CashBox® API Guide Preface

CashBox API Overview .................................................. 2
Input Parameters ....................................................... 2
Merchant Identifiers ..................................................... 2
The Return Object ....................................................... 4

Chapter 1 The Account Object ........................................ 1-6
1.1 Account Object Hierarchies ....................................... 1-7
1.2 Account Data Members ............................................. 1-8
1.3 Account Subobjects ............................................... 1-11
   Credit Subobject .................................................. 1-11
   CurrencyAmount Subobject ...................................... 1-11
   TimeInterval Subobject ......................................... 1-12
   TokenAmount Subobject ......................................... 1-13
   EmailPreference Subobject ..................................... 1-14
   TaxExemption Subobject ........................................ 1-14
1.4 Account Methods .................................................. 1-14
   addChildren ...................................................... 17
   decrementTokens ............................................... 19
   extendEntitlementByInterval .................................. 21
   extendEntitlementToDate ....................................... 23
   fetchAllCreditHistory ........................................... 25
   fetchByEmail .................................................... 28
   fetchByExternalId .............................................. 29
   fetchByMerchantAccountId .................................... 30
   fetchByPaymentMethod ........................................ 31
   fetchByVid ...................................................... 33
   fetchByWebSessionVid ......................................... 34
   fetchCreatedSince ............................................. 36
   fetchCreditHistory ............................................. 38
   fetchFamily ..................................................... 40
   grantCredit ...................................................... 41
   grantEntitlement ............................................... 43
   incrementTokens ............................................... 44
   isEntitled ........................................................ 46
   makePayment .................................................... 48
   redeemGiftCard .................................................. 51
   removeChildren .................................................. 53
   reversePayment ................................................ 55
   revokeCredit .................................................... 57
revokeEntitlement ........................................... .59
stopAutoBilling ................................................. .61
tokenBalance .................................................. .63
tokenTransaction ................................................ .64
transfer ............................................................. .67
transferCredit ...................................................... .69
update .................................................................. .71
updatePaymentMethod ............................................ .73

Chapter 2  The Activity Object ................................. 1-79
  2.1 Activity Data Members ...................................... 1-80
  2.2 Activity Subobjects ........................................... 1-81
     ActivityCallType Subobject ................................. 1-81
     ActivityCancelInitType Subobject ......................... 1-82
     ActivityCancellation Subobject ............................ 1-82
     ActivityEmailContact Subobject ........................... 1-82
     ActivityFulfillment Subobject ............................. 1-83
     ActivityLogin Subobject ..................................... 1-83
     ActivityLogout Subobject .................................... 1-83
     ActivityNamedValue Subobject ............................... 1-84
     ActivityNote Subobject ....................................... 1-84
     ActivityPhoneContact Subobject ........................... 1-85
     ActivityType Subobject ........................................ 1-86
     ActivityTypeArg Subobject ................................... 1-87
     ActivityURIView Subobject .................................... 1-88
     ActivityUsage Subobject ....................................... 1-88
  2.3 Activity Method ................................................ 1-89
     record ................................................................ 90

Chapter 3  The Address Object ................................. 1-92
  3.1 Address Data Members ....................................... 1-93
  3.2 Address Methods ............................................... 1-94
     fetchByVid ......................................................... 95
     update .............................................................. 96

Chapter 4  The AutoBill Object ................................. 1-97
  4.1 AutoBill Data Members ........................................ 1-98
     AutoBill billingState Data Member Detail .................. 1-103
     AutoBill status Data Member Detail ............................ 1-105
  4.2 AutoBill Subobjects ............................................ 1-107
     AutoBillItem Subobject ......................................... 1-107
Mandate Subobject ............................................. 1-109
PaymentMethod Subobject ................................. 1-110
CancelReason Subobject .................................. 1-110
AutoBillCancelFee Subobject ......................... 1-110

4.3 AutoBill Methods ........................................ 1-111
activate ..................................................... 113
addCampaign ............................................ 114
addCharge ............................................... 116
cancel ..................................................... 118
changeBillingDayOfMonth ................................ 121
delayBillingByDays ..................................... 123
delayBillingToDate ...................................... 124
fetchAllCancelReason .................................. 126
fetchAllCreditHistory .................................... 127
fetchAllInSeason ....................................... 129
fetchAllOffSeason ...................................... 130
fetchBillingItemHistory ................................. 131
fetchByAccount .......................................... 132
fetchByAccountAndProduct ......................... 134
fetchByEmail ........................................... 136
fetchByMerchantAutoBillId ......................... 137
fetchByVid ............................................... 138
fetchByWebSessionVid ................................. 139
fetchCreditHistory ....................................... 141
fetchDailyInvoiceBillings ............................. 143
fetchDeltaSince ......................................... 145
fetchFutureRebills ..................................... 148
fetchInvoice ............................................. 150
fetchInvoiceNumbers ................................... 152
fetchRemainingPaymentDetails ...................... 153
fetchUpgradeHistoryByMerchantAutoBillId ........ 155
fetchUpgradeHistoryByVid ............................ 156
finalizeCarrierBilling ................................ 157
finalizeCustomerAction ............................... 158
finalizePayPalAuth .................................. 160
grantCredit ............................................. 162
makePayment ............................................ 164
migrate ................................................... 167
modify ..................................................... 174
redeemGiftCard .......................................... 180
reversePayment ......................................... 182
revokeCredit ............................................. 184
settlementQuote .................................................. 186
update .......................................................... 189
updateCancelReason ................................. 196
writeOffInvoice ........................................ 197

Chapter 5  The BillingPlan Object .................. 1-198
5.1 BillingPlan Data Members ...................... 1-199
5.2 BillingPlan Subobjects ......................... 1-202
   BillingPlanPeriod Subobject .................... 1-202
   BillingPlanPeriodType Subobject ............ 1-203
   BillingPlanPrice Subobject ............. 1-203
   BillingPlanStatus Subobject ........... 1-204
   BillingPlanCancelFee Subobject .......... 1-204
5.3 BillingPlan Methods .............................. 1-204
   fetchAll ................................................. 206
   fetchAllInSeason ................................. 207
   fetchAllOffSeason ............................... 208
   fetchByBillingPlanStatus ..................... 209
   fetchByMerchantBillingPlanId ............... 210
   fetchByMerchantEntitlementId .............. 211
   fetchByVid ............................................ 212
   update ................................................. 213

Chapter 6  The Campaign Object ............... 1-215
6.1 Campaign Data Members ....................... 1-216
6.2 Campaign Related Object ......................... 1-219
   CouponCode Object ............................. 1-220
6.3 Campaign Methods ................................. 1-221
   activateCampaign ............................... 222
   activateCode ....................................... 223
   cancelCampaign .................................. 224
   deactivateCampaign ............................ 225
   fetchAllCampaigns ............................... 226
   fetchByCampaignId .............................. 227
   fetchByVid ......................................... 228
   retrieveCouponCodes ............................ 229
   validateCode ..................................... 231

Chapter 7  The Chargeback Object .............. 1-232
7.1 Chargeback Data Members ....................... 1-233
7.2 Chargeback Methods ........................................ 1-235
    fetchByCaseNumber and fetchByReferenceNumber ........ 236
    fetchByMerchantTransactionId .............................. 238
    fetchByStatus ............................................. 239
    fetchByStatusSince ....................................... 242
    fetchByVid ................................................. 244
    fetchDelta ................................................... 246
    fetchDeltaSince ............................................ 248
    report ...................................................... 250
    update ....................................................... 252

Chapter 8 The Entitlement Object ............................... 1-254
    8.1 Entitlement Data Members ............................... 1-256
    8.2 Entitlement Methods .................................... 1-257
        fetchByAccount .......................................... 258
        fetchByEntitlementIdAndAccount ........................ 261
        fetchDeltaSince ....................................... 263

Chapter 9 The GiftCard Object ................................. 1-266
    9.1 GiftCard Data Members .................................. 1-267
    9.2 GiftCard Subobjects ..................................... 1-269
        GiftCardStatus Subobject ................................ 1-269
        GiftCardStatusType Subobject ........................... 1-270
    9.3 GiftCard Methods ......................................... 1-271
        reverse .................................................. 272
        statusInquiry ......................................... 274

Chapter 10 The NameValuePair Object ......................... 1-276
    10.1 NameValuePair Data Members ............................ 1-277
    10.2 NameValuePair Methods ................................ 1-278
        fetchNameValuePairNames ............................... 278
        fetchNameValuePairTypes ............................... 280

Chapter 11 The PaymentMethod Object ......................... 1-281
    11.1 PaymentMethod Data Members ............................ 1-282
    11.2 PaymentMethod Subobjects ............................... 1-286
        Amazon Subobject ....................................... 1-286
        Boleto Subobject ....................................... 1-287
        CarrierBilling Subobject ............................... 1-287
        CreditCard Subobject ................................... 1-288
<table>
<thead>
<tr>
<th>Subobject</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectDebit Subobject</td>
<td>1-290</td>
</tr>
<tr>
<td>ECP Subobject</td>
<td>1-291</td>
</tr>
<tr>
<td>ExtendedCardAttributes Subobject</td>
<td>1-293</td>
</tr>
<tr>
<td>ExtendedVerification Subobject</td>
<td>1-294</td>
</tr>
<tr>
<td>ExternalBilling Subobject</td>
<td>1-294</td>
</tr>
<tr>
<td>HostedPage Subobject</td>
<td>1-295</td>
</tr>
<tr>
<td>MerchantAcceptedPayment Subobject</td>
<td>1-296</td>
</tr>
<tr>
<td>MilitaryStar Subobject</td>
<td>1-297</td>
</tr>
<tr>
<td>PaymentMethodType Subobject</td>
<td>1-298</td>
</tr>
<tr>
<td>PayPal Subobject</td>
<td>1-299</td>
</tr>
<tr>
<td>PhoneNumber Subobject</td>
<td>1-300</td>
</tr>
<tr>
<td>PriceCriteria Subobject</td>
<td>1-301</td>
</tr>
<tr>
<td>Skrill Subobject</td>
<td>1-302</td>
</tr>
<tr>
<td>ApplePay Subobject</td>
<td>1-303</td>
</tr>
<tr>
<td>GooglePay Subobject</td>
<td>1-303</td>
</tr>
</tbody>
</table>

### 11.3 PaymentMethod Methods
- fetchByAccount 305
- fetchByMerchantPaymentMethodId 307
- fetchByVid 308
- fetchByWebSessionVid 310
- update 312
- validate 319

### Chapter 12 The PaymentProvider Object 1-322

#### 12.1 PaymentProvider Data Members 1-323

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataRequest</td>
<td>324</td>
</tr>
<tr>
<td>fetchByAccount</td>
<td>325</td>
</tr>
<tr>
<td>fetchByName</td>
<td>326</td>
</tr>
</tbody>
</table>

#### 12.2 PaymentProvider Methods 1-324

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataRequest</td>
<td>324</td>
</tr>
<tr>
<td>fetchByAccount</td>
<td>325</td>
</tr>
<tr>
<td>fetchByName</td>
<td>326</td>
</tr>
</tbody>
</table>

### Chapter 13 The Product Object 1-327

#### 13.1 Product Data Members 1-328

#### 13.2 Product Subobjects 1-330

<table>
<thead>
<tr>
<th>Subobject</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductDescription Subobject</td>
<td>1-330</td>
</tr>
<tr>
<td>ProductPrice Subobject</td>
<td>1-330</td>
</tr>
<tr>
<td>ProductStatus Subobject</td>
<td>1-331</td>
</tr>
</tbody>
</table>

#### 13.3 Product Methods 1-332

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetchAll</td>
<td>333</td>
</tr>
<tr>
<td>fetchByAccount</td>
<td>335</td>
</tr>
<tr>
<td>fetchByMerchantEntitlementId</td>
<td>337</td>
</tr>
<tr>
<td>fetchByMerchantProductIds</td>
<td>338</td>
</tr>
<tr>
<td>fetchByVid</td>
<td>339</td>
</tr>
</tbody>
</table>
Chapter 14 The RatePlan Object ........................................ 1-342
14.1 RatePlan Data Members ............................................. 1-343
14.2 RatePlan Subobjects .................................................... 1-346
  Event Subobject .......................................................... 1-346
  RatedUnitSummary Subobject ........................................ 1-348
  RatePlanTier Subobject .................................................. 1-349
14.3 RatePlan Methods ....................................................... 1-350
  deductEvent ............................................................... 351
  fetchAll ................................................................. 352
  fetchByMerchantRatePlanId .......................................... 353
  fetchByVid ............................................................... 354
  fetchEventByVid ....................................................... 355
  fetchEvents ............................................................. 356
  fetchUnbilledEvents .................................................. 359
  fetchUnbilledRatedUnitsTotal ...................................... 361
  recordEvent ............................................................ 363
  reverseEvent ........................................................... 364
Chapter 15 The Refund Object ........................................... 1-365
15.1 Refund Data Members .................................................. 1-366
15.2 Refund Subobject ....................................................... 1-369
  RefundItem Subobject ................................................ 1-369
  RefundTokenAction Subobject ....................................... 1-370
15.3 Refund Methods ........................................................ 1-371
  fetchByAccount ....................................................... 372
  fetchByTransaction ................................................... 374
  fetchByVid ............................................................... 376
  fetchDeltaSince ....................................................... 377
  perform ................................................................. 379
  report ................................................................. 382
Chapter 16 The SeasonSet Object ........................................ 1-384
16.1 SeasonSet Data Members .............................................. 1-385
16.2 SeasonSet Methods .................................................... 1-386
  fetchAll ................................................................. 387
  fetchAllInSeason .................................................... 388
  fetchAllOff Season ................................................... 389
fetchByMerchantSeasonSetId .......................... 390
fetchByVid .............................................. 391
fetchCurrentSeason ................................. 392
fetchNextSeason ........................................ 393
isInSeason ............................................. 394
update ................................................. 395

Chapter 17 The Token Object ....................... 1-396
  17.1 Token Data Members ............................ 1-397
  17.2 Token Methods .................................. 1-398
    fetch ............................................... 399
    update ............................................ 400

Chapter 18 The Transaction Object ............... 1-401
  18.1 Transaction Data Members ................. 1-404
  18.2 Transaction Subobjects ..................... 1-411
    AVSMatchType Subobject ................... 1-411
    Mandate Subobject ......................... 1-412
    MigrationTaxItem Subobject ............. 1-412
    MigrationTransaction Subobject ......... 1-413
    MigrationTransactionItem Subobject .... 1-416
    MigrationTransactionType Subobject .... 1-417
    TransactionItem Subobject .............. 1-419
    TransactionStatus Subobject ........... 1-423
    TransactionStatusBoleto Subobject .... 1-425
    TransactionStatusCreditCard Subobject .. 1-426
    TransactionStatusECP Subobject ........ 1-426
    TransactionStatusHostedPage Subobject .. 1-427
    TransactionStatusPayPal Subobject ...... 1-427
    TransactionStatusSkrill Subobject ...... 1-428
    TransactionStatusType Subobject ....... 1-428
    TransactionValidationResponse Subobject .. 1-430
  18.3 Transaction Methods ......................... 1-431
    addressAndSalesTaxFromPayPalOrder ..... 433
    auth ............................................. 436
    authCapture ..................................... 443
    calculateSalesTax ............................. 453
    cancel ........................................... 457
    capture ......................................... 460
    fetchByAccount ................................. 464
    fetchByAutoBill .................................. 466
Chapter 19  The WebSession Object ................................. 1-494

19.1 WebSession Data Members ................................. 1-495

19.2 WebSession Methods ......................................... 1-499
  fetchByVid ................................................ 500
  finalize ................................................... 502
  initialize ............................................... 505
CashBox® API Guide Preface

CashBox is an on-demand solution for one-time and recurring 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 to 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.

Merchant Identifiers

All CashBox Objects have a VID (a Vindicia Globally Unique Identifier), which is generated by the system, and a merchant-defined identifier, which is generated or provided by the merchant system. In the newly supported REST API, Merchant Identifiers containing the
forward slash (/) character will no longer be allowed. You must remove any forward slash characters from your merchant identifiers before you can use the REST API. Meanwhile, you can continue to use the forward slash in SOAP versions prior to release 23.0.0, but you must eliminate all instances of the forward slash before you can upgrade. As policy going forward, the "/" character will be prohibited from use within any Merchant Identifiers, across API versions.
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. To help you debug your application in the development and production phases, use the CashBox API to generate a log of `returnString`.

