:** The customer's Account must exist before any Hosted Page related call references that Account.

## Input

***transaction:*** the `Transaction` object to authorize and capture. Identify this object with either its VID or your transaction ID (`merchantTransactionId`).

> **Note:** `PaymentMethods` may not be duplicated for an `Account`. Passing in an existing credit card number and expiration date (in the `sourcePaymentMethod` for the `Transaction`) in an attempt to create a new `PaymentMethod` for an `Account` will return the pre-existing `PaymentMethod` instead.

***sendEmailNotification:*** a Boolean flag that, if set to `true`, triggers an email notification from CashBox to the `Account` object for the `Transaction` object. Use the `Transaction` data member `preferredNotificationLanguage` to set the language for the notification. (For more information, see Section 9.1: Setting the Preferred Language in the ***CashBox Programming Guide***.)

***ignoreAvsPolicy:*** a Boolean flag that, if set to `true`, will override the AVS policy, and update the `paymentMethod`, regardless of the AVS return code. If set to `false` or `null`, the AVS return code will be used to determine whether to update the `paymentMethod`.

***ignoreCvnPolicy:*** an optional Boolean flag that, if set to `true`, will override the CVN policy, and update the `paymentMethod`, regardless of the CVN return code. If set to `false` or `null`, the CVN return code will be used to determine whether to update the `paymentMethod`.

***campaignCode:*** the Coupon or Promotion Code used to obtain a discount on this `Transaction`. (This discount will be applied to all eligible Transaction items.)

***dryrun:*** a Boolean flag that, if set to `true`, will return the updated `Transaction`, without recording the result in the CashBox database. Use this method to compute the cost of a `Transaction` without committing to the change.

If the `Transaction` did not exist before, it will not exist afterward; if it did exist before, it will not change. (No payment method validations, authorizations or charges will be performed if ***dryrun*** is true.)

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transaction:*** the `Transaction` object that contains a `TransactionStatus` object, which encapsulates the results of real-time authorization (also called online authorization) obtained from the payment processor. If this transaction is approved by the processor, CashBox has already scheduled it for batch capture.

## *Returns*

If successful, the `authCapture()` call returns a `returnCode` value of `200` along with the transaction status in the first (and latest) entry in the `statusLog` array. That `200` code does not necessarily mean that your transaction has been approved by the payment processor. For example, if your processor denies the transaction, CashBox sets a status of `Cancelled` in the latest entry in the `statusLog` array in the returned `Transaction` object, but the return code still remains `200`.

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `202` | `Taxes temporarily unavailable.` |
| `400` | One of the following:<br>• `Must specify line items in transaction to calculate sales tax for auth!`<br>• `Data validation error` ***`error-description.`***<br>• `Must specify transaction to authorize!`<br>• `Auth attempt failed to return a valid Transaction.`<br>• `Vindicia fault` ***`fault-code`*** `encountered.`<br>• ***`Internal-error-description.`***<br>• `Data validation error Failed to create Payment-Type-Specific Payment Record: Credit Card conversion failed: Credit Card failed Luhn check.` |
| `402` | One of the following:<br>• `The transaction ID` ***`merchantTransactionId`*** `collides with reserved Vindicia namespace, which is:` ***`namespace`***.<br>• `Unable to create transaction ID consistent with reserved Vindicia namespace, which is:` ***`namespace`***.<br>• `No payment method found in transaction or account.`<br>• `Transaction already previously authorized!` |
| `409` | `AVS and CVN policy evaluations failed.` |
| `410` | `AVS and CVN policy evaluations could not be performed.` |

## Examples

The following examples are for credit card, Boleto Bancário, ECP, and PayPal.

### Credit-Card Payment Method

The following example creates, populates, authorizes, and captures a `Transaction` object with `CreditCard` as the payment method. The code checks if the `Transaction` status after an `authCapture()` call is `Authorized`. If so, the payment processor has authorized the transaction, and CashBox has marked it for capture with the processor. However, if the status is `Cancelled`, it means that the payment processor has denied the transaction.

```
$tx = new Transaction();
$tx->setAmount('9.90');
$tx->setCurrency('USD');
$tx->setMerchantTransactionId('txid-123456');

$paymentMethod = new PaymentMethod();
$paymentMethod->setBillingAddress($address);
$paymentMethod->setType('CreditCard');

$card = new CreditCard();
$card->setAccount('4444222211113333');
$card->setExpirationDate('xxxxxx'); // Use YYYYMM format for date
$paymentMethod->setCreditCard($card);

$nv = new NameValuePair();
$nv->setName("CVN");
$nv->setValue("123"); // this is the card security code provided by customer

// set the card security code inside the payment method
$paymentMethod->setNameValues(array($nv));

$tx->setSourcePaymentMethod($paymentMethod);

// set other transaction attributes here

// make the authCapture call
$sendEmailNotification = true;
$ignoreAvsPolicy = true;
$ignoreCvnPolicy = true;
$response = $tx->authCapture($sendEmailNotification, $ignoreAvsPolicy,
          $ignoreCvnPolicy);

if ($response['returnCode'] == 200) {
   if ($tx->statusLog[0]->status == 'Authorized') {
      print "Card approved.\n";
   }
   else ($tx->statusLog[0]->status == 'Cancelled') {
      // The transaction did not go through
      print "Declined. Reason code received from
         payment processor: ";
      print $tx->statusLog[0]->status->creditCardStatus->authCode . "\n";
   }
}
```

### *Boleto Bancário Payment Method*

For the Boleto Bancário payment method, the transaction success status after an `authCapture()` call is `Authorized`. That means that CashBox has validated the fiscal number and the payment processor has accepted it. In response, the payment processor returns a URL in the `TransactionStatus` object. That URL contains further instructions for completing the transaction and is actually a payment document the customer must print and take to their bank. After the call is complete, CashBox changes the transaction status to `AuthorizedPending` to indicate that CashBox is awaiting customer action and further response from the payment processor.

Present the URL returned by this call to your customer. When the transaction is complete, the payment processor notifies CashBox, which then updates the status to `Captured` or `Cancelled`, depending on the success or failure of the transaction. This step might take several days, because it requires that the customer physically present the payment document to the bank.

The following example creates, populates, and sets a fiscal number for a `Transaction` object with Boleto Bancário as the payment method.

```
$txn = new Transaction();

// Populate the transaction as shown in the previous example.
// When associating a customer account with this transaction ensure
// that the account has language preference indicated. This will set
// the language to be used in the payment instructions
// displayed to the customer

$tx->setAccount($account);

$paymentMethod = new PaymentMethod();

// For Boleto payment make sure country is specified in the address

$paymentMethod->setBillingAddress($address);

$paymentMethod->setType('Boleto');
$blt = new Boleto();
$blt->setFiscalNumber('123456789');
$paymentMethod->setBoleto($blt);
// populate payment method billing address, country must be specified

$tx->setSourcePaymentMethod($paymentMethod);
$sendEmailNotification=false;

$response = $tx->authCapture($sendEmailNotification);

if($response['returnCode']==200) {
    $ret_tx = $response['data']->transaction;
    if($ret_tx->statusLog[0]->status=='Authorized') {
        print "Successful\n";
        display(print $ret_tx->statusLog[0]->status->boletoStatus ->uri);
    }
    else if($ret_tx->statusLog[0]->status=='Cancelled') {
        // The transaction was denied
    }
}
```

| **Note** | For the Boleto Bancário payment method, be certain to specify the country in the payment method billing address, and the language preference in the customer account. Those two attributes set the language used in customer communications. |
|---|---|

### *ECP Payment Method*

For the ECP payment method, the status of a `Transaction` immediately after an `authCapture()` call is `Authorized`, which means that the payment processor has performed a real-time validation of the payment information to ensure, for example, that the bank routing number is not blacklisted. To configure this validation for a more thorough check, contact Vindicia Client Services.

Next, CashBox submits the transaction to the payment processor for deposit or withdrawal from the specified bank, and changes `Transaction` status to `AuthorizedPending`, meaning that processing of the Transaction has begun.

Six banking days must elapse before CashBox sets the status to `Captured`. During that time, if CashBox receives notice from the payment processor that the transaction has failed, CashBox changes the `Transaction` status to `Cancelled`.

If the reason code indicates that the payment processor will attempt a retry (for example, due to insufficient funds), CashBox changes the `Transaction` status to `RetryPending`. The retry date depends on the retry schedule that the payment processor has previously defined with you according to your division ID. Be certain to provide Vindicia with your division ID's retry schedule.