- **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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>The call succeeded.</td>
</tr>
<tr>
<td>400</td>
<td>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.</td>
</tr>
<tr>
<td>403</td>
<td>The Vindicia server cannot authenticate your request.</td>
</tr>
<tr>
<td>500</td>
<td>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.</td>
</tr>
<tr>
<td>503</td>
<td>A Vindicia back-end service, such as a database, is unavailable. Retry your call later.</td>
</tr>
</tbody>
</table>
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.

Note: Use of the forward slash character (/) in identifiers is not allowed. See Merchant Identifiers for more information.
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.

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>company</td>
<td>string</td>
<td>The customer’s company name, if specified.</td>
</tr>
<tr>
<td>createdDate</td>
<td>dateTime</td>
<td>Date the Account was created.</td>
</tr>
<tr>
<td>credit</td>
<td>Credit</td>
<td>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. Do not directly set the value of this attribute. To manipulate credit available to this Account, use methods such as grantCredit() or revokeCredit(). See the Credit Subobject.</td>
</tr>
<tr>
<td>defaultCurrency</td>
<td>string</td>
<td>The three-character ISO 4217 currency code CashBox uses when you pass in a one-time Transaction or AutoBill without specifying a currency. Also, you can request Client Services to define a Global default currency. When you do not specify a currency, CashBox determines the currency to use by proceeding from the specific to the general, in the following order: 1. Transaction 2. AutoBill 3. Account 4. Global (merchant) default 5. Vindicia default If you specify a currency for a Transaction or AutoBill, CashBox will always use that code. Defaults are applied only when no currency code is specified. Because one-time Transactions have no associated AutoBill, if you create a new Transaction without specifying a currency, CashBox uses the defaultCurrency of the Account. If there is none set in Account, CashBox uses the Global default. If no Global default is set, CashBox applies the Vindicia default, which is USD.</td>
</tr>
<tr>
<td>Data Members</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>emailAddress</td>
<td>string</td>
<td>The email address for this Account object, if specified.</td>
</tr>
<tr>
<td>emailTypePreference</td>
<td>EmailPreference</td>
<td>The CashBox enumerated data type that specifies whether to send email to Account as plain text or HTML. See the EmailPreference Subobject.</td>
</tr>
<tr>
<td>entitlements</td>
<td>Entitlement</td>
<td>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. See Entitlement Data Members.</td>
</tr>
<tr>
<td>externalId</td>
<td>string</td>
<td>Optional. An additional unique identifier for this Account. A caller can use either the VID, merchant Account ID (merchantAccountId), or this field to uniquely reference a specific account when making API calls to CashBox. Best Practice is to use this field when you have two external (non-CashBox) systems using two different identification schemes for the same customer. Note: the externalId is enforced to be unique or null in the Vindicia system. Attempting to create an Account with the existing externalId returns Failed to update Account with no specific error.</td>
</tr>
<tr>
<td>merchantAccountId</td>
<td>string</td>
<td>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 can refer to the Account using the ID for future operations. Note: use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>The customer’s name, if specified. This name usually corresponds to the name on the credit card listed for the Account.</td>
</tr>
<tr>
<td>nameValues</td>
<td>NameValuePair[]</td>
<td>Optional. An array of name-value pairs associated with the customer for later reference.</td>
</tr>
<tr>
<td>parentMerchantAccountId</td>
<td>string</td>
<td>The merchant unique identifier for the parent Account of this Account. If an account has a parent account, the Account Object is populated with the MerchantAccountId of the parent account. If the account does not have a parent account, this is null.</td>
</tr>
</tbody>
</table>
## Table 1-1 Account Object Data Members (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>paymentMethods</td>
<td>PaymentMethod[]</td>
<td>A list of default methods, one of which will be applied to a recurring transaction generated for this <code>Account</code> object if the customer has not explicitly specified a payment method for a subscription (<code>AutoBill</code>). Mark the payment methods active or inactive and sort them in order of preference. The first <code>paymentMethod</code> in the sort order will be used as the default. See <code>PaymentMethod Data Members</code>.</td>
</tr>
<tr>
<td>preferredLanguage</td>
<td>string</td>
<td>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. If you use the CashBox email notification feature, and have uploaded an email template in the preferred language to CashBox, CashBox notifies the customer in this language.</td>
</tr>
<tr>
<td>shippingAddress</td>
<td>Address</td>
<td>The customer’s shipping address. This field is optional if, for example, it is the same as <code>billingAddress</code>. CashBox looks up this address first when calculating a transaction’s sales tax for this <code>Account</code> object. See <code>Address Data Members</code>.</td>
</tr>
<tr>
<td>taxExemptions</td>
<td>TaxExemption[]</td>
<td>An array of default exemptions for the sales tax on this <code>Account</code>’s transactions. Multiple tax exemptions may be defined. See the <code>TaxExemption Subobject</code>.</td>
</tr>
<tr>
<td>taxType</td>
<td>string</td>
<td>Account-level code used with external tax system rules for exemption behavior and custom tax handling.</td>
</tr>
<tr>
<td>tokenBalances</td>
<td>TokenAmount[]</td>
<td>An array of <code>TokenAmount</code> 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 <code>Account</code> object in response to an <code>update()</code> call. See the <code>TokenAmount Subobject</code>.</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>The Vindicia Globally Unique Identifier (GUID) for this object. When creating a new <code>Account</code> object, leave this field blank; it will be automatically populated by CashBox.</td>
</tr>
<tr>
<td>warnBeforeAutoBilling</td>
<td>Boolean</td>
<td>A Boolean flag, if set to <code>true</code>, and if you are using the CashBox email notification feature, triggers an email notification to the customer before every recurring billing.</td>
</tr>
</tbody>
</table>
1.3 Account Subobjects

The Account object has several subobjects:

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

Credit Subobject

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

Table 1-2 Credit Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>currency-Amounts</td>
<td>CurrencyAmount</td>
<td>An array of CurrencyAmount objects.</td>
</tr>
<tr>
<td>timeIntervals</td>
<td>TimeInterval</td>
<td>An array of TimeInterval objects, each of which specifies a unit of time (day, week, month, year) and its amount.</td>
</tr>
<tr>
<td>tokenAmounts</td>
<td>TokenAmount</td>
<td>An array of TokenAmount objects. Each TokenAmount 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.</td>
</tr>
</tbody>
</table>

CurrencyAmount Subobject

Defines the Currency Credit.

Table 1-3 CurrencyAmount Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>decimal</td>
<td>The amount of currency granted. Must be a positive value.</td>
</tr>
<tr>
<td>currency</td>
<td>string</td>
<td>The ISO 4217 currency code used for the currency amount. Default is USD.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>A description of the currency grant.</td>
</tr>
<tr>
<td>nameValues</td>
<td>NameValuePair</td>
<td>An optional array of name-value pairs to associate with this currency credit.</td>
</tr>
</tbody>
</table>

See Section 10: The NameValuePair Object.
Table 1-3  CurrencyAmount Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason</td>
<td>string</td>
<td>The reason for the currency credit.</td>
</tr>
</tbody>
</table>
| source       | string    | A read-only field indicating the source of the corresponding credit. Possible values include:  
  • Account  
  • AutoBill  
  • Unknown—indicates the credit is free-floating and not associated at the Account or AutoBill level. |
| 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. |

TimeInterval Subobject

Defines the Time Interval Credit.

Table 1-4  TimeInterval Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountVID</td>
<td>string</td>
<td>Vindicia's Globally Unique Identifier (GUID) for the account associated with this object.</td>
</tr>
<tr>
<td>autoBillVID</td>
<td>string</td>
<td>Vindicia's Globally Unique Identifier (GUID) for the AutoBill associated with this object.</td>
</tr>
<tr>
<td>days</td>
<td>integer</td>
<td>Number of days in this time interval.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>A description of the time interval grant.</td>
</tr>
<tr>
<td>granted</td>
<td>dateTime</td>
<td>A time stamp that shows when the credit was granted.</td>
</tr>
<tr>
<td>grantedByCashbox</td>
<td>boolean</td>
<td>Indicates if the interval credit was granted by CashBox.</td>
</tr>
<tr>
<td>merchantAccountId</td>
<td>string</td>
<td>The merchant's unique identifier for the account associated with this object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong>: use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>The merchant's unique identifier for the AutoBill associated with this object.</td>
</tr>
<tr>
<td>months</td>
<td>integer</td>
<td>Number of months in this time interval.</td>
</tr>
</tbody>
</table>
### TokenAmount Subobject

Defines the Token Credit.

#### Table 1-5  TokenAmount Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>integer</td>
<td>The number of Tokens to be credited.</td>
</tr>
<tr>
<td>token</td>
<td>Token</td>
<td>The Type of Token for this Credit. See Section 17.1: Token Data Members.</td>
</tr>
</tbody>
</table>
| source       | string    | A read-only field indicating the source of the corresponding credit. Possible values include:  
  • Account  
  • AutoBill  
  • Unknown—indicates the credit is free-floating and not associated at the Account or AutoBill level. |
EmailPreference Subobject

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

Table 1-6  EmailPreference Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>html</td>
<td>string</td>
<td>The customer prefers to receive email in HTML format.</td>
</tr>
</tbody>
</table>
| multipart    | string    | The customer prefers to receive email messages in mixed media format.  
  |             | **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 1-7  TaxExemption Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>Boolean</td>
<td>If set to true, specifies that the exemption is active and serves as a criterion for calculation of sales tax.</td>
</tr>
<tr>
<td>exemptionId</td>
<td>string</td>
<td>Specifies the type of exemption, such as the U.S. Tax ID or value-added tax (VAT) ID.</td>
</tr>
<tr>
<td>region</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>

1.4  Account Methods

The following table summarizes the methods for the Account object.

Table 1-8  Account Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addChildren</td>
<td>Creates a parent-child relationship.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>decrementTokens</td>
<td>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.</td>
</tr>
<tr>
<td>extendEntitlementByInterval</td>
<td>Extends an Account entitlement by the interval specified. (The entitlement must already exist and be on the Account when this method is called.)</td>
</tr>
<tr>
<td>extendEntitlementToDate</td>
<td>Extends an account entitlement to the date specified. (The entitlement must already exist and be on the Account when this method is called.)</td>
</tr>
<tr>
<td>fetchAllCreditHistory</td>
<td>Returns all credit grants and decrements for all Accounts.</td>
</tr>
<tr>
<td>fetchByEmail</td>
<td>Returns the Account objects with the specified email address.</td>
</tr>
<tr>
<td>fetchByMerchantAccountId</td>
<td>Returns the Account with the specified ID (merchantAccountId).</td>
</tr>
<tr>
<td>fetchByPaymentMethod</td>
<td>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.</td>
</tr>
<tr>
<td>fetchByVid</td>
<td>Returns the Account object with the specified VID.</td>
</tr>
<tr>
<td>fetchByWebSessionVid</td>
<td>Returns the Account object with the specified WebSession VID.</td>
</tr>
<tr>
<td>fetchCreatedSince</td>
<td>Returns Account objects created between a specified time window.</td>
</tr>
<tr>
<td>fetchCreditHistory</td>
<td>Returns an audit log of credit-related events for an Account, or for all Accounts.</td>
</tr>
<tr>
<td>fetchFamily</td>
<td>Returns the family of the given Account.</td>
</tr>
<tr>
<td>grantCredit</td>
<td>Adds a specified amount of credit to an Account.</td>
</tr>
<tr>
<td>grantEntitlement</td>
<td>Grants entitlement to an Account.</td>
</tr>
<tr>
<td>incrementTokens</td>
<td>Adds the specified number of tokens to this Account. This method is equivalent to tokenTransaction with positive values for the token amounts.</td>
</tr>
<tr>
<td>isEntitled</td>
<td>Determines whether or not an Account has an entitlement. This checks Account entitlements, as well as entitlements associated with the Account’s AutoBills.</td>
</tr>
<tr>
<td>makePayment</td>
<td>Enters a payment against the Account.</td>
</tr>
<tr>
<td>redeemGiftCard</td>
<td>Redeems a specified gift card and adds the corresponding credit to an Account.</td>
</tr>
<tr>
<td>removeChildren</td>
<td>Removes a child or multiple children from a parent.</td>
</tr>
</tbody>
</table>
Table 1-8  Account Object Methods  (Continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reversePayment</td>
<td>Reverses an Account payment made using makePayment. This method may only be used with payments using MerchantAcceptedPayment payment methods.</td>
</tr>
<tr>
<td>revokeCredit</td>
<td>Deducts a specified amount of credit from the Account.</td>
</tr>
<tr>
<td>revokeEntitlement</td>
<td>Revokes an entitlement from the Account. Note: This method will revoke only those Entitlements granted using the grantEntitlement method; it will not revoke entitlements acquired through an AutoBill.</td>
</tr>
<tr>
<td>stopAutoBilling</td>
<td>Cancels one or more AutoBill objects (subscriptions) associated with this Account object.</td>
</tr>
<tr>
<td>tokenBalance</td>
<td>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.</td>
</tr>
<tr>
<td>tokenTransaction</td>
<td>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.</td>
</tr>
<tr>
<td>transfer</td>
<td>Merges the target Account with a given (source) Account, and returns the target Account with the merged content.</td>
</tr>
<tr>
<td>transferCredit</td>
<td>Transfers credits from one Account to another.</td>
</tr>
<tr>
<td>update</td>
<td>Creates or updates an Account object.</td>
</tr>
<tr>
<td>updatePaymentMethod</td>
<td>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.</td>
</tr>
</tbody>
</table>
addChildren

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

**Input**

**srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**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:

<table>
<thead>
<tr>
<th>payerReplacementBehavior</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReplaceOnAllAutoBills</td>
<td>This option specifies that any AutoBills that the child has, or will have, are to be paid by the parent Account.</td>
</tr>
<tr>
<td>ReplaceOnlyFutureAutoBills</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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";
}
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

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

- **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

```
$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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Entitlement not specified.</td>
</tr>
<tr>
<td></td>
<td>• Base Account not specified.</td>
</tr>
<tr>
<td></td>
<td>• Extension interval not specified.</td>
</tr>
<tr>
<td></td>
<td>• Account not found.</td>
</tr>
<tr>
<td></td>
<td>• Entitlement not found.</td>
</tr>
<tr>
<td></td>
<td>• Entitlement extension failed: <code>error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Failed to save Account after entitlement extension: <code>error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Failed to reload account after entitlement extension: <code>error-description</code>.</td>
</tr>
</tbody>
</table>
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**

* srd: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

* 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Entitlement not specified.</td>
</tr>
<tr>
<td></td>
<td>• Base Account not specified.</td>
</tr>
<tr>
<td></td>
<td>• Extension date not specified.</td>
</tr>
<tr>
<td></td>
<td>• Account not found.</td>
</tr>
<tr>
<td></td>
<td>• Entitlement not found.</td>
</tr>
<tr>
<td></td>
<td>• Failed to convert extension date: error-description.</td>
</tr>
<tr>
<td></td>
<td>• Entitlement extension failed: error-description.</td>
</tr>
<tr>
<td></td>
<td>• Failed to save Account after entitlement extension: error-description.</td>
</tr>
<tr>
<td></td>
<td>• Failed to reload account after entitlement extension: error-description.</td>
</tr>
</tbody>
</table>
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**

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

**Table 1-10  CreditEventType Object Enumeration Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumption</td>
<td>Credit decremented due to use in a recurring or one time transaction</td>
</tr>
<tr>
<td>GiftCardRedemption</td>
<td>Credit added due to a redemption of a gift card.</td>
</tr>
<tr>
<td>GiftCardReversal</td>
<td>Credit decremented due to a reversal of a gift card that was previously redeemed.</td>
</tr>
<tr>
<td>GiftCardStatusInquiry</td>
<td>No change in credit.</td>
</tr>
<tr>
<td>Grant</td>
<td>Credit added due to a credit grant you made.</td>
</tr>
<tr>
<td>Refund</td>
<td>Credit added due to refund of a credit based transaction.</td>
</tr>
<tr>
<td>Revocation</td>
<td>Credit decremented due to a credit revocation you made.</td>
</tr>
</tbody>
</table>

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

**srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

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

**endTimestamp**: the ending time stamp (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 time stamp and event type.

Returns

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load account.</td>
</tr>
<tr>
<td></td>
<td>• No matching credit events found.</td>
</tr>
<tr>
<td></td>
<td>• Invalid value or values of time stamp, and/or page, and/or page size.</td>
</tr>
</tbody>
</table>
**Example**

// to fetch all credit history for an account

```php
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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Must specify email address to load by!</td>
</tr>
<tr>
<td>404</td>
<td>One of the following:</td>
</tr>
</tbody>
</table>

  - Unable to load account by `emailAddress` input-`emailAddress`: No match.
  - No AutoBills found for email address `input-emailAddress`: No match.
  - Unable to load account by email address `input-emailAddress`: No match.

**Example**

```php
// 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";
}
```
fetchByExternalId

The `fetchByExternalId` method returns an `Account` object whose ID, the `ExternalId` assigned by you, matches the input. To fetch an account by an external ID, specify a unique value for the `externalId` field of the `Account` object when you create it.

Beginning in release 22.0, you can use this optional new unique ID in place of the `merchantAccountId`, or in conjunction with it by mapping the `merchantAccountId` to your merchant-facing Interface, and the `externalId` to your external billing system.

The VID, the `ExternalId`, and the `merchantAccountId` each identify an account in any API call, and this ID is part of the `Account` object in the API and visible in the Portal.

**Input**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`externalId`: your Account ID (`externalId`), 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 (`externalId`) matches the input.

**Returns**

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load account by <code>externalId</code> <strong><code>externalId</code>: No match.</strong></td>
</tr>
<tr>
<td></td>
<td>• Unable to load account by <code>externalId</code> <strong><code>externalId</code>: error-description.</strong></td>
</tr>
<tr>
<td></td>
<td>• Must specify <code>externalId</code> to load by!</td>
</tr>
</tbody>
</table>

**Example**

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

// now load an account into the Account object
// by (unique) Account ID
$response = $account->fetchByByExternalId($externalId);
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";
}
```
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**

- `srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- `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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load account by VID <code>input-merchantAccountId</code>:</td>
</tr>
<tr>
<td></td>
<td>No match.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load account by VID <code>input-merchantAccountId</code>:</td>
</tr>
<tr>
<td></td>
<td>Vindicia internal error.</td>
</tr>
<tr>
<td></td>
<td>• Must specify merchantAccountId to load by!</td>
</tr>
</tbody>
</table>

**Example**

```php
// 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

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

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

**Example**

```php
$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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load account by VID input-vid: No match.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load account by VID input-vid: Vindicia internal error.</td>
</tr>
<tr>
<td></td>
<td>• Must specify VID to load by!</td>
</tr>
</tbody>
</table>

**Example**

```php
$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**

- **sr**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the sr. A null sr returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Missing required parameter 'vid'.</td>
</tr>
<tr>
<td>404</td>
<td>Unable to find requested Account: No matches.</td>
</tr>
</tbody>
</table>
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
}
fetchCreatedSince

The fetchCreatedSince method returns Account objects created between a specified time window.

Input

srdd: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

startTime: the starting time stamp (lower limit) for the range of Accounts you wish to retrieve.

endTime: the ending time stamp (upper limit) for the range of Accounts 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.

accounts: an array of account objects matching the search criteria.

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load account.</td>
</tr>
<tr>
<td></td>
<td>• Invalid value or values of time stamp, and/or page, and/or page size.</td>
</tr>
<tr>
<td></td>
<td>• No matching records found.</td>
</tr>
</tbody>
</table>
Example

```php
$acct = new Account();

$page = 0;        // paging begins at 0
$pageSize = 5;   // five records
$startTime = '2015-01-01T22:34:32.265Z';
$endTime = '2015-01-30T22:34:32.265Z';

do {
    $ret =$acct->fetchCreatedSince($startTime, $endTime, $page, $pageSize);

    $count = 0;
    if ($ret['returnCode'] == 200) {
        $fetchedAccounts = $ret['accounts'];
        $count = sizeof($fetchedAccounts);
        foreach ( $fetchedAccounts as $act ) {
            // Process account details
        }
        $page++;
    }
} while ($count > 0);
```
fetchCreditHistory

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

**Input**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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 time stamp (lower limit) for the range of credit event logs you wish to retrieve.

`endTimestamp`: the ending time stamp (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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>- Unable to load account.</td>
</tr>
<tr>
<td></td>
<td>- Invalid value or values of time stamp, and/or page, and/or page size.</td>
</tr>
<tr>
<td>404</td>
<td>No matching credit events found.</td>
</tr>
</tbody>
</table>
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**

**srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**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**

```php
$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**

**srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Account not found.</td>
</tr>
<tr>
<td></td>
<td>• Failed to translate credit error-description.</td>
</tr>
<tr>
<td></td>
<td>• Failed to grant credit error-description.</td>
</tr>
<tr>
<td></td>
<td>• Failed to save Account after granting credit.</td>
</tr>
<tr>
<td></td>
<td>• Failed to reload Account after granting credit error-description.</td>
</tr>
<tr>
<td></td>
<td>• Time interval credit cannot have amount 0.</td>
</tr>
</tbody>
</table>
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 "$availableTokens
    foreach($availableTokens as $stkAmt) {
        print "$stkAmt->getMerchantTokenId() . " "
        print "$stkAmt->getAmount() . "\n"
    }
} else {
    // Error while granting credit to the account
    print $response['returnString'] . "\n";
}
grantEntitlement

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

**Input**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- `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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Entitlement not specified.</td>
</tr>
<tr>
<td></td>
<td>• Base Account not specified.</td>
</tr>
<tr>
<td></td>
<td>• Account not found.</td>
</tr>
<tr>
<td></td>
<td>• Entitlement grant failed: <code>error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Failed to save Account after entitlement extension: <code>error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Failed to reload account after entitlement extension: <code>error-description</code>.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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**

- `srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

  - `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>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>Integer</td>
<td>The number of tokens.</td>
</tr>
<tr>
<td>Token</td>
<td>Token</td>
<td>The token type, which must be previously defined.</td>
</tr>
</tbody>
</table>

**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**

`srdf`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

- `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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Entitlement not specified.</td>
</tr>
<tr>
<td></td>
<td>• Base Account not specified.</td>
</tr>
<tr>
<td></td>
<td>• Account not found.</td>
</tr>
<tr>
<td></td>
<td>• Entitlement test failed: <code>error-description</code>.</td>
</tr>
</tbody>
</table>
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. You can use this method 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 causes 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

srd: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

account: the Account to which this payment applies.

paymentMethod: the PaymentMethod to be used for this payment. This paymentMethod can be of type MerchantAcceptedPayment, credit card, or PayPal. If you use PayPal, the PayPal payment method must be already populated with a Billing Agreement ID (BAID), acquired either from an earlier AutoBill creation or from a one-time transaction conducted for the customer's account, in which a PayPal reference ID was requested. You can not use an PayPal payment method that requires customer approval at the PayPal site in this call.

Note: For tracking purposes, assign a unique ID for every Account.makePayment call that uses a MerchantAcceptedPayment Payment Method.

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 not specified, the oldest unpaid invoice or AutoBill will be paid first. To make a payment against an invoice in Open status, an ID must be specified—CashBox will not direct overage against such a future invoice.

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

overageDisposition: an object of type PaymentOverageDisposition, defines how to allocate payments in excess of a required AutoBill payment amount. Enumerated values include:

- applyToOldestInvoice (the default)
• applyToThisAutoBill
• 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Account not found.</td>
</tr>
<tr>
<td></td>
<td>• Failed to translate payment method.</td>
</tr>
<tr>
<td></td>
<td>• Failed to make payment: <strong>error-description.</strong></td>
</tr>
<tr>
<td></td>
<td>• Transaction not returned from payment attempt.</td>
</tr>
<tr>
<td></td>
<td>• This Credit Card already exists—Policy Violation” (eradicate the newly created but failed one or ensure it is set INACTIVE).</td>
</tr>
<tr>
<td></td>
<td>You receive this error message when you have the <strong>Credit_Card_Constraints</strong> merchant option enabled on your site. This message means that someone is attempting to use a credit card that is already being used as the payment method on an existing account. This is not allowed.</td>
</tr>
<tr>
<td>405</td>
<td>Payment transaction failed - payment not applied.</td>
</tr>
<tr>
<td>406</td>
<td>Specified Account could not be found - payment not applied.</td>
</tr>
<tr>
<td>407</td>
<td>Specified PaymentMethod could not be found - payment not applied.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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 (where supported). 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

srd: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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:

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

**Example**

```php
// 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 $stkAmt) {
        print "Token type: " . $stkAmt->getMerchantTokenId() . " ";
        print "Amount: " . $stkAmt->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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReplaceOnAllAutoBills</td>
<td>This option will replace the Payment Method on each of the child’s AutoBills with the child’s default (index_number = 0) Payment Method.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>ReplaceOnlyFutureAutoBills</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

**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

```php
$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**

**srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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

**timestamp:** the time that payment reversal occurred. Set 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 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Account not found.</td>
</tr>
<tr>
<td></td>
<td>• Neither paymentId nor invoiceId: indexNumber provided for reversal attempt.</td>
</tr>
<tr>
<td></td>
<td>• Failed to add reverse payment: error-description.</td>
</tr>
</tbody>
</table>
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**

`srdd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a `method` call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdd`. A null `srdd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Account not found.</td>
</tr>
<tr>
<td></td>
<td>• Failed to translate credit <em>error-description</em>.</td>
</tr>
<tr>
<td></td>
<td>• Failed to revoke credit <em>error-description</em>.</td>
</tr>
<tr>
<td></td>
<td>• Failed to save Account after revoking credit.</td>
</tr>
<tr>
<td></td>
<td>• Failed to reload Account after revoking credit <em>error-description</em>.</td>
</tr>
<tr>
<td></td>
<td>• Data validation error: Missing required parameter <code>credit</code>.</td>
</tr>
</tbody>
</table>
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**

- `srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- `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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Entitlement not specified.</td>
</tr>
<tr>
<td></td>
<td>• Base Account not specified.</td>
</tr>
<tr>
<td></td>
<td>• Account not found.</td>
</tr>
<tr>
<td></td>
<td>• Entitlement revocation failed: Could not find active entitlement for revocation.</td>
</tr>
<tr>
<td></td>
<td>• Entitlement revocation failed: <code>error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Failed to save Account after revoking entitlement: <code>error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Failed to reload account after entitlement revocation: <code>error-description</code>.</td>
</tr>
</tbody>
</table>
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

Use the stopAutoBilling method to cancel one or more AutoBill objects (subscriptions) associated with an Account object. Rather than making separate cancel calls for each AutoBill, you can cancel all active AutoBill objects in a single call with this method. Note, however, that any AutoBills with a future start date (status: Pending Activation) will not be automatically canceled. Use AutoBill.cancel() to cancel any AutoBills whose activation is pending.

**Input**

- **srdd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srdd. A null srdd returns the complete response.

- **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.)

- **cancelReason:** (Optional) reason for canceling the AutoBill. You can use predefined CashBox cancel reason codes, or define additional codes using the CashBox Portal or the API. (See "Canceling AutoBills with Reason Codes" in the CashBox Programmer’s Guide.) Supplying an undefined cancel reason code may result in an error.

**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.

- **autobill:** the AutoBill object that was canceled.

**Returns**

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Base Account not specified.</td>
</tr>
</tbody>
</table>
// 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->set('xyz-111');

$autobill2 = new AutoBill();
$autobill2->set('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**

- **srdf**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

- **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**

```php
$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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- **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**

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Data Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>authTimestamp</td>
<td>dateTime</td>
<td>A time stamp that specifies the date and time of when you processed the transaction. Insert this data with your code.</td>
</tr>
<tr>
<td>clearedTimestamp</td>
<td>dateTime</td>
<td>A time stamp that specifies the date and time of when Vindicia processed the transaction. CashBox inserts this data.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Optional. A memo for the transaction.</td>
</tr>
<tr>
<td>tokenAmount</td>
<td>TokenAmount</td>
<td>Required. An enumerated string value that categorizes the type of account activity you are recording.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>401</td>
<td>Balance too low. Returned if one or more transactions requested would drop the user's balance below 0.</td>
</tr>
<tr>
<td>404</td>
<td>Token not found. Returned if one or more tokens specified are not a saved type; however, the tokens available on the account are still returned,</td>
</tr>
</tbody>
</table>
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 " . $acct->merchantAccountId . 
; foreach ($newTokBalances as $newTokBal) {
      print "Token type" . $newTokBal->token->merchantTokenId . 
; print "Token amount available" . $newTokBal->amount; 
}"; 
}
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

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
</table>
| 400         | One of the following:  
• No match found for target account.  
• No match found for source account.  
• Unable to transfer account. |

**Example**

```php
$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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

  - **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• No match found for <code>toAccount</code>.</td>
</tr>
<tr>
<td></td>
<td>• No match found for <code>fromAccount</code>.</td>
</tr>
<tr>
<td>500</td>
<td>Unable to transfer account.</td>
</tr>
</tbody>
</table>
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.)
```

When you update an Account, CashBox edits the existing Payment Method.

- If the `merchantPaymentMethod` of an existing Payment Method is passed, then that Payment Method is edited. If `replaceOnAllAutoBills` is `false`, this edit is applied to all AutoBills using that Payment Method, and no other.
- If a new `merchantPaymentMethodId` is passed, then a new Payment Method is created. If `replaceOnAllAutoBills` is `false`, then the new Payment Method will not be applied to any autoBills. If it is `true`, it will be applied to all.

**Input**

- `srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- `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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Data validation error. Failed to create Payment-Type-Specific Payment Record: Credit Card conversion failed: Credit Card failed Luhn check.</td>
</tr>
<tr>
<td></td>
<td>• Failed to save account.</td>
</tr>
<tr>
<td></td>
<td>• This Credit Card already exists—Policy Violation&quot; (eradicate the newly created but failed one or ensure it is set INACTIVE).</td>
</tr>
<tr>
<td></td>
<td>You receive this error message when you have the Credit_Card_Constraints merchant option enabled on your site. This message means that someone is attempting to use a credit card that is already being used as the payment method on an existing account. This is not allowed.</td>
</tr>
</tbody>
</table>

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();
}

Returns
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 `ignoreAvisPolicy` 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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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.
updateScopeOnAccount: enumerated options controlling whether and when to bill and/or replace the paymentMethod on associated AutoBills belonging to the specified Account. If you do not specify a value, updateScopeOnAccount defaults to the equivalent of the matchingPaymentMethod option, as defined below.

Use the following enumerated values to specify which AutoBills to catch-up billing on, and on which AutoBills you want to replace the payment method.

- None—Do not bill on any AutoBills for this/these Accounts. Do not replace the paymentMethod on this/these Account(s).
- matchingPaymentMethod—Bill only on those ready AutoBills whose payment method matches the default payment method on the specified Account. Replace the payment method only on those AutoBills whose payment method matches the default payment method on the specified Account.
- AllDue—Bill all (account and children) AutoBills with payment methods belonging to the specified Account that are in error or ready to bill. The payment method is replaced on those billed.
- AllDueAndMatching—Bill all (Account and children) AutoBills with payment method belonging to the specified Account that are in error or ready to bill. The payment method is replaced on those billed, as well as on any AutoBills, with a payment method matching the specified Account’s primary payment method.
- AllActive—Bill all ready AutoBills, replace on all AutoBills with payment methods belonging to the specified Account.

Note: The enumerated values are the same for updateScopeOnChildren and updateScopeOnAccount. You can apply them independently or leave them null. For example, you could set AllActive on updateScopeOnChildren and None on updateScopeOnAccount. This would set the payment method on every child AutoBill to the new payment method, set the payment method on the Account to the new payment method, but not change the payment method on any AutoBills belonging directly to the specified (primary/parent) Account.