If CashBox does not receive any decline codes for six banking days after the retry, CashBox sets the `Transaction` status to `Captured`. The following code example creates and populates a `Transaction` object with ECP as the payment method.

```
$txn = new Transaction();

// populate the transaction as shown in the previous example
$paymentMethod = new PaymentMethod();
$paymentMethod->setBillingAddress($address);
$paymentMethod->setType('ECP');

$ecp = new ECP();

// specify account number where funds will be with withdrawn from
$ecp->setAccount('123456789');

// specify bank routing number
$ecp->setRoutingNumber('3409284043');
$ecp->setAccountType('ConsumerChecking');
$paymentMethod->setECP($ecp);

// If this is an inbound payment i.e. a withdrawal from specified
// bank account and deposit into merchant's account set source
// payment method in the transaction.
// For paying out i.e. a deposit into specified bank account
// and withdrawal from merchant's bank account, set destination
// PaymentMethod attribute of the transaction

$tx->setSourcePaymentMethod($paymentMethod);
$tx->setEcpTransactionType('Inbound');
```

```
$sendEmailNotification = false;
$response = $tx->authCapture($sendEmailNotification);

if($response['returnCode']==200) {
    $ret_tx = $response['data']->transaction;
    if($ret_tx->statusLog[0]->status=='Authorized') {
        print "Successful\n";
    }
    else if($ret_tx->statusLog[0]->status=='Cancelled') {
        // The transaction did not go through
        print "Declined.
            Reason code received from payment processor: ";
        print $ret_tx->statusLog[0]->status->ecpStatus->authCode
            . "\n";
    }
}
```

### *PayPal Payment Method*

For the PayPal payment method, the transaction status after an `authCapture()` call is `AuthorizationPending`. The payment flow for PayPal-based real-time transactions proceeds as follows:

1. When a customer clicks the PayPal button on your site, create a `Transaction` object that specifies PayPal as the payment method and makes a `Transaction->authCapture()` call to CashBox.
2. When that call returns, examine the status of the returned `Transaction` object. If the status is not a failure (`Cancelled`), it is `AuthorizationPending`, meaning that the transaction is in the CashBox and PayPal systems, and that it requires further action from the customer for completion.
3. PayPal notifies CashBox of the successful creation of the transaction by issuing a PayPal token, which keeps the transaction valid for the next few hours.
4. The returned `Transaction` object contains a PayPal-specific status along with a URL, which contains the token information. Redirect the customer to that URL to complete PayPal's payment sequence.
5. Depending on the customer's success or failure in completing the payment process, PayPal redirects the customer to a success or failure URL on your site. (Provide CashBox with the success and failure URLs as attributes named `returnUrl` and `cancelUrl`, respectively, of the PayPal payment method for the `Transaction`.) From this page, make a call to CashBox to finalize the PayPal authorization so that CashBox can update the status of the `Transaction`. This call requires you to pass in the ID of the `Transaction`, which you can find in redirected URL. It is value associated with the name `vindicia_vid` in the redirect URL

The following example illustrates the process.

```
$tx = new Transaction();

// populate the transaction as shown in earlier examples

$paymentMethod = new PaymentMethod();
$paymentMethod->setType('PayPal');

$payPal = new PayPal();

// This is the URL the customer will be redirected to after they
// arrive at the Vindicia landing page after completing the payment
// process at PayPal's site
$payPal->setReturnUrl('http://myshoppingcart.merchant.com');

// specify bank routing number
$payPal->setCancelUrl('http://tryagain.merchant.com');
$paymentMethod->setPayPal($payPal);
$tx->setSourcePaymentMethod($paymentMethod);
$sendEmailNotification = false;
$response = $tx->authCapture($sendEmailNotification);

if($response['returnCode']==200) {

   if($tx->statusLog[0]->status=='AuthorizationPending') {
      $payPalUrl = $tx->statusLog[0]->payPalStatus->redirectUrl;

      // send customer to this URL for completion of payment
      // process at PayPal's site
   }
}
```

After successfully completing the payment process, the customer is redirected to the URL *www.myshoppingcart.merchant.com*, which is the return URL in the PayPal-based `PaymentMethod` object. From this page, finalize the Transaction so that CashBox will acquire its status.

```
$soap_caller = new Transaction();

// obtain id of the PayPal transaction from the redirect URL.
// It is the value associated with name 'vindicia_vid'

$payPalTxId = … ;

// if calling from the return URL reached when the PayPal
// transaction is successfully authorized, set the
// success input parameter to true, from the cancelUrl, set the
// success input parameter to false. Let's assume success here:

$success = true;
$response =
   $soap_caller->finalizePayPalAuth($payPalTxId, $success);

if($response['returnCode'] == 200) {
   printLog "Transaction authorized";
}
```

Upon completion, CashBox updates the `Transaction` status to `Authorized`, which changes to `Captured` after CashBox batch-processes this and other PayPal transactions.

## calculateSalesTax

The `calculateSalesTax` method calculates the sales tax of a `Transaction` object.

Transactions may be taxable by several local and state governments. For example, in the United States, depending on the address, a transaction might be taxable by the city, county, and state. For each applicable tax, this method adds a line item to your `Transaction` (see the `Transaction` object's `items` data member).

The CashBox sales-tax engine works as follows:

1. Taxes are collected according to the buyer's address. If the shipping address is specified on the `Transaction`, CashBox considers that address for tax calculation first. If not, CashBox uses the billing address on the payment method. In the absence of those two addresses, CashBox cannot calculate the taxes. For U.S. and Canadian addresses, be sure to provide full address information since taxes vary from state to state and, in many cases, from city to city.
2. CashBox "cleans up" the address chosen to apply taxes. For example, CashBox converts SAINT FORT, SAINTE FORT, or STE FORT to ST FORT, discards punctuation marks, and converts dashes to spaces.
3. CashBox "fixes up" the address in question, by correcting misspelled street or city names, and by applying the correct postal code according to the street address. CashBox does not change the actual address in the `Transaction` object; instead, CashBox stores the corrected address in the `Transaction` object's `salesTaxAddress` data member when returning the object to you. This step enables the CashBox sales-tax engine to pinpoint the correct final jurisdiction (country, district, county, city, and postal code) to calculate taxes.
4. CashBox looks in a database for the applicable tax rates for the jurisdiction. That database is continually updated with the latest information.

Customize the applicable tax rates as follows:

- **Upload overriding tax rules to the Vindicia database.** In those rules, you may define a specific tax rate for CashBox to apply to your transactions if the customer address is in a specific city, county, state, or other location. You may also specify a date range for applying those tax rules. For more information, contact your Vindicia Client Services representative.
- **Specify your tax nexus.** In the United States, your tax nexus is the set of local and state governments that may collect sales tax on your transactions. This nexus depends on the physical location of your business registration. For example, if your company is registered only in California, only the State of California may collect sales tax on your transactions, and CashBox applies sales tax only if your customer's address is also in California. Contact your Vindicia Client Services representative for more information.
- **Define the tax exemptions on your customer accounts.** See the `taxExemptions` attribute in Section 1: The Account Object.

- **Define the tax classification on your `Product` and `TransactionItem` objects.** The tax classification enables you to specify the categories, such as physical goods and electronic data, to which your sales items belong. If your nexus specifies that an item is taxable, CashBox applies sales tax accordingly. See the `taxClassification` attribute in Section 13: The Product Object, and in the `TransactionItem` Subobject.

CashBox includes sales-tax items added to your Transaction as new items in the returned `Transaction` object. The names of those transaction items begin with the prefix `VIN`, for example, `VIN_SALES_TAX_STATE`. CashBox also adds a line item that contains the total amount of all the tax items with the name `VIN_SALES_TAX`.

Note that the `calculateSalesTax` method does not ***save*** the transaction sent for tax calculation in the CashBox database. When a customer makes a one-time purchase on your site, create a `Transaction` object and call `calculateSalesTax` on it to calculate the applicable taxes. CashBox will return the total amount of the purchase after adding the applicable taxes. Then present the amount to your customer. Once the customer has finalized the purchase, capture the transaction by calling `authCapture` on the original `Transaction`.

The `authCapture()` and `auth()` methods automatically calculate and add taxes to a transaction before processing it with the payment processor. CashBox also adds applicable sales tax to recurring billing transactions generated for `AutoBill` objects.

## *Input*

***transaction:*** the `Transaction` object for which to calculate sales tax. This object must have an address and a line item that describes the product sold, as well as a price. Identify this object with either its VID or your `merchantTransactionId`.

## *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transaction:*** the `Transaction` object that contains the added tax line items, the total amount with the total sales tax added, and the `salesTaxAddress` attribute filled in with the (corrected) address used to compute taxes.

***addressType:*** the address CashBox chose to calculate sales tax. This parameter has a value of either `Shipping` or `Billing`.

***originalAddress:*** the original value of the address chosen by CashBox for tax calculation.

***correctedAddress:*** the final value of the selected address, after CashBox has corrected inconsistencies.

***taxItems:*** an array of `SalesTax` objects, each of which contains a `description` attribute, which describes a specific type of tax added (for example, city tax); and a `tax` attribute, which contains the amount of the tax calculated by CashBox.

***totalTax:*** the total sales tax calculated by CashBox.

## *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `404` | One of the following:<br>• `Address not specified on transaction, and unable to load it from customer accounts - unable to calculate sales tax!`<br>• `Must specify line items in transaction to calculate sales tax!` |

## *Example*

```
$tx = new Transaction();
$tx->setAmount('29.90');
$tx->setCurrency('USD');
$tx->setMerchantTransactionId('txid-123456');

$tx->setSourceIp('35.45.123.158');

$account = new Account();
$account->setMerchantAccountId('9876-5432');
$account->setEmailAddress('jdoe@mail.com');
$account->setName('J Doe');
$tx->setAccount($account);

$shippingAddress = new Address();
$shippingAddress->setName('Jane Doe');
$shippingAddress->setAddr1('44 Elm St.');
$shippingAddress->setCity('San Mateo');
$shippingAddress->setDistrict('CA');
$shippingAddress->setPostalCode('94403');
$shippingAddress->setCountry('US');

$tx->setShippingAddress($shippingAddress);

// The line items of the transaction
$tx_item = new TransactionItem();
$tx_item->setSku('sku-1234');
$tx_item->setName('Widget');
$tx_item->setPrice('3.30');
$tx_item->setQuantity('3');
$tx->setTransactionItems(array($tx_item));

$paymentMethod = new PaymentMethod();
$ccCard = new CreditCard();
$ccCard->setAccount('4111111111111111');
$ccCard->setExpirationDate('201109');
$paymentMethod->setType('CreditCard');
$paymentMethod->setCreditCard($ccCard);
$paymentMethod->setBillingAddress($shippingAddress);

$tx->setSourcePaymentMethod($paymentMethod);

$response = $tx->calculateSalesTax();
if ($response['returnCode'] == 200) {
   print "Address type used for computing tax: ";
   print $response['addressType'] . "\n";
   print "Taxes added: \n";
   $taxes = $response['taxItems'];
   foreach($taxes as $tax) {
      print $tax->getDescription() . " : " ;
      print $tax->getTax() . "\n";
   }
   print "Total tax: " . $response['totalTax'];
   print "Total transaction amount: " ;
   print $response['transaction']->getAmount() . "\n";
}
```

## cancel

The `cancel` method cancels a batch of previously authorized (but not yet captured) one-time `Transaction` objects, so that CashBox does not attempt to capture them with your payment processor. See the `auth` and `capture` methods for details.

For certain payment processors, who charge a fee if you do not capture an authorized transaction, Cashbox also reverses the authorization. For other processors, CashBox simply deletes its internal to-be-captured flag so that the `Transaction` is no longer scheduled for capture. To determine whether CashBox performs authorization reversal with your payment processor as a part of this call, contact your Vindicia Client Services representative.

For the `Transaction` objects for which this call is successful, CashBox changes their status to `Cancelled`. For those transactions whose authorization CashBox was able to reverse with the payment processors concerned, the status `Void` is displayed on the CashBox Portal. However, if you fetch those transactions with a `fetch` call, the status in the corresponding `Transaction` objects is `Cancelled`.

| **Note** | You may only cancel `Transactions` that have not yet been captured. You may refund captured transactions but not cancel them. For details on refunds, see the `Refund` object. |
|---|---|

### *Input*

***transactions:*** an array of `Transaction` objects to cancel.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***qtySuccess:*** the number of successful cancellations.

***qtyFail:*** the number of failed cancellations.

***results:*** an array of `CancelResult` objects that contain information on the success or failure of the call on each transaction.

The following table lists and describes the data members for the `CancelResult` object.

Table 18-19 `CancelResult` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `merchantTrans-actionId` | `string` | Your unique identifier for the `Transaction` object you asked to cancel. |
| `returnCode` | `integer` | The reason for the success or failure:<br>• `200: cancel() succeeded.`<br>• `402: The Transaction object has expired and cannot be cancelled.`<br>• `404: cancel() cannot load the Transaction object, likely because the VID or your transaction ID (merchantTransactionId) is invalid.`<br>• `405: You did not specify an authorized transaction.` |

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Must specify transaction.`<br>• `Unable to save transactions:` ***error-description.*** |

**Note** A return code of `200` does not mean that all input transactions have been successfully cancelled. Be sure to examine the output parameters, such as ***qtySuccess***, ***qtyFail***, and ***results***, to check which transactions were successfully cancelled and which failed to cancel.

## Example

```
// create an empty transaction object to the make the SOAP calls
// against
$soap_tx = new Transaction();

$tx1 = new Transaction();
$tx2 = new Transaction();

// ids of previously authorized transactions
$merchantTxnId1 = '9876-5432';
$merchantTxnId2 = '9876-5437';

$tx1->setMerchantTransactionId($merchantTxnId1);
$tx2->setMerchantTransactionId($merchantTxnId2);

$txnArray = array($tx1, $tx2);

$response = $soap_tx->cancel($txnArray);

if($response['returnCode']==200) {
   $cancelResults = $response['results'];
   foreach ($cancelResults as $cancelResult) {
      if ($cancelResult->returnCode == 200) {
      print ("Transaction with id " .
         $cancelResult->merchantTransactionId .
         " was successfully cancelled");
      }
   }
}
```

## capture

The `capture` method schedules a batch of previously authorized transactions for the capture operation with your payment processor. For `capture` to succeed, the authorization you previously obtained from the processor through the `auth()` call must still be valid. After a `capture()` call, actual capture occurs within the next 12 hours when the Vindicia server back-end processes run the regularly scheduled batch capture operation with your payment processor.

Typically, payment processors issue authorizations for only a short duration. If a previously authorized transaction has not been captured within a certain period of time, usually a few days, CashBox sets the transaction status to `AuthExpired`; the corresponding `TransactionStatusType` enumerated value is `Cancelled`. This method will attempt to reauthorize `AuthExpired` transactions before scheduling a capture.

The business meaning of a successful capture varies according to the transaction's payment method, as follows:

- For credit card transactions, the payment processor charges the credit card specified in the `sourcePaymentMethod` data member of the `Transaction` object for the transaction amount.
- For ECP transactions, `capture()` executes the payment, that is, a fund transfer is initiated between the banks.
- For PayPal transactions, capturing a previously authorized transaction enables you to receive the customer's payment.
- For Boleto Bancário transactions, you cannot call `capture()`. Instead, authorize and capture transactions in the single call `authCapture()`. (See the `authCapture` method.)

### *Input*

***transactions:*** an array of `Transaction` objects to schedule for capture with the payment processor.

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***qtySuccess:*** the number of transactions that can be successfully scheduled for capture.

***qtyFail:*** the number of transactions that cannot be scheduled for capture.

***results:*** an array of `CaptureResult` objects that contain information on the success or failure of the call on each transaction.

The following table lists the `CaptureResult` object data members.

Table 18-20 `CaptureResult` Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| `merchantTransactionId` | string | Your unique identifier for this `Transaction` object. Although you normally assign this value, Vindicia might assign it for the transactions it generates for re-authorization. |
| `originalMerchantTransactionId` | string | Your unique identifier for the original `Transaction` object in the case of a reauthorization. |
| `returnCode` | integer | The reason for the success or failure:<br>• `200: capture() succeeded.`<br>• `402: The Transaction object has expired and cannot be reauthorized by capture().`<br>• `404: capture() cannot load the Transaction object, likely because the VID or your transaction ID (merchantTransactionId) is invalid.`<br>• `405: You did not specify an authorized transaction.`<br>• `500: capture() encountered an internal failure.` |

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Must specify transaction.`<br>• `Unable to save transactions:` ***error-description.*** |

**Note** A return code of `200` does not mean that all input transactions have been successfully captured. Be sure to verify the number of successfully captured transactions in the ***qtySuccess*** output parameter against the number of input transactions. If some transactions have failed to be captured, examine the return codes in the ***results*** output parameter for possible explanation.

### *Example*

```
// to capture a batch of previously authorized transactions

// create an empty transaction object to the make the SOAP calls
// against
$soap_tx = new Transaction();

$tx1 = new Transaction();
$tx2 = new Transaction();

// ids of previously authorized transactions
$merchantTxnId1 = '9876-5432';
$merchantTxnId2 = '9876-5437';

$tx1->setMerchantTransactionId($merchantTxnId1);
$tx2->setMerchantTransactionId($merchantTxnId2);

$txnArray = array($tx1, $tx2);

$response = $soap_tx->capture($txnArray);

if($response['returnCode']==200) {
   $captureResults = $response['results'];
   foreach ($captureResults as $captureResult) {
      if ($captureResult->returnCode == 200) {
         print ("Transaction with id " .
              $captureResult->merchantTransactionId .
                   " was successfully captured");
      }
   }
}
```

## fetchByAccount

The `fetchByAccount` method returns one or more `Transaction` objects associated with the `Account` object specified in the input. Call this method to retrieve one-time, recurring, migrated, or other types of transactions in CashBox for a given customer.

Since transactions change their status as they go through their life cycle in CashBox, the returned `Transaction` objects might show a different status from before, especially for CashBox-processed transactions. The latest `Transaction` status is the first entry in the `statusLog` array (see the `statusLog` attribute in the table on the `Transaction` object data members).

### *Input*

***account:*** the `Account` object that serves as the search criterion. Use the `merchantAccountId` or `VID` to identify the object.

***includeChildren:*** an optional Boolean flag that, if set to `true`, includes any children associated with this `Account`. If this flag is omitted, CashBox will interpret it as `false`, and constructs the query without looking at any child's account.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transactions:*** an array of one or more `Transaction` objects associated with the `Account` object specified in the input.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load account to search by: No matches.`<br>• `No account specified to load transaction by!` |
| `404` | `Unable to load account to search by:` ***error-description.*** |

## *Example*

```
// Create an Account object to represent an
// existing customer account by its id
$account = new Account();
$account->setMerchantAccountId('jdoe101');

// create a transaction object to make the call
$soap_tx= new Transaction();

// fetch the record(s)
$response = $tx->fetchByAccount($account);
if($response['returnCode'] == 200) {
   $fetchedTxns = $response['data']->transactions;

   // process fetched transactions here
   if ($fetchedTxns != null) {
      foreach ($fetchedTxns as $fetchedTx) {
         // process a fetched transaction here
         print "Transaction VID " . $fetchedTx->getVID();
         print "Transaction amount ". $fetchedTx->getAmount();
         print "Transaction status ";
         print $fetchedTx->statusLog[0]->status . "\n";
      }
   }
   else {
      print "No transactions found \n";
   }
}
```

## fetchByAutobill

The `fetchByAutobill` method, which returns all the `Transaction` objects generated by CashBox for an `AutoBill` object, enables you to retrieve the rebilling transactions related to a specific `AutoBill`. Because Transactions are automatically generated and completed by CashBox, they are usually not in your system. Occasionally, you might need to access them in order to respond to customer queries.

### *Input*

***autobill:*** the `AutoBill` object that serves as the search criterion. You can identify this object with either its VID or your AutoBill ID (`merchantAutoBillId`).

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transactions:*** an array of one or more `Transaction` objects whose `AutoBill` object matches the input.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load autobill to search by: No matches.`<br>• `No autobill specified to load transaction by!` |
| `404` | `Unable to load autobill to search by:` ***error-description.*** |

### *Example*

```
// Create an AutoBill object to represent an
// existing customer subscription by its id
$autobill = new AutoBill();
$autobill->setMerchantAutoBillId('AB101');

// create a transaction object to make the call
$soap_tx= new Transaction();

// fetch the record(s)
$response = $tx->fetchByAutobill($autobill);
if($response['returnCode'] == 200) {
    $fetchedTxns = $response['data']->transactions;

    // process fetched transactions here
    if ($fetchedTxns != null) {
        foreach ($fetchedTxns as $fetchedTx) {
            // process a fetched transaction here
            print "Transaction VID " . $fetchedTx->getVID();
            print "Transaction amount ". $fetchedTx->getAmount();
            print "Transaction status ";
            print $fetchedTx->statusLog[0]->status . "\n";
        }
    }
    else {
        print "No transactions found \n";
    }
}
```

## fetchByMerchantTransactionId

The `fetchByMerchantTransactionId` method returns a `Transaction` object whose `merchantTransactionId` value matches the input. This ID could be assigned by you (for example, when you conduct a one-time transaction) or by CashBox while generating a rebilling transaction for an active `AutoBill` object.