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

- Update: the paymentMethod is saved and associated with the account. If the updateScopeOnAccount and/or updateScopeOnChildren values are set to a value other than None, or if the updateScopeOnAccount value is left unspecified, the payment method associated with AutoBills within the specified scope of the update will also be set to the new or updated payment method specified in the call.
- Validate: the call attempts to validate the payment method and, if successful, proceeds as in the Update case. Does not update and returns the appropriate 4xx error code if validation fails.
- CatchUp: if there are no AutoBills within updateScopeOnAccount and updateScopeOnChildren eligible for billing, the call proceeds as if Validation were set. Otherwise, for each AutoBill in the updateScopeOnAccount and updateScopeOnChildren purview, catch-up billing is performed such that:
• If the AutoBill is in Good Standing and is not ready to bill, the paymentMethod on the AutoBill is set to the new or updated payment method.

• If the AutoBill is in Good Standing and is due to bill today, or if the AutoBill is in Soft or Hard Error and its end date is after today (that is, still within the retry period), an attempt is made to bill (or reprocess the failed billing attempt) on the AutoBill using the new or updated payment method specified in the Account.updatePaymentMethod() call. If the attempt is successful, the payment method for subsequent billing attempts on the AutoBill is set to the new or updated payment method. If the AutoBill was in error and the reprocessing attempt was successful, the AutoBill status is set to Good Standing. If the AutoBill was in Hard Error, it is reactivated.

• After catch-up billing, if any AutoBill billing or reprocessing attempts were successful, the payment method on the Account is set to the new or updated payment method.

After all catch-up billing and AutoBill payment method updates have been performed:

• If all AutoBill billing attempts were successful and any or all payment methods were updated successfully on the AutoBill(s) and Account, the call returns 200 (success) and a list of merchant AutoBill identifiers of successfully billed AutoBills in the Successes field of the Return object.

• If some AutoBill billing attempts were successful, the call returns 206 (partial success) and a list of merchant AutoBill identifiers of successfully billed or updated AutoBills in the Successes field of the Return object, and a list of merchant AutoBill identifiers of unsuccessfully billed or updated AutoBills in the Failures field of the Return object.

• If the paymentMethod is invalid, or catch-up billing or update failed on all eligible AutoBills (where there were AutoBills eligible for billing or update), the call returns a list of failures in the Return object and an appropriate 4xxx return code.

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 updatePaymentMethod is set to Validate) 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 updatePaymentMethod is set to Validate) the CVN return code will be used to determine whether to update the paymentMethod.

updateScopeOnChildren: enumerated options controlling whether and when to bill and/or replace the payment method on associated AutoBills belonging to the child Accounts of the specified Account. If you do not specify a value, updateScopeOnChildren defaults to the equivalent of the none option.

Use the following enumerated values to specify which child Accounts you want to catch-up billing on, and on which AutoBills you want to replace the payment method.

• None—Do not bill on any AutoBills for this/these Accounts. Do not replace the payment method on this/these Account(s).

• matchingPaymentMethod—Bill only on those ready AutoBills whose payment method matches the default payment method on the specified Account. Replace the payment method only on those AutoBills whose payment method matches the default payment method on the specified Account.
- **AllDue**—Bill all (account and children) AutoBills with payment methods belonging to the specified Account that are in error or ready to bill. The payment method is replaced on those billed.
- **AllDueAndMatching**—Bill all (account and children) AutoBills with payment methods belonging to the specified Account that are in error or ready to bill. The payment method is replaced on those billed, as well as on any AutoBills, with a payment method matching the specified Account's primary payment method.
- **AllActive**—Bill all ready AutoBills, replace on all AutoBills with payment methods belonging to the specified Account.

**Note**

The enumerated values are the same for `updateScopeOnChildren` and `updateScopeOnAccount`. You can apply them independently or leave them null. For example, you could set **AllActive** on `updateScopeOnChildren` and **None** on `updateScopeOnAccount`. This would set the payment method on every child AutoBill to the new payment method, set the payment method on the Account to the new payment method, but not change the payment method on any AutoBills belonging directly to the specified (primary/parent) Account.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>The call succeeded. The call returns a list of merchant AutoBill identifiers of successfully billed AutoBills in the Successes field of the Return object.</td>
</tr>
<tr>
<td>206</td>
<td>Partial success. The call returns a list of merchant AutoBill identifiers of successfully billed or updated AutoBills in the Successes field of the Return object, and a list of merchant AutoBill identifiers of unsuccessfully billed or updated AutoBills in the Failures field of the Return object.</td>
</tr>
<tr>
<td>261</td>
<td>All active AutoBills were updated. AutoBills which are both expired and Suspended cannot be updated.</td>
</tr>
</tbody>
</table>
### Return Code | Return String
---|---
400 | One of the following:
• Invalid Payment Method Type. (You cannot change the Payment Method Type on an existing Payment Method.)
• No PaymentMethod specified in arguments.
• Data validation error Failed to create Payment-Type-Specific Payment Record: Credit Card conversion failed: Credit Card failed Luhn check.
• This Credit Card already exists—Policy Violation (eradicate the newly created but failed one or ensure it is set INACTIVE).
You receive this error message when you have the Credit_Card_Constraints merchant option enabled on your site. This message means that someone is attempting to use a credit card that is already being used as the payment method on an existing account. This is not allowed.

402 | One of the following:
• PaymentMethod failed validation.
• Error attempting to authorize card.
• Unable to authorize card.

404 | No match found **error-description**.
Returned if CashBox cannot find an account that matches the input in the Vindicia database.
An error string and the failures key and list of AutoBills that failed to bill or update.

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
$updateScopeOnAccount = 'AllActive';
$updateScopeOnChildren = 'AllActive';
$response = $account->updatePaymentMethod($paymentMethod, $updateScopeOnAccount, 'CatchUp', 0, 0, $updateScopeOnChildren);

if($response['returnCode'] == 200) {
    print "Call succeeded\n";
    if ( $response['successes'] && $response['successes'][0] ) {
        print $response['successes'][0]." is an invoice successfully rebilled.\n";
    }
} else if($response['returnCode'] == 402) {
    print "Payment method validation failed\n";
    if ( $response['failures'] && $response['failures'][0] ) {
        print $response['failures'][0]." is an invoice unsuccessfully rebilled.\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:

| Note: | Use of the forward slash character (/) in identifiers is not allowed. See Merchant Identifiers for more information. |
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

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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.  
**Note:** use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.  
See Account Data Members. |
| activityArgs | ActivityTypeArg  | Required. An object that details the activity you are recording. The content varies, depending on the activity type specified.  
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().  
See the ActivityType Subobject. |
| timestamp    | dateTime         | Required. A time stamp 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>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fromCustomer-ToMerchant</td>
<td>string</td>
<td>The customer called you or your agent, for example, Technical Support.</td>
</tr>
<tr>
<td>fromCustomer-ToOther</td>
<td>string</td>
<td>The customer called someone other than you.</td>
</tr>
<tr>
<td>fromMerchant-ToCustomer</td>
<td>string</td>
<td>You called the customer.</td>
</tr>
<tr>
<td>fromMerchant-ToOther</td>
<td>string</td>
<td>You called someone other than the customer.</td>
</tr>
<tr>
<td>fromOtherTo-Customer</td>
<td>string</td>
<td>Someone other than you called the customer.</td>
</tr>
<tr>
<td>fromOtherTo-Merchant</td>
<td>string</td>
<td>Someone other than the customer called you.</td>
</tr>
</tbody>
</table>
ActivityCancelInitType Subobject

A list of known types if initiators for cancellation activities.

Table 2-3  ActivityCancelInitType Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chargeback</td>
<td>string</td>
<td>The service was cancelled due to a chargeback.</td>
</tr>
<tr>
<td>Customer</td>
<td>string</td>
<td>The customer initiated the cancellation.</td>
</tr>
<tr>
<td>Merchant</td>
<td>string</td>
<td>You initiated the cancellation.</td>
</tr>
</tbody>
</table>

ActivityCancellation Subobject

Supplies information about a customer cancellation.

Table 2-4  ActivityCancellation Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>confirmation-Code</td>
<td>int</td>
<td>The confirmation code for the cancellation.</td>
</tr>
<tr>
<td>initiator</td>
<td>ActivityCancelInitType</td>
<td>The type of initiator for cancellation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See the ActivityCancelInitType Subobject.</td>
</tr>
<tr>
<td>reason</td>
<td>string</td>
<td>The reason for cancellation.</td>
</tr>
</tbody>
</table>

ActivityEmailContact Subobject

Supplies information about an email contact with a customer.

Table 2-5  ActivityEmailContact Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destEmail</td>
<td>string</td>
<td>The recipient's email address.</td>
</tr>
<tr>
<td>note</td>
<td>string</td>
<td>A note on the content of the email message.</td>
</tr>
<tr>
<td>srcEmail</td>
<td>string</td>
<td>The sender's email address.</td>
</tr>
</tbody>
</table>
ActivityFulfillment Subobject

Supplies information about physical fulfillment of an order.

Table 2-6 ActivityFulfillment Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delivered</td>
<td>Boolean</td>
<td>A Boolean flag that, if set to true, indicates that the merchandise delivery is complete.</td>
</tr>
<tr>
<td>merchantTrans-</td>
<td>string</td>
<td>Your unique identifier for the transaction.</td>
</tr>
<tr>
<td>actionId</td>
<td></td>
<td><strong>Note:</strong> use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
<tr>
<td>receiptName</td>
<td>string</td>
<td>The recipient’s name as reported by the shipping agent.</td>
</tr>
<tr>
<td>receivedTs</td>
<td>dateTime</td>
<td>A time stamp that corresponds to the date and time reported by the shipping agent of when the merchandise delivery was completed.</td>
</tr>
<tr>
<td>shipper</td>
<td>string</td>
<td>The identifier of the shipping agent (such as UPS or FedEx) if any.</td>
</tr>
<tr>
<td>shippingAddress</td>
<td>Address</td>
<td>The shipping address for the product. This data member encapsulates the customer’s mailing address, billing address, or both. See Section 3.1: Address Data Members.</td>
</tr>
<tr>
<td>trackingString</td>
<td>string</td>
<td>The tracking information on the physical package.</td>
</tr>
</tbody>
</table>

ActivityLogin Subobject

Supplies information about an Account login.

Table 2-7 ActivityLogin Object Data Member

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip</td>
<td>string</td>
<td>The IP address from which a login originated.</td>
</tr>
</tbody>
</table>

ActivityLogout Subobject

Supplies information about an Account logout.

Table 2-8 ActivityLogout Object Data Member

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip</td>
<td>string</td>
<td>The IP address from which a logout originated. Set ip to null if the logout is implicit due to, for example, a server timeout.</td>
</tr>
</tbody>
</table>
**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>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The Activity name. For example, if you sell music online, set the value to musicDownload.</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td>The Activity type. For example, if you sell different types of music online, specify in this field the type, such as Rock.</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>The Activity value. For example, fill in this field with the name of the artist or song for the download.</td>
</tr>
</tbody>
</table>

**ActivityNote Subobject**

Supplies a note or memo.

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>note</td>
<td>string</td>
<td>Notes (maximum of 1,024 characters) on the Activity object.</td>
</tr>
</tbody>
</table>
**ActivityPhoneContact Subobject**

Supplies information about a phone contact with a customer.

Table 2-11  ActivityPhoneContact Object Data Members

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

Describes a list of known types of Activities.

**Table 2-12 ActivityType Object Data Members**

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancellation</td>
<td>string</td>
<td>A cancellation of a product or service offered by you.</td>
</tr>
<tr>
<td>Email</td>
<td>string</td>
<td>An email interaction related to the account.</td>
</tr>
<tr>
<td>Fulfillment</td>
<td>string</td>
<td>An order fulfillment.</td>
</tr>
<tr>
<td>Login</td>
<td>string</td>
<td>A customer login on your site.</td>
</tr>
<tr>
<td>Logout</td>
<td>string</td>
<td>A customer logout from your site.</td>
</tr>
<tr>
<td>NamedValue</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>Note</td>
<td>string</td>
<td>An optional memo regarding the activity.</td>
</tr>
<tr>
<td>Phone</td>
<td>string</td>
<td>A phone interaction related to the account.</td>
</tr>
<tr>
<td>URIView</td>
<td>string</td>
<td>A viewing of a particular Web resource.</td>
</tr>
<tr>
<td>Usage</td>
<td>string</td>
<td>The amount of use of the resources provided by you, such as electronic downloads, or website access.</td>
</tr>
</tbody>
</table>
**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>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancellation-</td>
<td>ActivityCancellation</td>
<td>The customer’s cancellation of a service or product. See the ActivityCancellation Subobject.</td>
</tr>
<tr>
<td>emailArgs</td>
<td>ActivityEmail-</td>
<td>An email event. See the ActivityEmailContact Subobject.</td>
</tr>
<tr>
<td>fulfillment</td>
<td>ActivityFulfillment</td>
<td>The status of your fulfillment of a customer order. See the ActivityFulfillment Subobject.</td>
</tr>
<tr>
<td>loginArgs</td>
<td>ActivityLogin</td>
<td>The IP address from which a login originated. See the ActivityLogin Subobject.</td>
</tr>
<tr>
<td>logoutArgs</td>
<td>ActivityLogout</td>
<td>The IP address from which a logout originated. See the ActivityLogout Subobject.</td>
</tr>
<tr>
<td>namedValueArgs</td>
<td>ActivityNamed-Value</td>
<td>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. See the ActivityNamedValue Subobject.</td>
</tr>
<tr>
<td>noteArgs</td>
<td>ActivityNote</td>
<td>An optional memo on the Activity object. See the ActivityNote Subobject.</td>
</tr>
<tr>
<td>phoneArgs</td>
<td>ActivityPhone-</td>
<td>A phone contact that relates to the Activity object. See the ActivityPhoneContact Subobject.</td>
</tr>
<tr>
<td>uriviewArgs</td>
<td>ActivityURIView</td>
<td>A customer’s visit to a Web page, and possible download activity. See the ActivityURIView Subobject.</td>
</tr>
<tr>
<td>usageArgs</td>
<td>ActivityUsage</td>
<td>The amount of use of a resource (such as the number of downloads) you provide to the customer. See the ActivityUsage Subobject.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytesTransferred</td>
<td>int</td>
<td>The number of bytes actually transferred to the customer.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>A description of the bytes transferred.</td>
</tr>
<tr>
<td>ip</td>
<td>string</td>
<td>The IP address to which the data was transferred.</td>
</tr>
<tr>
<td>size</td>
<td>int</td>
<td>The size of the download.</td>
</tr>
<tr>
<td>transferTime</td>
<td>int</td>
<td>The length of the data transfer in seconds.</td>
</tr>
<tr>
<td>uri</td>
<td>string</td>
<td>Required. The URI of the page.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>string</td>
<td>The amount of use.</td>
</tr>
<tr>
<td>lastDay</td>
<td>int</td>
<td>The amount of use on the last day.</td>
</tr>
<tr>
<td>lastMonth</td>
<td>int</td>
<td>The amount of use in the last month.</td>
</tr>
<tr>
<td>lastUsageDate</td>
<td>dateTime</td>
<td>The last date of use.</td>
</tr>
<tr>
<td>lastWeek</td>
<td>int</td>
<td>The amount of use in the last week.</td>
</tr>
<tr>
<td>lastYear</td>
<td>int</td>
<td>The amount of use in the last year.</td>
</tr>
<tr>
<td>total</td>
<td>int</td>
<td>The duration of use.</td>
</tr>
</tbody>
</table>
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.