Because Transactions change their status as they go through their life cycle in CashBox, returned `Transaction` objects might show a different status each time they are returned, especially for CashBox-processed transactions. The latest Transaction status is the first entry in the `statusLog` array (see the `statusLog` attribute in Section 18.1: Transaction Data Members. For example, if you create a one-time transaction and call `authCapture()` on it, the latest transaction status is `Authorized`. Later, if you retrieve the same `Transaction` by its ID with this method, the latest status could be `Captured`.

### *Input*

***merchantTransactionId:*** the `merchantTransactionId` value, which serves as the search criterion.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transaction:*** the `Transaction` object whose `merchantTransactionId` value matches the input.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `404` | One of the following:<br>• `Unable to load transaction: No match for merchantTransactionId` ***input-merchantTransactionId.***<br>• `Unable to load transaction by merchantTransactionId` ***input-merchantTransactionId: error-description.*** |

### *Example*

```
// Known transaction id
$txId = "MERCH42202";

// create a transaction object to make the call
$soap_tx= new Transaction();

// fetch the record(s)
$response = $tx->fetchByMerchantTransactionId($txId);
if($response['returnCode'] == 200) {
   $fetchedTx = $response['data']->transaction;

   // process fetched transactions here
   if ($fetchedTx != null) {
      // process a fetched transaction here
      print "Transaction VID " . $fetchedTx->getVID();
      print "Transaction amount ". $fetchedTx->getAmount();
      print "Transaction status ";
      print $fetchedTx->statusLog[0]->status . "\n";
   }
}
else if($response['returnCode'] == 404) {
   print "No transaction found: ";
   print $response['returnString'] . "\n";
}
```

## fetchByPaymentMethod

The `fetchByPaymentMethod` returns all `Transaction` objects that use the specified payment method. For example, call this method to search for all `Transactions` that use a certain credit-card number.

This method supports paging to limit the number of records returned per call. Returning a large number of records in one call may swamp buffers, and might cause a failure. Vindicia recommends that you call this method in a loop, incrementing the page for each loop iteration with an optimal page size (number of records returned in one call) until the page contains a number of records that is less than the given page size.

### *Input*

***paymentMethod:*** the `Transaction` object's payment method, which serves as the search criterion. Identify the payment method with its VID, your payment method ID (`merchantPaymentMethodId`), or one of the following:

- The account number for a credit card. Be certain to set the `type` attribute of the input `PaymentMethod` object to `CreditCard`. This call does **not** support wildcards in the account number.
- The account number-bank routing number combination for ACH and ECP. Be certain to set the `type` attribute of the input `PaymentMethod` object to ECP.
- The fiscal number for a Boleto. Be certain to set the `type` attribute of the input `PaymentMethod` object to `Boleto`.
- The `PaypalEmail` for PayPal.

**Note:** If you use SOAP releases prior to 3.5, you will not be able to search accounts using the PayPal payment method. SOAP release 3.6.0 and later allows you to search accounts and transactions by the `PaypalEmail`.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transactions:*** an array of one or more `Transaction` objects that were conducted with the payment method specified in the input.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• Payment method type is credit card, but credit card information is incomplete.<br>• Payment method type is ECP, but ECP account and routing information is incomplete.<br>• Payment method type is Boleto, but Boleto payment information is incomplete.<br>• Payment method type is currently not supported.<br>• Must specify a PaymentMethod object, a non-negative page number, and a page size greater than 0. |
| 404 | No matching transactions. |

## Example

```
$pm = new PaymentMethod();
$pm->setType('CreditCard');
$cc = new CreditCard();

// this is the card number we want to search by
$cc->setAccount('4111111111111111');
$cc->setExpirationDate('201208');
$pm->setCreditCard($cc);

$soap_tx = new Transaction();
$page = 0;
$pageSize = 10; // max 10 records per page

do {
    $response = $soap_tx->fetchByPaymentMethod($pm,
        $page, $pageSize);

    if($response['returnCode']==200) {
        $txns = $response['data']->transactions;
        if ($txns != null) {
            $count = count($txns);
            foreach ($txns as $fetchedTx) {
                 // process each transaction found here
                 print "Found transaction with id: ";
                 print $fetchedTx->getMerchantTransactionId() . "\n";
            }
        }
        else {
            $count = 0;
        }
    }
    else {
        $count = 0;
    }
    $page++
} while ($count > 0);
```

## fetchByVid

The `fetchByVid` method returns a `Transaction` object whose VID matches the input.

VID is Vindicia's unique identifier for an object. While saving a `Transaction` object in its database for the first time after you've made a call (such as `migrate()`, `auth()`, or `authCapture()`), CashBox generates and assigns a unique identifier for the object. Some calls return the newly created and updated `Transaction` object to you in their output response with the VID populated in the output `Transaction` object. Once you know a `Transaction` object's VID, you may refer to that object by its VID in future calls.

Never assign a VID to a new `Transaction` object; CashBox will generate the VID.

### Input

***vid:*** the `Transaction` object's Vindicia unique identifier, which serves as the search criterion.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transaction:*** the `Transaction` object whose VID matches the input.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | No VID specified to load transaction by. |
| 404 | One of the following:<br>• Unable to load transaction: No match for VID ***input-vid.***<br>• Unable to load transaction by VID ***input-vid: error-description.*** |

### Example

```
// Known VID
$vid = "29ed2ea9753896f980095911972d6695b049f54c";

// create a transaction object to make the call
$soap_tx= new Transaction();

// fetch the record(s)
$response = $tx->fetchByVid($vid);
if($response['returnCode'] == 200) {
   $fetchedTx = $response['data']->transaction;

   // process fetched transactions here
   if ($fetchedTx != null) {
      // process a fetched transaction here
      print "Transaction VID " . $fetchedTx->getVID();
      print "Transaction amount ". $fetchedTx->getAmount();
      print "Transaction status ";
      print $fetchedTx->statusLog[0]->status . "\n";
   }
}
else if($response['returnCode'] == 404) {
   print "No transaction found: ";
   print $response['returnString'] . "\n";
}
```

## fetchByWebSessionVid

Call the `fetchByWebSessionVid` method within your HOA implementation to retrieve the `Transaction` object created by HOA on Vindicia's servers when a customer submits an order form, which results in a one-time or recurring bill. You must create a `WebSession` object on Vindicia's servers before serving the form to your customer to track the form's submission to Vindicia. For more information, see Section 19: The WebSession Object.

The `WebSession` object's VID serves as the tracking ID for various activities, starting from serving the order form to a customer, and ending in returning a success or failure page to that same customer.

Use `fetchByWebSessionVid` to program the success page (see the `WebSession` object's `returnURL` attribute), to which HOA redirects the customer's browser after successfully processing the data in the order form. The `WebSession` object's VID is available to you on the success page, because HOA passes it during the redirection. Pass that VID as the input parameter to this call, and retrieve the `Transaction` object created by HOA. Then, extract the contents of the `Transaction` object and include them, as appropriate, in the success page to be returned to the customer.

### *Input*

***vid:*** the `WebSession` object's Vindicia unique identifier for tracking the submission of the order form.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transaction:*** a `Transaction` object that was created by HOA as a result of an order form submitted by a customer.

### *Returns*

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | Missing required parameter 'vid'. |
| 404 | Unable to find requested Transaction: No matches. |

## *Example*

```
// to use the fetchByWebSessionVid call on a success web page

$webSessionVid = …; //passed in by redirected page
$soap = new WebSession($soapLogin, $soapPwd);
$response = $soap->fetchByVID($webSessionVid);

if ($response['returnCode'] == 200) {

    $fetchedWs = $response['data']->session;

    // check if the CashBox API call made by HOA was successful
    $retCode = $fetchedWs->apiReturn->returnCode;
    if ($retCode == 200) {

        // Assuming HOA created a Transaction object, let's
        // fetch it
        $soapTxn = new Transaction($soapLogin, $soapPwd);
        $resp = $soapTxn->fetchByWebSessionVid($webSessionVid);

        if ($resp['returnCode'] == 200) {
            $createdTxn = $resp['data']->transaction;

            // Get Transaction contents here to be included in
            // HTML returned to the customer.
        }
        else {
            // Return error message to customer
        }
    }
    else {
        // return failure page to customer
    }
}
else {
    // Return error message to the customer
}
```

## fetchDelta

The behavior of the `fetchDelta()` call is similar to that of `fetchDeltaSince`, except that you need not specify a timestamp as a parameter. CashBox tracks your calls to this method, and returns the `Transaction` objects whose status has changed since your last call. If you have never called this method, CashBox returns all `Transactions` created since January 1, 1970 (“epoch”).

For paging, specify the page size only for this method. Like `fetchDeltaSince`, there is no need to increment through page numbers, because this call keeps a record of the last item returned to you in the previous call. Each time you make this call, the results will continue from the last position in the result set.

### Input

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transactions:*** an array of one or more `Transaction` objects whose status has changed since this method was last called.