**Note:** use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.

**Input**

- **srdf:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the **srdf**. A null **srdf** returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unknown activity type <strong>input-type</strong>. Must be one of <strong>list-of-allowed-types</strong>.</td>
</tr>
<tr>
<td></td>
<td>• Required field 'timestamp' missing!</td>
</tr>
<tr>
<td></td>
<td>• Required field 'account' missing!</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr1</td>
<td>string</td>
<td>The first address line.</td>
</tr>
<tr>
<td>addr2</td>
<td>string</td>
<td>The second, auxiliary address line.</td>
</tr>
<tr>
<td>addr3</td>
<td>string</td>
<td>The third, auxiliary address line.</td>
</tr>
<tr>
<td>city</td>
<td>string</td>
<td>The city of the customer’s address.</td>
</tr>
<tr>
<td>country</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>county</td>
<td>string</td>
<td>The county of the customer’s address if known.</td>
</tr>
<tr>
<td>district</td>
<td>string</td>
<td>The state, province, or district of the customer’s address.</td>
</tr>
<tr>
<td>fax</td>
<td>string</td>
<td>The customer’s fax number.</td>
</tr>
<tr>
<td>latitude</td>
<td>decimal</td>
<td>The customer’s latitude as a signed decimal. In some cases, Vindicia fills in this field.</td>
</tr>
<tr>
<td>longitude</td>
<td>decimal</td>
<td>The customer’s longitude as a signed decimal. In some cases, Vindicia fills in this field.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>The customer’s full name. (256 character limit.)</td>
</tr>
<tr>
<td>phone</td>
<td>string</td>
<td>The customer’s preferred phone number.</td>
</tr>
<tr>
<td>postalCode</td>
<td>string</td>
<td>The postal code of the customer’s address. Note: Your payment processor may limit this field to 9 characters.</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetchByVid</td>
<td>Returns an Address object whose VID matches the input.</td>
</tr>
<tr>
<td>update</td>
<td>Creates or updates an Address object.</td>
</tr>
</tbody>
</table>
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

srld: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

vid: the Address object’s Vindicia identifier, which serves as the search criterion.

Output

returnl: 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• No addresses match VID input-vid.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load VID input-vid: error-description.</td>
</tr>
<tr>
<td></td>
<td>• Missing required parameter vid.</td>
</tr>
</tbody>
</table>

Example

```
$accountVid = 'MyVindiciaAccountVID';

// Create a SOAP caller object
$addr = new Address();
$addrVID = "14e1dce6f48e901464fce22145982a596422a9f4";

// 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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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**

```php
// 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";
}
```

© 2019 Vindicia, Inc.  Table of Contents  The Address Object  96
4 The AutoBill Object

The AutoBill object defines the relationship between an Account object (the customer description), an AutoBillItem object (products and services purchased), and a BillingPlan object (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:

| Note: | Use of the forward slash character (/) in identifiers is not allowed. See Merchant Identifiers for more information. |
4.1 *AutoBill Data Members*

The following table describes the data members of the *AutoBill* object.

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>Account</td>
<td><strong>Required.</strong> The <code>Account</code> object to which this <code>AutoBill</code> object applies. If you do not specify a valid VID or <code>merchantAccountId</code>, CashBox creates a new <code>Account</code> object. See <a href="#">Account Data Members</a> for details. <strong>Note:</strong> use of the forward slash character (/) in merchant identifiers is not allowed. See <a href="#">Merchant Identifiers</a> for more information.</td>
</tr>
<tr>
<td>billingDay</td>
<td>int</td>
<td>The day of the month on which to bill the customer, which, if unspecified, defaults to the day of the . 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. This attribute is useful if <code>AutoBill</code> 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. <strong>Note:</strong> If the Billing Plan for the <code>AutoBill</code> 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.</td>
</tr>
<tr>
<td>billingOffset</td>
<td>int</td>
<td>Number days before or after the scheduled billing day that you want the <code>AutoBill</code> to bill. Use <code>billingRule</code> to set the direction of the offset.</td>
</tr>
<tr>
<td>billingPlan</td>
<td>BillingPlan</td>
<td>The billing plan to be used for this <code>AutoBill</code> object. This attribute determines the frequency and amount of periodic billing transactions generated by this <code>AutoBill</code> object. If you do not specify this attribute, CashBox uses the default billing plan associated with the primary <code>Product</code> object in this <code>AutoBill</code> object. If you have not defined a default billing plan for <code>Product</code>, be sure to specify it here. If the <code>BillingPlan</code> object already exists, simply populate it with its VID or <code>merchantBillingPlanId</code>. If the <code>BillingPlan</code> does not yet exist, CashBox creates a new <code>BillingPlan</code> object along with this <code>AutoBill</code> object. See <a href="#">BillingPlan Data Members</a>.</td>
</tr>
<tr>
<td>billingPlanCamp-</td>
<td>string</td>
<td>The Campaign code redeemed on this <code>AutoBill</code> against the price of the Billing Plan. To apply a Campaign, use this field to pass in a valid Coupon or Promotion code. <strong>Note:</strong> This data member will not be returned.</td>
</tr>
<tr>
<td>aignCode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>billingPlanCampa-</td>
<td>string</td>
<td><strong>Read only.</strong> The unique identifier for a Campaign applied to this <code>AutoBill</code>'s <code>BillingPlan</code>. This is a read-only field returned by CashBox for informational purposes. Values sent in with a SOAP call will be ignored.</td>
</tr>
<tr>
<td>aignId</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Members</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>billingRule</td>
<td>BillingRule</td>
<td>An enumerated string value (Advance or Arrears), informing CashBox in which direction from the scheduled billing date to select the specified billing date. If no specified billing day is provided, this parameter is ignored.</td>
</tr>
<tr>
<td>billingState</td>
<td>BillingState</td>
<td><em>(Required)</em> The billingState data member presents the financial billing condition of an AutoBill. This data member expands on the current billing condition of the subscription, and in combination with the Status field, provides a definitive representation of the current condition of an AutoBill. See the AutoBill billingState Data Member Detail section for descriptions of the enumerated values of the billingState data member.</td>
</tr>
<tr>
<td>billingStatementIdentifier</td>
<td>string</td>
<td>The identifier on a customer’s billing statement when the customer is charged for this AutoBill object. 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.</td>
</tr>
<tr>
<td>cancelFee</td>
<td>AutoBillCancelFee</td>
<td>The early termination fee as defined in the Billing Plan that was used when the subscription (AutoBill) was created. This cancel fee will remain on the AutoBill, even if you subsequently change the Billing Plan. See AutoBillCancelFee Subobject.</td>
</tr>
<tr>
<td>CancelReason</td>
<td>string</td>
<td>A subobject containing a cancel reason code and description for canceling the AutoBill. See the CancelReason Subobject.</td>
</tr>
<tr>
<td>commitmentDate</td>
<td>string</td>
<td>Read-only field indicating the date through which the customer is contractually obligated.</td>
</tr>
<tr>
<td>credit</td>
<td>Credit</td>
<td>This data member encapsulates credit available to the AutoBill. Token-based credits stored in this attribute can be applied toward Transactions generated by this AutoBill for Billing Plans which are defined with a Payment Method of the same Token Type. 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. 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. 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. See the Credit Subobject.</td>
</tr>
<tr>
<td>currency</td>
<td>string</td>
<td>The ISO 4217 currency code (see <a href="http://www.xe.com/iso4217.htm">www.xe.com/iso4217.htm</a>) for this AutoBill object. The default is USD. You should not change the currency type of an AutoBill object via the addCharge, makePayment, or update methods.</td>
</tr>
</tbody>
</table>
Table 4-1 AutoBill Object Data Members  (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customerAutoBillName</td>
<td>string</td>
<td>Optional. A name you specify (on your customer’s behalf) for this AutoBill object, such as 'Home Subscription.'</td>
</tr>
<tr>
<td>endTimestamp</td>
<td>dateTime</td>
<td>This is a read-only attribute in fetched AutoBill objects. CashBox will automatically set this timeStamp based on the AutoBill's last successful billing date, plus the length of the next Billing Period, and any grace period you have defined. This value is reset with every successful billing. <strong>Note:</strong> Do not set this value when creating or updating an AutoBill.</td>
</tr>
<tr>
<td>invoiceTerms</td>
<td>int</td>
<td>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.</td>
</tr>
<tr>
<td>items</td>
<td>AutoBillItem</td>
<td>An array of AutoBillItems to be included in the AutoBill. See the AutoBillItem Subobject.</td>
</tr>
<tr>
<td>mandate</td>
<td>Mandate</td>
<td>The mandate authorizing this subscription. See the Mandate Subobject subobject for details.</td>
</tr>
<tr>
<td>merchantAffiliateId</td>
<td>string</td>
<td>Your ID (a free-form string of 128 characters or less) for the affiliate that submitted this AutoBill object, if any. Maximum length: 256 characters. <strong>Note:</strong> Use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
<tr>
<td>merchantAffiliateSubId</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>merchantAutoBillId</td>
<td>string</td>
<td>Your unique identifier for this AutoBill object. <strong>Note:</strong> Use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
<tr>
<td>minimumCommitment</td>
<td>int</td>
<td>The number of billing cycles the customer is contractually obligated to complete before canceling 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. When you make a call to cancel an AutoBill object, CashBox checks this attribute. If the AutoBill has not completed the minimum commitment period, an AutoBill.cancel() call will succeed only if the force parameter is set to true. <strong>Note:</strong> Previous to CashBox release 25.0, it was possible to pass the minimumCommitment value when creating an AutoBill. Attempting to pass this parameter in the AutoBill in release 25.0 and higher will return an error. You now set this value in the Billing Plan. The minimum commitment is calculated from the BillingPlan attached to the subscription at the time of AutoBill creation. See BillingPlan Data Members.</td>
</tr>
</tbody>
</table>
Table 4-1 AutoBill Object Data Members  (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| nameValues     | NameValuePair | An (optional) array of name–value pair items for this AutoBill object. Some names are reserved for specific purposes. 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. CashBox provides the following name-value pairs for use with SEPA European Direct Debit and UK Direct Debit payment methods:  
  • Use name vin:MandateFlag and value 1 to associate the SEPA or UK Direct Debit Payment Method with the AutoBill.  
  • Use name vin:MandateVersion and value 1.0.1, to associate a mandate document of version 1.0.1 with the object.  
  • Use name vin:MandateID to pass the Mandate ID field of the SEPA or UK Direct Debit Extension record to Chase Paymentech.  
  • Use name vin:MandateApprovalDate to pass the Signature Date field of the SEPA or UK Direct Debit Extension Record to Chase Paymentech.  
  • Use vin:ValidateFullAmt to pass whether full amount authorization needs to be done before charging. This must be enabled if the processor is Global Collect.  
  
  Note: All name-value pairs included with an AutoBill object will be automatically copied to any resultant Transactions.  
  Note also that paypal:SuppressItems, if set to 1, suppresses display of line item details of a purchase during a PayPal ExpressCheckout request.  
  To prevent showing item detail and pricing, provide the name-value pairs: paypal:SuppressItems and paypal:NoCommitUserAction, both with a value of 1, in AutoBill or Transaction objects.  
  See Section 10: The NameValuePair Object. |
| nextBilling    | Transaction   | An object of type Transaction that represents the next projected billing for this AutoBill, if any.  
  See Section 18.1: Transaction Data Members. |
| paymentMethod  | PaymentMethod | Vindicia 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. If you do not specify this attribute, the AutoBill will automatically use the preferred PaymentMethod object associated with the Account.  
  See 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. |
### Table 4-1 AutoBill Object Data Members (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>specifiedBillingDay</code></td>
<td>int</td>
<td>An integer (1-31) specifying the fixed day of the month on which to bill (values 29-31 will automatically work as the last day of the month in calendar months that do not have 29, 30, or 31 days). Providing no value or null in this parameter instructs CashBox to bill on the service period start date as usual.</td>
</tr>
<tr>
<td><code>startTimestamp</code></td>
<td>dateTime</td>
<td>A time stamp that specifies the start date and time for this AutoBill object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If unspecified, the value defaults to today and the current time.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If unspecified, and the <code>unknownStart</code> flag is set to <code>True</code>, the time stamp value will be empty.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See <code>unknownStart</code> in the AutoBill update method.</td>
</tr>
<tr>
<td><code>statementFormat</code></td>
<td>StatementFormat</td>
<td>Defines the billing format used to send statements to a customer. Defaults to DoNotSend if not specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid input:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DoNotSend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Inline</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Attachment</td>
</tr>
<tr>
<td><code>statementOffset</code></td>
<td>int</td>
<td>Days prior to billing that a statement will be sent. This value must be “null” or “0” if the AutoBill's <code>PaymentMethodType</code> is <code>MerchantAcceptedPayment</code>. For conventionally-funded AutoBills, this value must be less than the prebilling notification days (if specified). The value will be ignored if <code>statementFormat</code> is <code>DoNotSend</code>.</td>
</tr>
<tr>
<td><code>statementTemplateId</code></td>
<td>string</td>
<td>Your identifier for a pre-defined statement template. If this value is <code>null</code> (or does not match any pre-defined statement templates), the CashBox-default template will be used.</td>
</tr>
<tr>
<td><code>status</code></td>
<td>AutoBillStatus</td>
<td>Provides a clear and mutually exclusive indicator of the current service status of an AutoBill, and of its place in its life cycle. See AutoBill status Data Member Detail for descriptions of the enumerated values of the <code>status</code> data member.</td>
</tr>
<tr>
<td><code>subscriptionBalance</code></td>
<td>decimal</td>
<td>Current balance (receivable) owed, in the currency specified, on this AutoBill.</td>
</tr>
<tr>
<td><code>VID</code></td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td><code>warnOnExpiration</code></td>
<td>Boolean</td>
<td>A flag that, if set to <code>true</code>, specifies that the customer be warned by email of an upcoming expiration of a trial period or subscription. The default is <code>false</code>. For more information, see the <code>expireWarningDays</code> attribute in the BillingPlanPeriod object.</td>
</tr>
</tbody>
</table>
AutoBill billingState Data Member Detail

The billingState data member presents the financial billing condition of an AutoBill. This data member expands on the current billing condition of the subscription, and in combination with the Status field, provides a definitive representation of the current condition of an AutoBill.

This section describes the enumerated values of the billingState data member of the AutoBill object.

Table 4-2  billingState Values

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown</td>
<td>For internal use or test purposes.</td>
</tr>
<tr>
<td>Unbilled</td>
<td>No billing has yet occurred for this AutoBill.</td>
</tr>
<tr>
<td>Good Standing</td>
<td>The AutoBill is in a billable state and has no overdue balance.</td>
</tr>
<tr>
<td>Free/Trial</td>
<td>The AutoBill's current Billing Period definition (as specified by the current Billing Plan) is forcibly free. This can be a temporary state or for the duration of the billing period configured to be free—or permanent if the Billing Plan is permanently free. This state does not apply when the free condition is arrived at due to a Campaign application.</td>
</tr>
<tr>
<td>In Retry</td>
<td>The AutoBill is still active but in retry, and has at least one billing period in which there was a failed attempt to charge; the AutoBill is still within the configured retry period and may yet be saved by automated processes or customer action. This state corresponds to the prior Soft Error status. In the new scheme, it will only be paired with the Active status.</td>
</tr>
<tr>
<td>Unusable Payment Method</td>
<td>The AutoBill is still active but is in retry, and has at least one billing period in which there was a failed attempt to charge, and received an error code designated as a Hard Fail, which is expected to be unrecoverable. The AutoBill will remain in this state, in the expectation that a new payment method will be provided by the customer, until the specified retry, or grace period, expires. This state corresponds to the previous Hard Error status when the AutoBill end date has not passed</td>
</tr>
<tr>
<td>Billing Completed</td>
<td>The AutoBill is not active; it successfully completed all required billing and has no further scheduled billings.</td>
</tr>
<tr>
<td>Failed To Collect</td>
<td>The AutoBill is no longer active, having exhausted the retry period without a successful payment method update. This state corresponds to the previous Hard Error status, in which the AutoBill end date has passed. This state will only exist for an AutoBill with the status: Canceled.</td>
</tr>
<tr>
<td>Overdue</td>
<td>The AutoBill is assigned a Merchant Accepted Payment (MAP) payment method (invoicing) and has at least one overdue invoice.</td>
</tr>
</tbody>
</table>
The AutoBill uses a payment method that requires merchant or
customer action before charges can be processed, and is wait-
ing on that external action or approval.

The AutoBill is currently in a free or $0 billing period for any pe-
riod excepting a Free=True condition on the initial billing peri-
od only (which uses the Free/Trial state).

The AutoBill has been forced into a terminal state by system or
merchant action. No further billing is planned.

For internal use or test purposes.

For internal use or test purposes.

The AutoBill has an unpaid balance and is in the retry period
but has passed the last scheduled retry attempt—the only path
to recovery is a successful payment method update (usually
customer intervention).

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pending Acceptance</td>
<td>The AutoBill uses a payment method that requires merchant or customer action before charges can be processed, and is waiting on that external action or approval.</td>
</tr>
<tr>
<td>Free Period</td>
<td>The AutoBill is currently in a free or $0 billing period for any period excepting a Free=True condition on the initial billing period only (which uses the Free/Trial state).</td>
</tr>
<tr>
<td>Terminated</td>
<td>The AutoBill has been forced into a terminal state by system or merchant action. No further billing is planned.</td>
</tr>
<tr>
<td>Legacy Unknown Billing-State</td>
<td>For internal use or test purposes.</td>
</tr>
<tr>
<td>Bogus BillingState</td>
<td>For internal use or test purposes.</td>
</tr>
<tr>
<td>Grace Period</td>
<td>The AutoBill has an unpaid balance and is in the retry period but has passed the last scheduled retry attempt—the only path to recovery is a successful payment method update (usually customer intervention).</td>
</tr>
</tbody>
</table>
AutoBill status Data Member Detail

AutoBill status provides a clear and mutually exclusive indicator of the current service status of an AutoBill, and of its place in its life cycle.

This section describes the enumerated values of the status data member of the AutoBill object. Figure 1: Status Flow Diagram illustrates the path a typical AutoBill follows in its life cycle, as well as special system statuses it can have.

Table 4-3 Status Values

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>The AutoBill is actively granting entitled service and billing in accordance with the assigned Billing Plan and associated AutoBill Items.</td>
</tr>
<tr>
<td></td>
<td>See Figure 1: Status Flow Diagram, for an illustration of Active AutoBill states.</td>
</tr>
<tr>
<td>Bogus AutoBill Status</td>
<td>For internal use or test purposes.</td>
</tr>
<tr>
<td>Canceled</td>
<td>The AutoBill is no longer actively granting service. It has been canceled due to user, merchant, or system action—for example, the customer has opted out of recurring billing or canceled the service; you have called cancel on the AutoBill object or stopAutoBilling on the Account object; or if a charge-back is received against a transaction generated by the AutoBill object.</td>
</tr>
<tr>
<td></td>
<td>Canceled is a terminal and permanent status. The Billing condition of this AutoBill is clarified by the current Billing State. Note that the customer who owns the AutoBill object is entitled until the AutoBill’s end-date.</td>
</tr>
<tr>
<td></td>
<td>(For for a flexible and automated way to indicate why an AutoBill is canceled, see the CancelReason Subobject. See also discussions of Cancel Reason Codes in the CashBox Programmer’s Guide and the CashBox User Guide.)</td>
</tr>
<tr>
<td>Deleted</td>
<td>The AutoBill was administratively removed—the content of this status can only be displayed in custom and specialty reports.</td>
</tr>
<tr>
<td>Dryrun</td>
<td>The AutoBill was temporarily created in service of a dry run—it will not be available beyond the duration of the dry run.</td>
</tr>
<tr>
<td>Expired</td>
<td>The AutoBill is no longer actively granting service—its fixed or non-recurring plan has ended and no further billing is scheduled or possible.</td>
</tr>
<tr>
<td>Legacy Suspended</td>
<td>This status is used in limited scenarios for legacy AutoBills that ended with a Hard Fail status but which did not otherwise meet the conditions of the new status values.</td>
</tr>
<tr>
<td>Pending Activation</td>
<td>The AutoBill has been fully created, has a start date that is in the future, and has not yet granted entitled service and is not yet billing. The AutoBill start date can still be changed.</td>
</tr>
</tbody>
</table>
## Status Values

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pending Cancel</td>
<td>The AutoBill is Active but scheduled for cancellation. The AutoBill will leave active service at the current end date.</td>
</tr>
<tr>
<td>Processing</td>
<td>The AutoBill is newly created and still being administratively defined. It is not currently providing entitled service and is not yet billing. Typically this status will be Pending Activation: the AutoBill has been completely created but has a future start date. The AutoBill is not yet granting entitled service and is not yet billing. The AutoBill start date can still be changed.</td>
</tr>
<tr>
<td>Unknown</td>
<td>For internal use or test purposes.</td>
</tr>
<tr>
<td>Upgraded</td>
<td>A terminal status given to an older version of an AutoBill that has been explicitly transitioned using the <code>AutoBill.upgrade()</code> API method.</td>
</tr>
</tbody>
</table>

![Status Flow Diagram](image)

**Figure 1:** Status Flow Diagram
4.2 AutoBill Subobjects

The AutoBill object has the following subobjects:

- **AutoBillItem Subobject**
- **Mandate Subobject**
- **PaymentMethod Subobject**
- **CancelReason Subobject**
- **AutoBillCancelFee 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>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addedDate</td>
<td>dateTime</td>
<td>Read only. Specifies the dateTime when the AutoBillItem was added to the AutoBill.</td>
</tr>
<tr>
<td>amount</td>
<td>decimal</td>
<td>The amount to bill. If non-null, this field will override the Product’s price. Value cannot be negative. This field is populated only if you wish to override the default (product-based) price for the item. Otherwise, it is blank. <strong>Note:</strong> AutoBillItems may have an amount, or a ratePlan, but not both.</td>
</tr>
<tr>
<td>campaignCode</td>
<td>string</td>
<td>The Campaign code redeemed on this AutoBillItem. To apply a Campaign, use this field to pass in a valid Coupon or Promotion code. <strong>Note:</strong> This data member will not be returned.</td>
</tr>
<tr>
<td>campaignId</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>currency</td>
<td>string</td>
<td>The ISO 4217 currency code to be used for the override amount. This value will be ignored if amount is null.</td>
</tr>
<tr>
<td>cycles</td>
<td>int</td>
<td>The number of billing cycles this item will be active. If null, the item will remain active until explicitly removed.</td>
</tr>
<tr>
<td>cyclesRemain-</td>
<td>int</td>
<td>A read-only field indicating how many billing cycles remain for this item.</td>
</tr>
<tr>
<td>- ing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Members</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>index</td>
<td>int</td>
<td>The index number of the item in the <code>items</code> field of an <code>AutoBill</code>. <em>(Should be unique in array. First item should have index 0.)</em></td>
</tr>
<tr>
<td>merchantAutoBillItemId</td>
<td>string</td>
<td>Your unique identifier for this <code>AutoBillItem</code> object. <strong>Note:</strong> use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
<tr>
<td>nameValues</td>
<td>NameValuePair</td>
<td>Optional. An array of name–value pair items for this <code>AutoBill</code> object. <strong>Note:</strong> All name-value pairs included with an <code>AutoBill</code> object will be automatically copied to any resultant transactions, and similarly <code>AutoBill Item</code> <code>nameValues</code> will be inherited by their corresponding transaction items. See The NameValuePair Object</td>
</tr>
<tr>
<td>product</td>
<td>Product</td>
<td>The Product to be AutoBilled. When creating a new <code>AutoBillItem</code>, 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. See Product Data Members.</td>
</tr>
<tr>
<td>quantity</td>
<td>string</td>
<td>Number of items, or the number of units of a product when the product can be purchased in multiples. CashBox uses this number to determine the cost of the item to the customer. *Quantity must be a whole number unless the item is a rated item. For license-based rated products, *Quantity indicates the license count.</td>
</tr>
<tr>
<td>ratePlan</td>
<td>RatePlan</td>
<td>The Rate Plan associated with this Item. <strong>Note:</strong> <code>AutoBill Items</code> may have an amount, or a <code>ratePlan</code>, but not both. See RatePlan Data Members.</td>
</tr>
<tr>
<td>removedDate</td>
<td>dateTime</td>
<td>A read-only attribute indicating the time this item was removed.</td>
</tr>
</tbody>
</table>
Mandate Subobject

The **Mandate** object authorizes Direct Debit charges.

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| startDate                  | string    | Specifies when the AutoBill will begin billing for the AutoBillItem, and when the item’s entitlements will become Active. Valid input includes null (for which the startDate will be today), yyyy-mm-dd, or a time interval. Acceptable time interval strings include an integer followed by:
  - Season
  - Month
  - Week
  - Day
  For example:
  - 1 Season
  - 2 Month
  - 3 Week
  - 1 Day
  Appending an "s" is not required for time intervals greater than 1; however, it is acceptable.
  The output format is yyyy-mm-dd. |
| token                      | Token     | The token associated with amount (if this is a Token-based AutoBill). This value will be ignored if amount is null.                               |
| transitioned-FromAutoBillItemVid | string    | Read only. The unique Vindicia identifier of the AutoBillItem this item replaced as the result of an AutoBill.modify call.                         |
| transitioned-FromMerchantAutoBillItemId | string    | Read only. Your identifier for the AutoBillItem this item replaced as the result of an AutoBill.modify call.                                       |
| transitioned-ToAutoBillItemVid | string    | Read only. The unique Vindicia identifier of the AutoBillItem that replaced this item as a result of an AutoBill.modify call.                     |
| transitioned-ToMerchantAutoBillItemId | string    | Read only. Your identifier for the AutoBillItem that replaced this item as a result of an AutoBill.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 populated by CashBox. |
Data members of the Mandate object.

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>created</td>
<td>xsd:dateTime</td>
<td>Date and time this Mandate was created.</td>
</tr>
<tr>
<td>identifier</td>
<td>xsd:string</td>
<td>Unique reference for this Mandate.</td>
</tr>
<tr>
<td>status</td>
<td>Base Type xsd:string</td>
<td>The current status for this Mandate (Pending, Active, Failed, Canceled, Expired).</td>
</tr>
</tbody>
</table>

PaymentMethod Subobject

See The PaymentMethod Object.

CancelReason Subobject

Use the CancelReason object to specify a reason for canceling an AutoBill.

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

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason_code</td>
<td>xsd:string</td>
<td>The reason code assigned for the cancellation. Can be 15 characters or fewer.</td>
</tr>
<tr>
<td>description</td>
<td>xsd:string</td>
<td>A description of the reason for the cancellation. Can be up to 255 characters.</td>
</tr>
</tbody>
</table>

AutoBillCancelFee Subobject

The CancelFee object to specify the amount and currency type of a cancel fee for early termination of the AutoBill. This fee is set on the Billing Plan when the AutoBill is created.

The following table describes the data members of the AutoBillCancelFee object.

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>xsd:decimal</td>
<td>The amount of the fee. Must be positive.</td>
</tr>
<tr>
<td>currency</td>
<td>xsd:string</td>
<td>The ISO 4217 Currency Code of billing. Defaults to USD.</td>
</tr>
</tbody>
</table>
# 4.3 AutoBill Methods

The following table summarizes the methods for the AutoBill object.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activate</td>
<td>Activates an Autobill with a deferred start date.</td>
</tr>
<tr>
<td>addCampaign</td>
<td>Adds a Campaign to an existing AutoBill.</td>
</tr>
<tr>
<td>addCharge</td>
<td>Adds a non-recurring charge to an AutoBill.</td>
</tr>
<tr>
<td>cancel</td>
<td>Cancels an AutoBill object.</td>
</tr>
<tr>
<td>changeBillingDayOfMonth</td>
<td>Updates the monthly billing day.</td>
</tr>
<tr>
<td>delayBillingByDays</td>
<td>Delays the next billing by the specified number of days.</td>
</tr>
<tr>
<td>delayBillingToDate</td>
<td>Delays the next billing until the specified date.</td>
</tr>
<tr>
<td>fetchAllCancelReason</td>
<td>Returns an array of CancelReasons objects.</td>
</tr>
<tr>
<td>fetchAllCreditHistory</td>
<td>Returns credit history for all AutoBills.</td>
</tr>
<tr>
<td>fetchAllInSeason</td>
<td>Returns an array of all in-season AutoBills.</td>
</tr>
<tr>
<td>fetchAllOffSeason</td>
<td>Returns an array of all off-season AutoBills.</td>
</tr>
<tr>
<td>fetchBillingItemHistory</td>
<td>Returns one or more AutoBill objects whose Account object matches the input.</td>
</tr>
<tr>
<td>fetchByAccount</td>
<td>Returns an array of AutoBills by Account object.</td>
</tr>
<tr>
<td>fetchByAccountAndProduct</td>
<td>Returns all AutoBill objects whose Account and Product objects match the input.</td>
</tr>
<tr>
<td>fetchByEmail</td>
<td>Returns one or more AutoBill objects whose email address matches the input.</td>
</tr>
<tr>
<td>fetchByMerchantAutoBillId</td>
<td>Returns an AutoBill object whose ID assigned by you () matches the input.</td>
</tr>
<tr>
<td>fetchByVid</td>
<td>Returns an AutoBill object whose VID matches the input.</td>
</tr>
<tr>
<td>fetchByWebSessionVid</td>
<td>Returns an AutoBill object whose WebSession VID matches the input.</td>
</tr>
<tr>
<td>fetchCreditHistory</td>
<td>Returns an audit log of credit-related events for the specified AutoBill, or for all AutoBills.</td>
</tr>
<tr>
<td>fetchDailyInvoiceBillings</td>
<td>Returns an array of Transaction objects, with MerchantAccepted-Payment Payment Methods, that must be billed for the day.</td>
</tr>
<tr>
<td>fetchDeltaSince</td>
<td>Returns one or more AutoBill objects whose status has changed since the specified time stamp.</td>
</tr>
<tr>
<td>fetchFutureRebills</td>
<td>Returns an array of planned future billing Transactions, that do not yet exist in CashBox, for the specified AutoBill object.</td>
</tr>
<tr>
<td>fetchInvoice</td>
<td>Returns an Invoice for the given invoice ID as plain text or a PDF.</td>
</tr>
</tbody>
</table>
## Table 4-8 AutoBill Object Methods (Continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetchInvoiceNumbers</td>
<td>Fetches the list of invoice numbers of invoices in the given state, for the given AutoBill.</td>
</tr>
<tr>
<td>fetchRemainingPaymentDetails</td>
<td>Returns information on an AutoBill’s remaining payments after the most recent payment.</td>
</tr>
<tr>
<td>fetchUpgradeHistoryMerchantAutoBillId</td>
<td>Returns an AutoBill’s upgrade history, given the merchantAutoBillId.</td>
</tr>
<tr>
<td>fetchUpgradeHistoryByVid</td>
<td>Returns an AutoBill’s upgrade history, given the VID.</td>
</tr>
<tr>
<td>finalizeCarrierBilling</td>
<td>Completes the process of signing up for a subscription using CarrierBilling as a payment method.</td>
</tr>
<tr>
<td>finalizeCustomerAction</td>
<td>Completes processing of a Transaction after the customer finishes payment activities at the payment provider-hosted web pages, and is redirected to your site.</td>
</tr>
<tr>
<td>finalizePayPalAuth</td>
<td>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.</td>
</tr>
<tr>
<td>grantCredit</td>
<td>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.</td>
</tr>
<tr>
<td>makePayment</td>
<td>Enters a payment against an AutoBill.</td>
</tr>
<tr>
<td>migrate</td>
<td>Allows you to import data to CashBox for billing cycles completed through a different system.</td>
</tr>
<tr>
<td>modify</td>
<td>Allows you to change an AutoBill, while maintaining its history.</td>
</tr>
<tr>
<td>redeemGiftCard</td>
<td>Redeems a specified gift card, and adds equivalent credit to the AutoBill.</td>
</tr>
<tr>
<td>reversePayment</td>
<td>Reverses an AutoBill payment made using makePayment. This method may only be used with payments using MerchantAcceptedPayment payment methods.</td>
</tr>
<tr>
<td>revokeCredit</td>
<td>Deducts from credit available to an AutoBill. Time-based credit cannot be revoked.</td>
</tr>
<tr>
<td>settlementQuote</td>
<td>The settlementQuote method generates a quote for the settlement amount that would result in an AutoBill.cancel call made with both the disentitle and settle parameters set to True.</td>
</tr>
<tr>
<td>update</td>
<td>Creates a new AutoBill object, or updates an existing one.</td>
</tr>
<tr>
<td>updateCancelReason</td>
<td>Creates or edits an AutoBill cancelReason.</td>
</tr>
<tr>
<td>writeOffInvoice</td>
<td>Marks an Invoice object as writtenOff, the debt unable to be collected.</td>
</tr>
</tbody>
</table>
activate

Use the `activate` method to start an existing `AutoBill` that was created with a deferred start date (`unknownStart = True`) and is in status: Pending Activation. Calling `activate` sets the `startTimestamp` to `today` and initiates the billing process.

**Input**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`auth`: credentials required to authenticate this request.

`autobill`: the object of type `AutoBill` to activate.

**Output**

`status`: the status of the activated `AutoBill`.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load <code>AutoBill</code>.</td>
</tr>
<tr>
<td></td>
<td>• Unable to activate.</td>
</tr>
</tbody>
</table>

**Example**

```//---unknownStart---
autobill->update( unknownStart => 1 );

autobillId = autobill->id();
.
//once ready to start autobill
autobill->load( autobillId )->activate();

//---knownStart---
nextWeek = Date->nextWeek();

autobill->update( startTimestamp => nextWeek );
autobillId = autobill->id();
.
//if ready prior to startTimestamp, autobill can activate
autobill->load( autobillId )->activate();```
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

srdf: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srdf. A null srdf returns the complete response.