***startDate:*** the starting timestamp for the range of `Transaction` objects fetched.

***endDate:*** the ending timestamp for the range of `Transaction` objects fetched.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
$soap_tx = new Transaction();
$pageSize = 50;

do {
    $ret = $soap_tx->fetchDelta ($pageSize);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedTxns = $ret['transactions'];
        if ($fetchedTxns != null) {
            $count = sizeof($fetchedTxns);
            foreach ($fetchedTxns as $txn) {

                 // process a fetched transaction here …
                 $status = $txn->statusLog[0]->status;
                 $transactionId = $txn->getMerchantTransactionId();
                 $amount = $txn->getAmount();
            }
            $page++;
        }
    }
} while ($count > 0);

// quit when no more objects are retrieved
```

## fetchDeltaSince

The `fetchDeltaSince` method returns one or more `Transaction` objects whose status has changed since the specified timestamp. Call this method to programmatically and periodically download Transactions from Vindicia for reconciliation with the payments deposited into your bank account by your payment processor, especially if you use CashBox for recurring billing only. In that case, because CashBox generates and processes all your transactions with your payment processor, you (may) have no records of them. For record-keeping, reporting, or any other purpose, periodically synchronize your database with the Transactions in the Vindicia database by calling this method.

Vindicia recommends that you call this method at regular intervals, and make note of the date and time, so that you can specify that as the timestamp for your next call. The appropriate interval for the calls depends on your transaction volume. If your volume is large, call this method more often to limit the amount of data you receive. You may also further filter and limit the number of transactions returned by specifying a payment method as another search criterion.

The `fetchDeltaSince` method supports paging to limit the number of records returned per call. Returning a large number of records in one call may swamp buffers and might cause a failure. Vindicia recommends that you call this method in a loop, incrementing the page for each loop iteration with an optimal page size (number of records returned in one call) until the page contains a number of records that is less than the given page size.

You may also download transaction-related reports from the CashBox Portal. See the *CashBox User's Guide* for details.

### *Input*

***timestamp:*** a timestamp that specifies the date and time on or after which the `Transaction` objects have changed status.

***endTimestamp:*** a timestamp that specifies the upper limit of the date and time before which the `Transaction` objects have changed status.

***page:*** the page number, starting at 0, for which to return the results. For example, if the total number of results is 85 and ***pageSize*** is 10:

- Specifying 0 for ***page*** gets the results from 1 through 10.
- Specifying 2 for ***page*** gets the results from 21 through 30.

***pageSize:*** the number of records to display per page per call. This value must be greater than 0.

***paymentMethod:*** a `PaymentMethod` object, an optional constraint that, if specified, restricts retrieval to only those `Transaction` objects whose source payment method matches the input. Identify the `PaymentMethod` with its VID or your payment method ID (`merchantPaymentMethodId`).

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transactions:*** an array of one or more `Transaction` objects whose status has changed since the specified timestamp but before ***endTimestamp***, if specified, and that use ***paymentMethod***, if specified.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• Invalid Arguments - Must specify a valid payment method type, if using that option.<br>• Must specify a timestamp to find transactions newer than … |
| 404 | Not Found - No match found for the Payment Method. |

## Example

```
$soap_tx = new Transaction();
$page = 0;
$pageSize = 50;

// Fetch transactions that have changed in status since the last time
// this call was run. Assume we have a function available to us that
// gives us the timestamp when the last time we ran this call.

$since = getLastCallTime();
do {
    // we will not filter returned transactions by end timestamp
    // and payment method
    $ret = $soap_tx->fetchDeltaSince($since, null, $page,
        $pageSize, null);
    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedTxns = $ret['transactions'];
        if ($fetchedTxns != null) {
            $count = sizeof($fetchedTxns);
            foreach ($fetchedTxns as $fetchedTx) {

                // process a fetched transaction here …
                $status = $fetchedTx->statusLog[0]->status;
                $transactionId =
                     $fetchedTx->getMerchantTransactionId();
                $amount = $fetchedTx->getAmount();
            }
            $page++;
        }
    }
} while ($count > 0);
```

## finalizeCustomerAction

The `finalizeCustomerAction` method completes the authorization of a Hosted Page payment method validation transaction. Use this method **only** when working with a `Transaction` that is paid for with this payment method.

> **Note:** The customer's Account must exist before calling `finalizeCustomerAction`.

### Input

***transactionVid:*** Vindicia's ID for the `Transaction` generated for a HostedPage payment method. This will be available to you through the URL when your customer is redirected to your site by the payment provider.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transaction:*** the resultant `Transaction` object after finalization. It contains the updated `status` of the transaction.

### Returns

This method returns the codes listed in Table 1: Standard Return Codes.

### Example

```
// Create a new Transaction with payment_product = 702
// The Transaction will be created with status: "New."
$tx = new_transaction($identifier, "702");

// Call authCapture on this transaction.
// Note: There is no support for the auth call.
// By definition, an auth request for Payment Methods
// aggregated through Hosted Pages will result in
// the transaction being captured.
// The Transaction status will be changed to
// "PendingCustomerAction" until your customer completes
// the payment on the hosted pages.

$rc = $trans->authCapture($trans, 0, 1, 1);

// Set the status of the Transaction to "AuthorizedPending"
// in case of success.
$rc = $trans->finalizeCustomerAction($VID);
```

## finalizePayPalAuth

The `finalizePayPalAuth` method completes the authorization of a PayPal payment method validation transaction. Use this method **only** when working with a `Transaction` that is paid for with a PayPal-based payment method. The `authCapture()` call made to conduct a one-time transaction returns a PayPal site URL. Ask your customer to visit that URL so that they may complete the authorization process necessary to validate the payment method at PayPal's site.

After the customer finishes the authorization sequence at the PayPal website, and is redirected to your site by PayPal, call the `finalizePayPalAuth` method from either the success page (`returnUrl` specified in the PayPal payment method) or the failure page (`cancelUrl` specified in the payment method) to which the customer was redirected. This method enables you to tell CashBox the status of the `Transaction`, so that CashBox can move it out of its `AuthorizationPending` status. If authorized, CashBox sets the status of the transaction to `Authorized`, and then schedules it for capture.

For more information on applying tax to PayPal transactions, please see The Transaction Object's addressAndSalesTaxFromPayPalOrder method.

**Note:** Billing Success emails will not be issued for the Transaction until this call is made.

### *Input*

***payPalTransactionId:*** Vindicia's ID for the PayPal payment method validation `Transaction`, generated when you called `Transaction.capture`. Retrieve this ID from the value associated with the name: `vindicia_vid` in the name–value pairs attached to the redirect URL.

***success:*** set to `true` if the customer successfully authorized the validation transaction at PayPal's site and was redirected to the success page (`returnUrl`) hosted by you. If the customer was redirected to the failure page (`cancelUrl`), set this to `false`.

### *Output*

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transaction:*** the resultant `Transaction` object after finalization. It contains the updated `status` of the transaction.

### *Returns*

This method returns the codes listed in Table 1: Standard Return Codes.

## *Example*

```
$soap_caller = new Transaction();

// obtain the id of the PayPal transaction from the redirect URL.
// It is the value associated with name 'vindicia_vid'

$payPalTxId = … ;

// if calling from return URL which is reached when the PayPal
// transaction is successfully authorized, set the
// success input parameter to true, from the cancelUrl,
// set it to false. Let's assume success here:

$success = true;
$response =
   $soap_caller->finalizePayPalAuth($payPalTxId, $success);

if($response['returnCode'] == 200) {
       $txId = $response['transaction']->getMerchantTransactionId();
       printLog "Transaction authorized: " . $txId;
}
```

## migrate

The `migrate` method allows you to enter Transactions, processed outside CashBox, to the CashBox database. `Transactions` imported to CashBox using this method are stored in the database. Those that are entered with a status of failed will be processed by CashBox according to your defined retry schedule.

Transactions entered using this method may be searched and analyzed, both through the CashBox UI, and using the `Transaction.fetchDeltaSince` method.

After migration, these Transactions will be processed and treated as if they originated with CashBox, allowing you to use this method to import historic billing information for your customers.

When you call this method to import a batch of Transactions, Vindicia queues the data, and then processes it in the order received, before adding it to the database. Lag time exists between the time you migrate a transaction, and the time it appears in the CashBox database and UI. The lag varies according to your transaction volume, and that of other merchants currently in the queue.

Vindicia recommends small batches for this call. If your migrated Transaction volume is high, call `Transaction.migrate` more often to reduce the amount of data sent in one call. (The optimal batch size depends on the total amount of data being sent.) To minimize timeouts, consider adjusting the timeout setting in the client library and the batch size for the call.

### Input

***migrationTransactions:*** an array of `migrationTransaction` objects to import to CashBox.

---

**Note:** While this method uses the same `migrationTransaction` subobject as the `AutoBill.migrate` method, the two methods require that different data members be populated.

Do not populate the following `migrationTransaction` data members for the `Transaction.migrate` call:

- `autoBillCycle`
- `billingPlanCycle`
- `merchantBillingPlanId`

Do not populate the following `migrationTransactionItem` data member for the `Transaction.migrate` call:

- `merchantAutoBillItemId`

---

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***response:*** an array of `TransactionValidationResponse` objects.

### Returns

When you migrate a batch of Transactions, a return code of `200` means that CashBox has received your data and queued it for processing. During this process, if CashBox discovers problems with the data that prevent it from being added to the CashBox database, CashBox

attempts to correct the data. If the attempt fails, CashBox will ask you to correct the errors and might request that you report the data again.

The `Return` object also contains an attribute called `soapId`. For the `migrate` call to succeed, you must log the value of `soapId`. If, for some reason, the migrated `Transactions` do not make it into the CashBox database, provide the `soapId` value to CashBox to facilitate tracking of your batch in the CashBox system.

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to save transactions:` ***error-description***.<br>  • *(This code is returned if an error occurs in the processing of a transaction and it is the only transaction in the batch.)*<br>• `One or more Transaction migrations failed.`<br>• `Error descriptions provided in TransactionValidationResponse (contained in the response array of the return).`<br>  • `Invalid field(s) for non-recurring Transaction Migration: (`***invalid fields***`)`<br>  Invalid `MigrationTransaction` fields (when calling `Transaction.migrate`): `autoBillCycle`, `merchantBillingPlanId`, `billingPlanCycle`, `billingDate`, `retryNumber`.<br>  Invalid `MigrationTransactionItem` fields (when calling `Transaction.migrate`): `servicePeriodStartDate`, `servicePeriodEndDate`, `merchantAutoBillItemId`.<br>• `Unable to prepare transaction for migration:` ***error.***<br>  • *(Details provided in common `AutoBill.migrate`/`Transaction.migrate` messages.)* |
| `400` | One of the following:<br>• `MigrationTransaction not provided.`<br>• `Invalid paymentProcessor:` ***paymentProcessor***.<br>• `MigrationTransaction must include at least one statusLog record.`<br>• `Failed to convert salesTaxAddress.`<br>• `Attempt to migrate Transaction which already exists.`<br>• `Unsupported Payment Type:` ***paymentType***.<br>• `Failed to prepare auth_response for Migrated Transactions.`<br>• `Unable to determine currency for migrated Transaction.`<br>• `Calculated Transaction amount (`***XXX.XX***`) does not match input amount (`***YYY.YY***`) on migrated Transaction.` |

## *Example*

```
//To migrate a Transaction that has been processed via an external system

//Create the customer account objects
my $address = new Address();
$address->setAddr1('11235 Fibonacci St.');
$address->setCity('San Mateo');
$address->setCountry('US');
$address->setDistrict('CA');
$address->setName('Forest Chump');
$address->setPhone('(650) 555-1212x42');
$address->setPostalCode('94403');

my $creditCard = new CreditCard();
$creditCard->setAccount('4222261111112664');
$creditCard->setBin('22226');
$creditCard->setAccountLength(16);
$creditCard->setExpirationDate('201602');
$creditCard->setLastDigits

my $paymentMethod = new PaymentMethod();
$paymentMethod->setAccountHolderName('Forest Chump');
$paymentMethod->setActive(1);
$paymentMethod->setBillingAddress($address);
$paymentMethod->setCreditCard($creditCard);
$paymentMethod->setCustomerSpecifiedType('VI');
$paymentMethod->setMerchantPaymentMethodId('vi_1391721679');
$paymentMethod->setSortOrder(0);
$paymentMethod->setType('CreditCard');

my $account = new Account();
$account->setEmailAddress('devnull@devnull.com');
$account->setEmailTypePreference('html');
$account->setMerchantAccountId('maccid_1391721679');
$account->setName('Forest Chump');
$account->setPaymentMethods(array($paymentMethod));
$account->setShippingAddress($address);

//Create the Transaction objects
$taxItemA = new MigrationTaxItem();
$taxItemA->setAmount(.38);
$taxItemA->setJurisdiction('COUNTY_19');
$taxItemA->setName('SALES TAX');

$taxItemB = new MigrationTaxItem();
$taxItemB->setAmount(2.75);
$taxItemB->setJurisdiction('DISTRICT');
$taxItemB->setName('CA DISTRICT SALES TAX');

$txItem = new MigrationTransactionItem();
$txItem->setItemType('NonRecurringCharge');
$txItem->setMigrationTaxItems(array($taxItemA, $taxItemB));
$txItem->setName('PetrifiedVomitOnAStick');
$txItem->setPrice(49.99);
$txItem->setSku('CB-4081');
$txItem->setTaxClassification('DC010500');
// This should be the Avalara tax code associated with this product
```

```
$creditCardStatusA = new CreditCardStatus();
$creditCardStatusA->setAuthCode('000');
$statusLogA = new TransactionStatus();
$statusLogA->setCreditCardStatus($creditCardStatusA);
$statusLogA->setPaymentMethodType('CreditCard');
$statusLogA->setStatus('Captured');
$statusLogA->setTimestamp('2014-02-06T13:22:16-08:00');

$creditCardStatusB = new CreditCardStatus();
$creditCardStatusB->setAutCode('000');
$statusLogB = new TransactionStatus();
$statusLogB->setCreditCardStatus($creditCardStatusB);
$statusLogB->setPaymentMethodType('CreditCard');
$statusLogB->setStatus('Authorized');
$statusLogB->setTimestamp('2014-02-06T13:21:33-08:00');

$statusLogC = new TransactionStatus();
$statusLogC->setPaymentMethodType('CreditCard');
$statusLogC->setStatus('New');
$statusLogC->setTimestamp('2014-02-06T13:21:23-08:00');

$migrationTransaction = new MigrationTransaction();
$migrationTransaction->setAccount($account);
$migrationTransaction->setAmount(41.08);
$migrationTransaction->setCurrency('USD');
$migrationTransaction->setDivisionNumber('iAmTheWalrus');
$migrationTransaction->setMerchantAffiliateId('Joe');
$migrationTransaction->setMerchantAffiliateSubId('Bob');
$migrationTransaction->setMerchantTransactionId('mTXID-1391721679-1');
$migrationTransaction->setMigrationTransactionItems(array($txItem));
$migrationTransaction->setPaymentMethod($paymentMethod);
$migrationTransaction->setPaymentProcessor('Litle');
$migrationTransaction->setPaymentProcessorTransactionId('1069127');
$migrationTransaction->setSalesTaxAddress($address);
$migrationTransaction->setShippingAddress($address);
$migrationTransaction->setSourceIp('63.201.132.182');
$migrationTransaction->setStatusLog(array($statusLogA, $statusLogB,
$statusLogC));
$migrationTransaction->setType('NonRecurring');

//Migrate Transaction into CashBox
$response = $transaction->migrate(array($migrationTransaction));
if($response['returnCode'] == 200)
{
   //Transaction(s) migrated successfully
}
else
{
   //One or more Transaction migrations failed.
   //Rummage through the TransactionValidationResponse objects
   //in the $response to determine the source of the problem(s)
}
```

## score

The `score` method evaluates the chargeback probability score (also called risk score) for the `Transaction` object specified in the input, and stores the object in the Vindicia database.

Scoring a transaction before accepting it is a recommended best practice in the payment industry. It helps keep your costs low by:

- Avoiding payment processor fees for authorization calls to the processor for transactions which your processor will not approve.
- Keeping your chargeback rate low. Processing and disputing chargebacks can be expensive. Payment processors typically require that you keep a very low chargeback rate.

The risk score is most applicable if the transaction's payment method is credit card.

This call evaluates the risk score by examining several elements, including:

- The IP address of the origin of the transaction:
  - Whether the transaction originated from a proxy IP address known to Vindicia as an originator of fraudulent, malicious transactions.
  - How the geolocation of the IP address compares with the transaction's billing address.
- The billing and shipping addresses:
  - Whether a transaction's billing address or shipping address (or both) is known for being a fraudulent mail drop.
  - Whether the country of the address is a country known for the origin of fraudulent transactions.
- The BIN (the first six digits the credit-card number), which provides information on the bank that issued the credit card: whether the country of the billing address matches that of the issuing bank.
- The customer's email address: whether it is from a free email provider, and if the email address has been associated with high–risk or fraudulent transactions.
- The credit-card account: whether the Vindicia database shows a previous chargeback against the transaction or the credit card used to pay for it. If so, `score()` returns the highest score of 100.

> **Note:** The `score` method initiates the CashBox risk-screening service. Be certain to subscribe to that service before calling `score`.

Call `score()` in these circumstances:

- If you subscribe to ChargeGuard only, that is, if you process your transactions outside of Vindicia and need to report them to Vindicia for chargeback dispute only, call this method to screen a transaction for fraud risk before processing it, and to simultaneously record it in the Vindicia database, saving you a separate reporting step.
- If you process one-time transactions through CashBox, call this method to screen a transaction before processing it with your payment processor.

This call requires that your transaction contain at least the following information:

- Source IP address
- Billing address:
  - City
  - District (state or province). If states or provinces do not exist in the country in question, fill in the field with `None`.
  - Country

A risk score of `100` indicates that Vindicia is certain that the transaction is fraudulent and will result in a chargeback; a risk score of 0 means that the transaction is sound with a minimal likelihood of chargeback. You must decide the score level that you can tolerate. If you pick a high threshold, you might end up accepting many fraudulent transactions that will result in chargebacks. On the other hand, a low threshold might cause you to reject potentially good transactions and lose revenue. Selecting the right threshold for your risk score takes a bit of work. We recommend that you watch the scores on both the legitimate and fraudulent transactions before setting the threshold.

You can also indirectly screen transactions for risk by calling the `Transaction` object's `auth()` method or the `AutoBill` object's `update()` method. See the ***minChargebackProbability*** parameter supported by these methods.

In addition to returning the risk score, the `score()` method also returns descriptive strings that explain the score. Those strings have associated codes (IDs) called `ScoreCode` objects, listed in Table 18-21. Use these score codes to trigger certain actions in your application, such as in customer messaging, especially if you are rejecting a transaction because of a high risk score.

Table 18-21 Score Code Descriptions

| Score Code (ID) | Description |
|---|---|
| `14` | The city and state in the shipping address do not match the ZIP code. |
| `15` | The city and state in the billing address do not match the ZIP code. |
| `16` | The shipping address is in the database of known risky mail drops. |
| `21` | The country of the issuing bank does not match the country of the billing address. |
| `31` | The password is in the database of high-risk passwords. |
| `32` | The user name is in the database of high-risk user names. |
| `41` | The email address is in the database of high-risk email addresses. |
| `42` | The email address is from a free email provider. |
| `51` | The IP address is in the database of known transparent proxy servers. |
| `52` | The IP address is an anonymous proxy. |
| `63` | The country of the IP address or billing address is a high-risk country. |

Table 18-21 Score Code Descriptions *(Continued)*

| Score Code (ID) | Description |
|---|---|
| 64 | The distance between the IP address and billing address is *XX* kilometers. |
| 65 | The IP address and billing address are in different countries. |
| 71 | The `Account` object is associated with known fraudulent (friendly or true-fraud) chargebacks. |

## Input

***transaction:*** the `Transaction` object to score.

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***transaction:*** a copy of the specified `Transaction` object, identified with a VID if not included in the input.

***score:*** the `Transaction` object's fraud risk score, which represents the estimated probability that this transaction will result in a chargeback. This number ranges from 0 (best) to 100 (worst). It can also be -1, meaning that Vindicia has no opinion. In particular, -1 applies to transactions with no originating IP addresses, incomplete addresses, or both. A score of -2 indicates an error; retry later.

If the score is not acceptable, you might want to contact the customer for more information, and then call this method again for another score.

***scoreCodes:*** an array of `ScoreCode` objects that explain the score. Each object contains two attributes: `id` and `description`. See Table 18-21: Score Code Descriptions for details.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | One of the following:<br>• `Unable to save transactions:` ***error-description.***<br>• `Data validation error:` ***error-description.*** |

## Example