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

product: a Product 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:

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

**Example**

```php
$autobill = new AutoBill();
$autobill->set($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**

**srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**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.)

**taxClassification**: tax classification for the charge. This field is required, except when the price is to be based on the SKU. If the price is based on the SKU, the tax classification can be based on the associated Product tax.

**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.

---

**Note**: Do not change the established currency type.

---

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• AutoBill not found.</td>
</tr>
<tr>
<td></td>
<td>• Failed to load token: error-description.</td>
</tr>
<tr>
<td></td>
<td>• Failed to add charge to AutoBill: error-description.</td>
</tr>
</tbody>
</table>

© 2019 Vindicia, Inc.  
Table of Contents  
The AutoBill Object  116
Example

```php
$autobill = new AutoBill();
$autobill->set($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 canceled 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 can cancel an AutoBill object within the minimum commitment period by enabling the `force` option.

Canceling 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`.

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

**Input**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- **autobill**: AutoBill object to cancel. You can identify this object with either its VID or your AutoBill ID.

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

- **force**: Boolean flag that, when set to `true`, cancels the subscription even if the minimum commitment period for this AutoBill object has not been satisfied. In the case of an AutoBill that has a BillingPlan with a minimumCommitment period, and the subscription is still in the commitment period, the penalty is calculated as defined in the cancelFeePolicy field.

- **settle**: Boolean flag that specifies whether to settle the AutoBill when it is canceled. When set to `true`, an attempt will be made to settle the AutoBill by either refunding the customer for a portion of the prepaid use that will not be available, or by charging for non-recurring-charges that the customer has not yet paid. When set to `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.

- **sendCancellationNotice**: Boolean flag that, when set to `true`, sends a cancellation email. When set to `false`, no cancellation email is sent. If this flag is not specified, the default value is `true`. 
cancelReasonCode: (optional) reason for canceling the AutoBill. You can use predefined CashBox cancel reason codes, or define additional codes using the CashBox Portal or the API. (See Canceling AutoBills with Reason Codes in the CashBox Programming Guide.) Supplying an undefined cancel reason code will result in an error.

cancelFeePolicy: policy for handling the settlement fee. Options for specifying how to cancel the policy include:

- Waive the Cancel Fee—the default.
- Prorate Cancel Fee—calculate fee based on time remaining in the commit period.
- Full Cancel Fee—apply the full early cancellation fee.

Output

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

autobill: the AutoBill object that was canceled.

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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load AutoBill: No match.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load AutoBill: error-description.</td>
</tr>
<tr>
<td></td>
<td>• Error saving AutoBill: error-description.</td>
</tr>
<tr>
<td>403</td>
<td>Minimum commitment not fulfilled for this AutoBill.</td>
</tr>
<tr>
<td>405</td>
<td>Unable to cancel upgraded AutoBill.</td>
</tr>
</tbody>
</table>

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**

*srd*: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the *srd*. A null *srd* returns the complete response.

- **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 ().
- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Date to delay to must be specified and have a length.</td>
</tr>
<tr>
<td></td>
<td>• No AutoBill specified in arguments.</td>
</tr>
<tr>
<td></td>
<td>• Unable to delay billing to date: <code>internal-error</code>.</td>
</tr>
<tr>
<td></td>
<td>• Unable to change billing day of month: <code>error-description</code>.</td>
</tr>
</tbody>
</table>
Example

// to change the billing day of the month

$autobill = new AutoBill();
$autobill->set('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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- `autobill`: the `AutoBill` object whose billing to delay. Identify this object by populating it with its VID or .
- `delayDays`: the number of days by which to delay the billing. (Must be a positive integer.)
- `movePermanently`: when set to `True`, makes the altered billing date change permanent for all subsequent billings.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Days to delay must be a positive integer.</td>
</tr>
<tr>
<td></td>
<td>• Must specify AutoBill to delay billing for.</td>
</tr>
<tr>
<td></td>
<td>• Unable to delay billing to date: <code>error-description</code>.</td>
</tr>
</tbody>
</table>

**Example**

```php
$autobill = new AutoBill();
$autobill->set('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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

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

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

- **movePermanently**: when set to `True`, makes the altered billing date change permanent for all subsequent billings.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Must specify AutoBill to delay billing for.</td>
</tr>
<tr>
<td></td>
<td>• Date to delay to must be specified and have a length.</td>
</tr>
<tr>
<td></td>
<td>• Unable to delay billing to date: <code>internal-error</code>.</td>
</tr>
</tbody>
</table>
Example

// to delay billing to a specified date

$autobill = new AutoBill();
$autobill->set('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";
}
fetchAllCancelReason

The `fetchAllCancelReason` method returns an array of `CancelReasons` objects.

**Input**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

**Output**

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

`cancelReason`: An array of cancel reasons.

**Example**

```php
$response = $autobill->fetchAllCancelReason($autobill);
$codes = $response->{'cancelReasons'};
```
fetchAllCreditHistory

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

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

**Input**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

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

`endTimestamp`: the ending time stamp (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**

```php
$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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- **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**

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

- **srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

- **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**

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

---

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

The `fetchBillingItemHistory` method returns a series of records describing BillingPlan changes that apply to this AutoBill, and a series of records requiring changes to the AutoBillItems (for example, an upgrade accomplished using modify).

**Input**

- **auth**: Credentials required to authenticate this request.
- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

- **autobill**: An AutoBill that must have, at minimum, a merchantAutoBillId or VID to identify it.

**Output**

- **return**: an object of type Return that indicates success or failure of the call.
- **autobill**: a line item for the creation of each AutoBillItem, and subsequent additions to it.
- **autoBillItemHistory**: an array showing the history of AutoBillItem changes for this AutoBill.
  - **action**: Action (Edit, Add, Remove) performed on the AutoBillItem.
  - **actionDate**: Date and time of the change request.
  - **effectiveDate**: Date and time the action on the AutoBill takes effect.
  - **autoBillItem**: The autoBill as it appears after the changes were made.

- **BillingPlanHistory**: an array showing the history of changes to the BillingPlan for this AutoBill.
  - **merchantBillingPlanId**: Identifier for the BillingPlan.
  - **startDate**: Date the BillingPlan was first associated with the AutoBill.
  - **endDate**: Date the BillingPlan was removed from the AutoBill.

**Returns**

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>The request succeeded.</td>
</tr>
</tbody>
</table>
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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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 account.

`page`: (optional) page number, starting at 0, to begin results returned. 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) 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.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Data validation error: <code>error-description</code>.</td>
</tr>
</tbody>
</table>

**Example**

```php
// Create and populate an Account object
$account = new Account();
$account->setMerchantAccountId('abc101');
$page = 0;
$pageSize = 100;

$autobill = new AutoBill();
$response = $autobill->fetchByAccount($account, '', $page, $pageSize);
if($response['returnCode'] == 200) {
    $fetchedAutoBills= $response['data']->autobills;
    foreach ($fetchedAutoBills as $autobill) {
```
// process each autobill found here
print "Found account with id: "
   . $autobill->get() . "\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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Data validation error: error-description.</td>
</tr>
</tbody>
</table>
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->get() . "\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**

**srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Must specify email address to load by!</td>
</tr>
<tr>
<td>404</td>
<td>No AutoBills found for email address input-email-address: No match.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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->get() . "\n";
    }
}
```
fetchByMerchantAutoBillId

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

**Input**

**srds**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srds. A null srds returns the complete response.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• No autobill matches serial number input-merchantAutoBillId.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load autobill by serial number input-merchantAutoBillId: error-description.</td>
</tr>
</tbody>
</table>

**Example**

```php
// 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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
</table>
| 400         | One of the following:  
|             | • Unable to load autobill by VID input-vid: error.  
|             | • Must specify VID to load by! |
| 404         | Unable to load autobill by VID input-vid: No match. |

**Example**

```php
$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**

- **srdf**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Missing required parameter 'vid'.</td>
</tr>
<tr>
<td>404</td>
<td>Unable to find requested AutoBill: No matches.</td>
</tr>
</tbody>
</table>
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 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

sr<code>d</code>: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the s<code>rd</code>. A null s<code>rd</code> returns the complete response.

<code>autobill</code>: the (optional) AutoBill object for which you wish to retrieve credit event history. Use the AutoBill’s VID to identify it. Leave this variable blank if you wish to fetch credit history across all AutoBills.

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

endTimestamp: the ending time stamp (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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
</table>
| 400         | One of the following:  
|             | • Unable to load autobill.  
|             | • Invalid value or values of the time stamp, 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**

`srд`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srд`. A null `srд` returns the complete response.

`startTimeStamp`: the starting time stamp for the range of `Transactions` you wish to retrieve. If set, the fetch will return `Transactions` with a time stamp 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 time stamp for the range of `Transactions` you wish to retrieve. If set, the fetch will return `Transactions` with a time stamp on or before the day of `endTimeStamp`. If not set, the method will return `Transactions` with a time stamp on or before the `startTimeStamp` input.

**Note:** If neither `startTimeStamp` nor `endTimeStamp` are specified, the fetch will return `Transactions` with a time stamp 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);
$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 a time stamp you specify. Call this method to find out which AutoBill objects are still active and which are not. AutoBills can become inactive for several reasons, 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. Returning a large number of records in one call can overflow 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 no additional records.

---

**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 current end-date of the AutoBill object, 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 customer entitlements is by making calls on the Entitlement object.

---

By default, this method returns all AutoBills that have had a Status change within the timestamp parameters you specify. The AutoBillEventType parameter gives you the ability to selectively return AutoBills that have had changes to an array of events you specify. By tailoring returns to your needs you can greatly reduce both the volume of the return and the response time. As with the default return of Status, the CashBox deduping functionality ensures that AutoBills with multiple events are returned only once per query.

**Input**

- `srdf`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

- `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 upper threshold of the time window by which to limit the search. If unspecified, this value defaults to the current time.

---

**Note**: To reduce high loads on the system, the maximum span for a `fetchDeltaSince()` call is 61 days. If you want to fetch an extended amount of data, best practice suggests that you do so via a loop that pulls a day's worth of data at a time, with another loop internally that handles fetching each resultant page for that day.

**AutoBillEventType**: an array of event types you want to be notified of. Event types include the values listed in Table 4-9: Enumerated Values. If unspecified, defaults to `status_change` and `transaction`.

### Table 4-9 Enumerated Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account_updater</td>
<td>Autobills that have had an Account Updater event.</td>
</tr>
<tr>
<td>cancellation</td>
<td>AutoBills that have been canceled.</td>
</tr>
<tr>
<td>catchup_billing</td>
<td>AutoBills that have experienced a Catch-up Billing.</td>
</tr>
<tr>
<td>chargeback</td>
<td>Autobills that have chargebacks received against them.</td>
</tr>
<tr>
<td>credit_operation</td>
<td>AutoBills that have had a credit action (grant/consumption).</td>
</tr>
<tr>
<td>delay_bill</td>
<td>AutoBills whose Billing has been delayed.</td>
</tr>
<tr>
<td>end</td>
<td>AutoBills that have ended.</td>
</tr>
<tr>
<td>end_date_change</td>
<td>AutoBills whose end date has been changed (high volume event).</td>
</tr>
<tr>
<td>expiration</td>
<td>AutoBills that have expired.</td>
</tr>
<tr>
<td>migration</td>
<td>AutoBills that have been migrated.</td>
</tr>
<tr>
<td>modification</td>
<td>AutoBills that have been modified.</td>
</tr>
<tr>
<td>non_recurring_transaction</td>
<td>AutoBills that have had a successful, associated one time Transaction executed.</td>
</tr>
<tr>
<td>notification</td>
<td>AutoBills that have generated an email Notification (high volume event).</td>
</tr>
<tr>
<td>payment_method_removed</td>
<td>AutoBills that have had a payment method removed.</td>
</tr>
<tr>
<td>payment_method_update</td>
<td>AutoBills that have had a payment method updated.</td>
</tr>
<tr>
<td>refund</td>
<td>AutoBills that have had a refund.</td>
</tr>
</tbody>
</table>
### 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Invalid value or values of the time stamp, and/or page, and/or page size.</td>
</tr>
</tbody>
</table>

### Example

```php
$ab = new AutoBill();
$page = 0;
$pageSize = 100;
$startTime = '2016-01-01T22:34:32.265Z';
$endTime = '2016-01-02T22:34:32.265Z';
$AutoBillEventTypes = ['status_change','chargeback'];
$srd = '{"autobills": ["VID",""]}';
do {
    $ret = $ab->fetchDeltaSince($srd, $startTime, $page, $pageSize, $endTime, $AutoBillEventTypes);
    $count = 0;
    if ($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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`autobill`: the `AutoBill` object for which to obtain the future billing transactions. Identify this object with either its VID or .

`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 more information, see Section 18: The Transaction Object.

**Returns**

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Quantity must be a positive integer.</td>
</tr>
<tr>
<td></td>
<td>• No AutoBill specified in arguments.</td>
</tr>
<tr>
<td>404</td>
<td>No matching AutoBill found.</td>
</tr>
<tr>
<td>405</td>
<td>AutoBill is in an inactive state <code>&lt;state&gt;</code>.</td>
</tr>
</tbody>
</table>
**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() . "\n";
        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

srd: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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. (Note that use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.)

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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
</table>
| 400 | One of the following:  
  - Unable to load AutoBill: No match.  
  - Unable to load AutoBill: error-description.  
  - Unable to load TransactionBilling: error-description.  
  - Unable to load StatementTemplate: error-description.  
  - Failed to render invoice: error-description. |
Example

```php
$autobill = new AutoBill();
$autobill->set($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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load AutoBill: No match.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load AutoBill: <code>error-description</code>.</td>
</tr>
</tbody>
</table>

**Example**

```php
$autobill = new AutoBill();
$autobill->set($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**

- `srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- `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`.
  - `autobillItemVid`: the Vindicia unique name (VID) for the `AutoBillItem`.
  - `merchantAutoBillItemId`: the unique identifier for this `AutoBillItem` object.
  - `merchantProductId`: the 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`: the Vindicia unique name (VID) for the `Product`.
  - `remainingBalanceInSet`: the balance remaining in the `AutoBill` for the `AutoBillItem`.
  - `remainingPaymentsInSet`: the number of payments remaining in the `AutoBill` for the `AutoBillItem`.