```
$tx = new Transaction();
$tx->setAmount('29.90');
$tx->setCurrency('USD');
$tx->setMerchantTransactionId('txid-123456');

// IP is one of required attributes for scoring a transaction
$tx->setSourceIp('35.45.123.158');

$account = new Account();
$account->setMerchantAccountId('9876-5432');
$account->setEmailAddress('jdoe@mail.com');
$account->setName('J Doe');
$tx->setAccount($account);

$shippingAddress = new Address();
```

```
$shippingAddress->setName('Jane Doe');
$shippingAddress->setAddr1('44 Elm St.');
$shippingAddress->setCity('San Mateo');
$shippingAddress->setDistrict('CA');
$shippingAddress->setPostalCode('94403');
$shippingAddress->setCountry('US');

$tx->setShippingAddress($shippingAddress);

// The line items of the transaction
$tx_item = new TransactionItem();
$tx_item->setSku('sku-1234');
$tx_item->setName('Widget');
$tx_item->setPrice('3.30');
$tx_item->setQuantity('3');
$tx->setTransactionItems(array($tx_item));

$paymentMethod = new PaymentMethod();
$ccCard = new CreditCard();
$ccCard->setAccount('4111111111111111');
$ccCard->setExpirationDate('201109');
$paymentMethod->setType('CreditCard');
$paymentMethod->setCreditCard($ccCard);

// Billing address city, district, country are required for score
// call to work
$paymentMethod->setBillingAddress($shippingAddress);

$tx->setSourcePaymentMethod($paymentMethod);

$response = $tx->score();

if ($response['returnCode'] == 200) {
    if($response['score']->score <= 50) {
        print "Acceptable score, processing transaction";
        // process the transaction further here
    }
    else {
        print "High risk of chargeback. Reasons are: \n";
        $scoreCodes = $response['scoreCodes'];
        foreach ($scoreCodes as $scoreCode) {
            print("Score code ". $scoreCode['id'] . " : " .
                      $scoreCode['description'] . "\n");
        }
    }
}
else {
    // the score call did not succeed, check return code
    // and return string and try to re-submit
}
```

# 19 The `WebSession` Object

Create `WebSession` objects, in the context of Vindicia's HOA function, in anticipation of the submission of the Web order form by a customer who requested the form from your server. While filling out the form, the customer enters sensitive payment data, such as a credit-card numbers, before submitting the form to HOA, which is hosted on Vindicia's server. Handling such data might mean that you must comply with PCI requirements. With HOA, however, your billing infrastructure need not handle any payment data at all. See Chapter 13: Hosted Order Automation in the ***CashBox Programming Guide***, for details.

Note that the `WebSession` object is only partly populated at creation. It might, for example, contain private data that you do not want to be visible in the form that you serve to the customer, but that is needed for the API call made by HOA at form submission. One key piece of data you must include in the `WebSession` object is the CashBox API call (see the `method` attribute) HOA should make when the customer submits the form. Once created, the `WebSession` object contains a VID. Embed that VID in the form you serve to the customer so that HOA can match the form's submission with the corresponding `WebSession` object instance.

After form submission by the customer, HOA makes the API call you specified in the `WebSession` object's `method` attribute to create an object that requires sensitive payment information, such as an `AutoBill`, a `PaymentMethod`, or a `Transaction`. Fetch the `WebSession` object by calling its `fetchByVid()` method, typically before returning the success or failure page to the customer: HOA redirects the customer's browser to one of those pages after receiving the form. See Chapter 13: Hosted Order Automation in the ***CashBox Programming Guide*** for details on the role of the `WebSession` object in the HOA process flow.

# 19.1 WebSession Data Members

The following table lists and describes the data members of the WebSession object.

Table 19-1 WebSession Object Data Members

| Data Members | Data Type | Description |
|---|---|---|
| apiReturn | Return | **Read-only.** The Return object returned to HOA by the API call specified in the method attribute. This attribute is available only after the WebSession object is finalized. |
| errorURL | string | **Optional**. The URL of your site's dynamic page, to which HOA redirects the customer's browser at form submission if initial validation (e.g. credit card Luhn check, expiration date does not begin with 20 ) of the form contents fails.<br>While redirecting the customer's browser to this page, HOA includes the VID of the WebSession object. On this page, fetch the WebSession object with that VID as the search criterion, and extract the reason why HOA's call failed, available through the returnString and returnCode attributes. Use this string to create a failure message to send to the customer in HTML.<br>If you do not specify this attribute, HOA uses the returnURL value. |
| expireTime | dateTime | **Read-only.** The timestamp of when this WebSession object expires. WebSession objects are valid (by default) for one hour. If the customer submits the order form after that time, HOA redirects the customer's browser to the page specified by errorURL.<br>When you fetch a WebSession object, if the current time is past this timestamp and the returnCode and returnString attributes are not populated in the WebSession object, assume that the customer never submitted the form, and that the WebSession object is no longer valid. |
| ipAddress | string | **Required.** The IP address from which the customer requested the order form. When the customer submits the form, HOA checks if the submission originated from the same IP address. If not, HOA does not make the API call specified in the method attribute. Instead, it updates the WebSession object with the error return code 401, and the return string "IP address does not match value associated with WebSession," and redirects the customer's browser to the page specified by errorURL. |
| method | string | **Required.** The CashBox API call made by HOA at form submission. The data loaded in the privateFormValues data member of this WebSession object and the data submitted through the form should be relevant to this call.<br>CashBox supports the AutoBill.update, Transaction.auth, Transaction.authCapture, and PaymentMethod.update calls. To specify a call in this string, concatenate the object name with the method name separated by an underscore, and omit the parentheses, for example, Transaction_authCapture. |

Table 19-1 `WebSession` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `methodParamVal-ues` | `NameValuePair[]` | **Optional.** The values for some of the parameters required by HOA to make the API call specified in the `method` attribute. To avoid hacking, include them here to exclude them at form submission.<br>For example, if the call is `AutoBill.update`, exclude the tolerance threshold in the risk score (`minChargebackProbability`) at form submission. The name for the value is the flattened object name, method name, and parameter name, concatenated with an underscore, for example, `AutoBill_Update_minChargebackProbability`.<br>See Section 10: The NameValuePair Object. |
| `nameValues` | `NameValuePair[]` | **Optional.** The name–value pairs to include in the objects created by HOA through the API call specified in the `method` attribute. Include this attribute when initializing the `WebSession` object. For example, if that call creates an `AutoBill` object and you want the latter's transactions to be routed to your payment processor under a specific division ID, include that ID in this name–value pair with the name `vin:Division`.<br>See Section 10: The NameValuePair Object. |
| `postValues` | `NameValuePair[]` | **Read-only**. The name-value pairs stored by HOA in the corresponding `WebSession` object at form submission by the customer if you include non-Vindicia form elements, those with no `vin` prefix in their names, in the order form. On your success or failure page, extract these pairs from the `WebSession` object you fetch.<br>See Section 10: The NameValuePair Object. |

Table 19-1 `WebSession` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `privateFormVal-ues` | `NameValuePair[]` | **Optional.** The object attribute values required by HOA to complete the API call specified in the `method` attribute at form submission. Once this attribute is populated, your application need not pass the related data to the form, which secures it against hacking.<br>For example, if the call is `AutoBill.update`, specify the customer account to which the call applies by populating this attribute with the `Account` object's VID. That way, hackers cannot change that VID in the form, because HOA looks it up only in this data member, `privateFormValues`, instead of from the data in the form.<br>Also, if a Vindicia form element can have only one of several values, include all the values in `privateFormValues`. That way, HOA can verify the validity of the form element's value at form submission. For example, when creating an `AutoBill` object, to enable the customer to choose only one of two billing plans, include the IDs of the two billing plans in this attribute. Afterwards, embed two radio buttons in the form with the same values.<br>The names of the form elements should match the names in this attribute. The names for these pairs follow the same convention as that for order-form elements; see Chapter 13: Hosted Order Automation in the ***CashBox Programming Guide***.<br>**Note:** Commas are a special reserved character for use in this data member, and should be used ***only*** as a separator between multiple possible values for the name of a name-value pair.<br>For example, to create an HOA order form which allows your customer to choose between three Billing Plans with `billingPlanId` gold, silver, and platinum, use the `privateFormValues` to populate the following name-value pair when initiating the WebSession object:<br>`vin_BillingPlan_merchantBillingPlanId = gold,silver,platinum`<br>Then, in the web order form presented to the customer, include a multiple choice field with name `vin_BillingPlan_merchantBillingPlanId`. This field will allow your customer to choose one value from the three offered: gold, silver, and platinum.<br>Do not use commas as values in the privateFormValues for any other purpose.<br>See Section 10: The NameValuePair Object. |
| `returnURL` | string | **Required**. The complete URL of your site's dynamic page, to which HOA redirects the customer's browser at form submission, after HOA has successfully made the API call specified in the `method` attribute.<br>While redirecting the customer's browser to this page, HOA includes the VID of the `WebSession` object. In your code to construct this page, fetch the `WebSession` object with its VID as the search criterion, and the CashBox object created by the API call specified in the `method` attribute. Afterwards, extract the information from the fetched objects and create a success message in HTML to send to the customer. |

Table 19-1 `WebSession` Object Data Members *(Continued)*

| Data Members | Data Type | Description |
|---|---|---|
| `version` | string | The CashBox API version HOA should use for the call specified in the `method` attribute. This value must be 3.3 or higher. |
| `VID` | string | Vindicia's Globally Unique Identifier (GUID) for this object. When creating a new `WebSession` object, leave this field blank; it will be automatically populated by CashBox.<br>We suggest that you embed the VID as a hidden form element named `vin_WebSession_vid` in the order form you present to the customer. That way, when the customer submits the form, HOA can load the corresponding `WebSession` object. |

## 19.2 `WebSession` Methods

The following table lists and summarizes the methods for the `WebSession` object.

Table 19-2 `WebSession` Object Methods

| Method | Description |
|---|---|
| `fetchByVid` | Returns an existing `WebSession` object whose VID matches the input VID. |
| `finalize` | Completes HOA activity by instructing HOA to make the API call to create CashBox objects containing sensitive payment data. Uses data submitted by the order form. |
| `initialize` | Creates a `WebSession` object. |

**Note:** As with all other CashBox methods, be certain to pass all required parameters. Do not rely on CashBox supplying a default value for your method parameters.

## fetchByVid

The `fetchByVid` method returns an existing `WebSession` object that matches the input VID. Make this call from the success or failure page, to which HOA redirects the customer's browser after form submission, and after HOA has created the object according to the corresponding `WebSession` object's `method` attribute. HOA includes the `WebSession` object's VID in the redirection URL to make the VID available to you in your success or failure page code.

### Input

***vid:*** the `WebSession` object's Vindicia identifier, which serves as the search criterion. This VID corresponds to the `vin_WebSession_vid` element in the order form submitted by the customer to HOA.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***session:*** the `WebSession` object that matches the input VID.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| 400 | No VID specified to load session by. |
| 404 | Unable to load session: No match for VID ***vid.*** |
| 500 | Unable to load session by VID ***vid: error-description.*** |

### Example

```
$sessionId = …; //passed in by redirected page

$soap = new WebSession($soapLogin, $soapPwd);

$response = $soap->fetchByVID($sessionId);
if ($response['returnCode'] == 200) {

    $fetchedWs = $response['data']->session;

    // Extract non-Vindicia values submitted by the web order form
    // and process them to prepare the HTML to be returned to
    // the customer
    $postVals = $fetchedWs->getPostValues()

    // Assuming HOA created an AutoBill object, let's fetch it
    $soapAbill = new AutoBill($soapLogin, $soapPwd);
    $resp = $soapAbill->fetchByWebSessionVid($sessionId);

    if ($resp['returnCode'] == 200) {
        $createdAutoBill = $resp['data']->autobill;

        // Get AutoBill contents here to be included in
        // HTML returned to the customer.
    }
    else {
        // Return error message to customer
    }
}
else {
    // Return error message to the customer
}
```

# finalize

The `finalize` method instructs Vindicia's Hosted Order Automation solution (HOA) to make the API call you specified in the `WebSession` object's `method` attribute to create CashBox objects containing sensitive payment data. Before you make this call, HOA has all the necessary data to create the CashBox objects available to it through the attributes of the `WebSession` object you populated when you initialized it, and the data the customer submits on the order form.

Call this method from the success page to which HOA redirects the customer's browser after that customer submits the order form containing sensitive payment information. Specify the URL of the success page in the `returnURL` attribute of the `WebSession` object when you initialize the `WebSession` object after the customer requests the form. When the customer submits the form, HOA receives the form data and stores it before redirecting the customer's browser to the success page. The VID of the `WebSession` object embedded in the form identifies the context in which the customer submitted the form. It is available to you in your success page as a parameter to the redirected URL. Thus, in your success page code you know which `WebSession` object instance you should finalize.

When you call `finalize()` on the `WebSession` object, HOA not only makes the API call specified in the `WebSession` object's `method` attribute, but also updates the `WebSession` object with results of the API call it made. These results are available to you in the updated `WebSession` object that is included in the response of this call (check the `returnCode` and `returnString` attributes of the `WebSession` object). Examine the results to determine the content of the customer's browser page that awaits the response to the form submission.

## Input

***session:*** the `WebSession` object to finalize. Include the VID of the object here. HOA passes this VID in as a URL parameter when it redirects the customer's browser to your success page from which you made this call.

## Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***session:*** the `WebSession` object updated with results of the CashBox API call specified in the `method` attribute, which HOA makes as a result of this call to create CashBox objects containing sensitive payment data.

## Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `400` | One of the following:<br>• `Unable to load session:` ***error-description.***<br>• `Must specify a WebSession to finalize!` |

## Example

```
$sessionId = …; //passed in by redirected page

$ws = new WebSession($soapLogin, $soapPwd);
$ws->setVID($sessionId);

// finalize the WebSession so HOA can make the API call to
// create CashBox object/s containing sensitive payment
// information

$response = $ws->finalize();

if ($response['returnCode'] == 200) {

    $updatedWs = $response['data']->session;

    // Check if the API call HOA made to create the
    // CashBox object containing sensitive payment
    // data succeeded

    if ($updatedWs->apiReturn->returnCode == 200) {

        // Extract non-Vindicia values submitted by the web
        // order form and process them to prepare the HTML to
        // be returned to the customer

        $postVals = $updatedWs->getPostValues()

        // Assuming HOA created an AutoBill object, let's fetch it

        $soapAbill = new AutoBill($soapLogin, $soapPwd);
        $resp = $soapAbill->fetchByWebSessionVid($sessionId);

        if ($resp['returnCode'] == 200) {
            $createdAutoBill = $resp['data']->autobill;

            // Get AutoBill contents here to be included in
            // HTML returned to the customer.
        }
    }
    else {
        // The API call HOA made to create or manipulate object
        // containing sensitive payment data did not succeed.
        // Return error message to customer

        $errorString =
            $updatedWs->apiReturn->returnString();

        …

    }
}
else {

    // Finalization failed
    // Return error message to the customer
}
```

## initialize

The `initialize` method creates a `WebSession` object. Call this method before presenting your HOA-based Web order form to your customer. The call returns the new `WebSession` object with a populated `VID` attribute. Embed that VID in the order form as a hidden form element with the name `vin_WebSession_vid` to make it available to HOA at form submission.

To create a `WebSession` object, set the values for its data members (see Section 19.1: WebSession Data Members) and then call `initialize()` to store the changes in the Vindicia database. Do not set a value for `VID` because CashBox automatically generates that when you call `initialize()`.

### Input

***session:*** the `WebSession` object to create.

### Output

***return:*** an object of type `Return` that indicates the success or failure of the call.

***session:*** the `WebSession` object that contains the data that you passed, the VID, and the `expireTime` value assigned by CashBox.

### Returns

In addition to those listed in Table 1: Standard Return Codes, this call returns:

| Return Code | Return String |
|---|---|
| `402` | One of the following:<br>• `Missing required parameter: version` ***version***.<br>• `Invalid parameter: Unsupported version.`<br>• `Missing required parameter: method.`<br>• `Invalid parameter: Unsupported method.`<br>• `Missing required parameter: returnURL.` |

### Example

```
// to create a WebSession object

$ws = new WebSession();

// HOA should make an AutoBill.update call when the form is submitted

$ws->setMethod('AutoBill_Update');

// Customer's IP address. When customer submits the form
// it should come from the same IP address
$ws->setIpAddress("124.23.210.175");

// Page to which HOA will redirect customer's browser
// after successfully making the AutoBill.update call when the
// customer submits the form
$ws->setReturnURL("https://merchant.com/subscribe/success.php");

// Page to which HOA will redirect customer's browser
// if the AutoBill.update call it makes when the customer submits
// the form unsuccessful

$ws->setErrorURL("https://merchant.com/subscribe/failed.php");
```

```
// Private name values pairs. These are needed to create the
// AutoBill object, but we do not want them to appear in the
// form the customer fills in

$pnv1 = new NameValuePair();

// The name is flattened Object name concatenated
// with attribute names with an underscore.

// The CashBox Account object for which HOA should create the
// AutoBill object
$pnv1->setName('Account_VID');
$pnv1->setValue('36c8de2cb74b2c2b08b259cf231ac8d90d1bb3b8');

// The CashBox Product object HOA should use in constructing
// the AutoBill object
$pnv2 = new NameValuePair();
$pnv2->setName('Product_merchantProductId');
$pnv2->setValue('StartWars II');

$pnv3 = new NameValuePair();
$pnv3->setName('vin_BillingPlan_merchantBillingPlanId');

// When customer submits the form, the billing plan
// should be one of the two comma separated values
$pnv3->setValue('GoldAccess2010, PlatinumAccess2010');

$ws->setPrivateFormValues(array($pnv1, $pnv2, $pnv3));

// Method parameter name values pairs. These are needed to make the
// AutoBill.update call which takes parameters in addition to the
// AutoBill object itself. We do not want these to come from the form
// submission because that makes them susceptible to hacking

$mpnv1 = new NameValuePair();

// The name is flattened object name, method name, and parameter
// name concatenated with an underscore.

$mpnv1->setName('AutoBill_Update_minChargebackProbability');
$mpnv1->setValue('80');

// Leave other parameter values to their default values
$ws->setMethodParamValues(array($mpnv1));

// Now create the WebSession object on Vindicia servers
// by making the SOAP call to initialize the object

$response = $ws->initialize();

if ($response->['returnCode'] == 200) {

   $ret_ws = $response['data']->session;

   // The VID of the WebSession object serves as session id

   $sessionId = $ret_ws->getVID();

   // Embed the sessionId as hidden field in the order web form
   // Compose and present the order web form here
}
else {
// Return error to the customer who requested the web order form
}
```