**Returns**

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

```php
/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**

*srd*: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the *srd*. A null *srd* returns the complete response.

*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 time stamp 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 fetchAutoBillUpgradeHistoryBy to generate a complete list of VIDs for the AutoBill.

Input

**srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**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 time stamp is omitted if the step is current.

Returns

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

Example

```php
$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;
}
```
finalizeCarrierBilling

The finalizeCarrierBilling method completes the process of signing up for a subscription using CarrierBilling as a payment method. This method enables merchants to pass a PIN, sent to customer’s phone, back to the Carrier for validation. Merchants must also pass a transactionVid to finalizeCarrierBilling. The transactionVID is the ID from the initial Transaction object returned by the Autobill.update() call. Use this method only when you are working with an AutoBill that is paid for with a Carrier-based payment method.

**Input**

**srdf:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**transactionVid:** Vindicia-generated ID (VID) for the Transaction. This will be available in the Transaction returned by the Autobill.update() or Transaction.authCapture() call.

**confirmationCode:** Set this to the code the carrier sent to your customer’s phone, or to true if Boku redirected the customer to your Success web page (returnUrl). If Boku redirected the customer to your Failure page, set this to false.

**Output**

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

authStatus: Status of the underlying validation transaction. This also contains responses received from PayPal.

autobill: Resultant AutoBill object. CashBox populates this only if there is no error processing the call.
**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. 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).

---

**Input**

- **srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- **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 validateForFuturePayment 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**

**sr**d: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the sr**d. A null sr**d returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Internal-error-string.</td>
</tr>
</tbody>
</table>

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
$succeed = true;
$response =
    $soap_caller->finalizePayPalAuth($payPalTxId, $success);

if($response['returnCode'] == 200) {
    printLog "PayPal validation transaction successful";
    $updatedAutoBill = $response['data']->autobill;
    printLog "AutoBill id " . $updatedAutoBill->get() . "\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

srq: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srq. A null srq returns the complete response.

autobill: an AutoBill object to which you wish to grant credit. Identify the AutoBill using its 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• AutoBill not found.</td>
</tr>
<tr>
<td></td>
<td>• Failed to translate credit error-description.</td>
</tr>
<tr>
<td></td>
<td>• Failed to grant credit error-description.</td>
</tr>
<tr>
<td></td>
<td>• Failed to save AutoBill after granting credit.</td>
</tr>
<tr>
<td></td>
<td>• Failed to reload AutoBill after granting credit error-description.</td>
</tr>
<tr>
<td></td>
<td>• Time interval credit cannot have amount 0.</td>
</tr>
</tbody>
</table>

Example

```php
$abill = new AutoBill();

// autobill id for an existing subscription
$abill->set('SBCR312345');

// We want to grant 2 days of credit
$time = new TimeInterval();
$time->setDays('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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

  - **autobill**: the `AutoBill` to which this payment applies.
  - **paymentMethod**: the `PaymentMethod` to be used for this payment. This `paymentMethod` can be of type `MerchantAcceptedPayment`, credit card, or PayPal. If you use PayPal, the PayPal payment method must be already populated with a Billing Agreement ID (BAID), acquired either from an earlier `AutoBill` creation or from a one-time transaction conducted for the customer’s account, in which a PayPal reference ID was requested. You can not use an PayPal payment method that requires customer approval at the PayPal site in this call.

    (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. To make a payment against an invoice in Open status, an ID must be specified—CashBox will not direct overage against such a future invoice.

**overageDisposition**: an object of type PaymentOverageDisposition, defines how to allocate payments in excess of a required AutoBill payment amount. Enumerated values include:

- applyToOldestInvoice (the default)
- applyToThisAutoBill
- 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 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
</table>
| 400         | One of the following:  
- Account not found.  
- Failed to translate payment method: **error-description**.  
- Failed to make payment: **error-description**.  
- Transaction not returned from payment attempt.  
- This Credit Card already exists—Policy Violation" (eradicate the newly created but failed one or ensure it is set INACTIVE).  
You receive this error message when you have the Credit_Card_Constraints merchant option enabled on your site. This message means that someone is attempting to use a credit card that is already being used as the payment method on an existing account. This is not allowed.

---

**Example**

```php
$autobill = new AutoBill();
$autobill->set($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 enables you to import existing subscription and transaction information from your current billing system into CashBox. This call creates new AutoBills that reflect the imported information.

You can call `AutoBill.migrate` multiple times for a given `AutoBill`. With the first call, CashBox creates the `AutoBill` and builds the billing schedule records that correspond to any `MigrationTransactions` that are included in the call.

You can also use this method to specify a regular billing in advance of, or after, the Service Period start date. You can allow the service dates to be maintained according to the billing plan, while billing for those periods on specified days of the month, before or after the scheduled service date, using the `specifiedBillingDay` and `billingRule` parameters.

When you call `AutoBill.migrate` on an existing `AutoBill`—that is, when you specify the VID or of an existing `AutoBill`—CashBox backfills the `AutoBill` Transaction history, (importing older Transactions for the subscription), and makes no attempt to update the `AutoBill` itself.

Note: While you can create new Accounts and PaymentMethods with this call, you cannot create new Products or BillingPlans, so be sure to create any Products or BillingPlans referenced by an input `MigrationTransaction` object before making the call.

AutoBill.migrate supports the following payment processors:

- Chase Paymentech
- Merchant e-Solutions
- Litle & Co.
- PayPal

AutoBill.migrate supports the following payment methods:

- Credit Card
- PayPal
- Merchant accepted payment (MAP)—Pay By Invoice

Input

srdf: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

`autobill`: the `AutoBill` object to migrate to CashBox.

To specify a regular billing in advance of, or after, the regularly scheduled Service Period start date, use the following parameters:
The AutoBill Object

- **specifiedBillingDay**: an integer (1-31) specifying the day of the month on which to bill (values 29-31 automatically work as the last day of the month for calendar months that do not have 29, 30, or 31 days). Providing no value or null in this parameter instructs CashBox to bill on the service period start date (the default).

- **billingRule**: an enumerated string value ([Advance or Arrears](#)), informing CashBox in which direction from the scheduled billing date to select the specified billing date. If no specified_billing_day is provided, this parameter is ignored.

You can use these parameters to change the billing date at migration (or creation) of an AutoBill object, or specify them later using `AutoBill.update`. For more information, see "Advance and Arrears Billing Option for AutoBills" in the [CashBox Programming Guide](#).

---

Note: For this call, the following AutoBill data members are required:

- account
- billingPlan
- currency
- items
- paymentMethod (For this call, this field must be CreditCard or MAP)
- 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.)

When migrating a subscription with a Merchant Accepted Payment (MAP), specifying a **nextPeriodStartDate** will alter the invoice/billing schedule. If this parameter is not set, CashBox will create all invoices from the last transaction provided to “now” and schedule the next (an invoice in OPEN status) according to the billing plan provided.

**migrationTransactions**: an array of `MigrationTransaction` objects that define the history of this AutoBill.

When migrating Transactions with a MAP, a 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, Canceled, Refunded, Settled, or Void. No other `TransactionStatus` values may be used for this call.

Also for MAP, we strongly recommend that you provide at least the last invoice transaction. Regardless of how many transactions you include, CashBox will create invoices for all periods between the “last transaction” and “now”. If you provide no Transactions, CashBox will create an invoice for every period since the indicated start date of the subscription. Unless a successful Transaction is provided, CashBox will assume an invoice period is unpaid (DUE or OVERDUE).

CashBox will not create invoices for periods prior to the last provided transaction unless covered by another migrated transaction—unspecified periods will be assumed to have been PAID in full.

---

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 Canceled will trigger a retry.

---

**cancelReason**: (Optional) reason for canceling the AutoBill. You can use predefined CashBox cancel reason codes, or define additional codes using the CashBox Portal or the
API. (See "Canceling AutoBills with Reason Codes" in the CashBox Programmer’s Guide.) Supplying an undefined cancel reason code may result in an error.

**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 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• At least one migrationTransaction must be included in <code>AutoBill.migrate</code> request.</td>
</tr>
<tr>
<td></td>
<td>• Error validating parameters for AutoBill Migration.</td>
</tr>
<tr>
<td></td>
<td>• Migrated AutoBills must have at least one item.</td>
</tr>
<tr>
<td></td>
<td>• Product must be pre-defined for AutoBillItem <code>autoBillItem</code>.</td>
</tr>
<tr>
<td></td>
<td>• Product not defined for AutoBillItem <code>autoBillItem</code>.</td>
</tr>
<tr>
<td></td>
<td>• Invalid AutoBillItem <code>autoBillItem.addedDate</code>: date.</td>
</tr>
<tr>
<td></td>
<td>• Invalid AutoBillItem <code>autoBillItem.removedDate</code>: date.</td>
</tr>
<tr>
<td></td>
<td>• BillingPlan must be pre-defined.</td>
</tr>
<tr>
<td></td>
<td>• BillingPlan with identifier <code>identifier</code> not found.</td>
</tr>
<tr>
<td></td>
<td>• Campaign code <code>campaignCode</code> is not valid.</td>
</tr>
<tr>
<td></td>
<td>• Invalid value for transitioningFromMerchantAutoBillItemId: <code>autoBillItemId</code>.</td>
</tr>
<tr>
<td></td>
<td>• Invalid value for transitioningToMerchantAutoBillItemId: <code>autoBillItemId</code>.</td>
</tr>
<tr>
<td></td>
<td>• Invalid value for <code>nextPeriodStartDate</code>.</td>
</tr>
<tr>
<td></td>
<td>• Transaction Migration attempt failed: <code>error</code>.</td>
</tr>
<tr>
<td></td>
<td>• (See the <code>Transaction.migrate</code> error list for details that could be appended to this message.)</td>
</tr>
<tr>
<td></td>
<td>• Migration Transactions not specified.</td>
</tr>
<tr>
<td></td>
<td>• PaymentMethod not specified.</td>
</tr>
<tr>
<td></td>
<td>• Unsupported Payment Type: <code>type</code>.</td>
</tr>
<tr>
<td></td>
<td>• TransactionBilling migration details not provided for initialization of AutoBill.</td>
</tr>
<tr>
<td></td>
<td>• No BillingPlan: Cannot determine TransactionBillingSequence details.</td>
</tr>
<tr>
<td></td>
<td>• Billing Plan on last Transaction submitted does not match AutoBill BillingPlan.</td>
</tr>
<tr>
<td></td>
<td>• AutoBillCycle (transaction_billing_sequence) not defined.</td>
</tr>
<tr>
<td></td>
<td>• Unable to determine current AutoBill BillingPlanPeriod.</td>
</tr>
<tr>
<td></td>
<td>• Unable to determine Billing Period billing sequence.Failed to determine BillingPlanPeriod index for AutoBill Migration.</td>
</tr>
<tr>
<td></td>
<td>• BillingPlanCycle cannot be greater than autoBillCycle (transaction_billing_sequence).</td>
</tr>
<tr>
<td></td>
<td>• Currency code not defined in row <code>n</code> of Transaction Array for AutoBill Migration.</td>
</tr>
</tbody>
</table>
400 (continued)  
- **Transaction currency (currency)** does not match AutoBill currency (currency).
- **Transaction for retry # n in billing sequence m** already exists.
- **Transaction migration failed for Transaction # n** during Migration of AutoBill autoBill.
  - *(See the Transaction.migrate error list for details that could be appended to this message.)*
- **Mismatch found between AutoBill billing sequence (n),** and the max TransactionBilling sequence (m).
- **Transactions cannot be associated with an AutoBill's final TransactionBilling.**
- **TransactionBilling for sequence sequence is in invalid state (state) for AutoBill migration.**
  - *(TransactionBillings generated by the migrate process should be in one of the following states: Success, Free, or Deferred.)*
- **Original activity date for historical TransactionBilling sequence n is in the future (date).**
- **billing date for TransactionBilling (date) is greater than latest allowed (date).**
- **Transaction (index) in sequence sequence is in an invalid state (state) for AutoBill migration.**
  - *(The Transaction state (which is derived from the disposition log) must be one of the following: Captured, Cancelled, Refunded, Settled, or Void.)*
- **NRC Transaction in sequence $sequence is in an invalid state (state) for AutoBill migration**
  - *(Non-Recurring-Charge Transactions must be in one of the states specified for Recurring Transactions (above).)*
- **autoBillCycle not defined.**
- **billingPlanCycle not defined.**
- **billingPeriodStart not defined.**
- **Failed to map AutoBill Items to Transaction Items during Transaction Migration.**
- **Cannot map Transaction Items to AutoBill Items - No Transaction Items!**
- **Cannot map Transaction Items to AutoBill Items - No AutoBill Items!**
- **Failed to determine Product SKU for AutoBill Item index n while attempting to map Transaction Items to AutoBill Items.**
- **Failed to map AutoBillItem ID autoBillItemid, to a TransactionItem.**
- **Unable to determine AutoBill BillingPlanPeriod for Transaction ident transactionId.**
- **Unable to determine Billing Period billing sequence for Transaction ident transactionId.**
- **Failed to determine BillingPlanPeriod index associated with Transaction ident transactionId.**
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 = 'c6743226ea41af9db71c0c612a870bfcaa68fa7';
$productVid = '124a5540e359d59ba2a301a4b6cd5434f5c99d3';
$accountVid = '3915038987280cc31b103dbdf291cfd68181b385';
$paymentmethodVid = '822690237671dbb37014bb4c5262e0067ab94f97';
$addressVid = '3c14d744baa5cd618c1e0ff2c1f54b408e8c65e9';

//Define the AutoBill to be Migrated
$autobill = new AutoBill();

//Set the BillingPlan, Product, Account, PaymentMethod, Address
$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);

$autobill->setaccount($account);
$autobill->setItems(array($item));
$autobill->setBillingPlan($billPlan);
$autobill->currency('USD');
$autobill->setCustomerAutoBillName('MGRT_1391710573_cabn_1391710456');
$autobill->set('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();
$txItemA->setItem($itemType, '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();
$txItemB->setItem($itemType, 'RecurringCharge');
$txItemB->setMerchantAutoBillItemId('merchantAutoBillItem1391710456');
$txItemB->setName('product_1391710450_1');
$txItemB->setPrice(42.00);
$txItemB->setServicePeriodStartDate('2014-01-06T00:00:00');
$txItemB->setServicePeriodEndDate('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 TransactionStatusCreditCard();
$creditCardStatusA->setAuthCode('000');
$statusLogA = new TransactionStatus();
$statusLogA->setCreditCardStatus($creditCardStatusA);
$statusLogA->setPaymentMethodType('CreditCard');
$statusLogA->setStatus('Captured');
$statusLogA->setTimestamp('2014-02-06T16:06:08-08:00');

$creditCardStatusB = new TransactionStatusCreditCard();
$creditCardStatusB->setAuthCode('000');
$statusLogB = new TransactionStatus();
$statusLogB->setCreditCardStatus($creditCardStatusB);
$statusLogB->setPaymentMethodType('CreditCard');
$statusLogB->setStatus('New');
$statusLogB->setTimestamp('2014-02-06T14:51:08-08:00');
$migrationTransaction = new MigrationTransaction();
$migrationTransaction->setAccount($account);
$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.

You cannot use the AutoBill.modify method to

- create new AutoBills or Billing Plans. You can only use this method to modify existing AutoBills or Billing Plans.
- modify AutoBills that use the MerchantAcceptedPayment payment method.

For AutoBills that use Seasonal Billing Plans, note the following considerations for subscription changes:

- You can use AutoBill.modify to credit customers for paid but unused days of service, and debit customers for additions to their subscriptions. AutoBill.modify does not prorate debits for subscription additions. The net amount is the full cost of the debit minus the prorated credits.
- You can use AutoBill.modify to credit customers for paid but unused days of service, and debit customers a reduced amount for a new, downgraded subscription. AutoBill.modify does not prorate subscription downgrades. The net credit amount is the full amount of the prorated credits minus the cost of the debit.

---

**Note:** When you use AutoBill.modify for Seasonal Billing Plans, the effectiveDate must be equal to today.

---

AutoBill.modify allows you to

- modify AutoBills that have any number of AutoBillItems.
- add, remove, edit or replace multiple AutoBillItems. You can use remove and replace in a single call, but you must use edit by itself.
- 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.)
- work with Seasonal Billing Plans to credit or debit customers for subscription changes.
- maintain a history of modified AutoBills.
- generate a single, prorated 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:** If you are using CashBox for sales tax calculations, you must integrate your application with the Avalara Tax Engine to use the AutoBill.modify method. To integrate your application with the Avalara Tax Engine, contact your Vindicia support representative.

---

- change the billing date to a new date that is before or after the current billing date (or service date) within the billing cycle, and have CashBox automatically prorate the charges.
Input

**srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**autobill**: the AutoBill object to modify. Identify this object with its VID or .

**billProratedPeriod**: a Boolean flag that specifies 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 prorate regardless of the billProratedPeriod setting. Transactions generated as a result of this option will automatically include the vin:AutoBillVID and the vin:entifier name-value pairs.

**effectiveDate**: indicates when the modification becomes effective.

- nextBill initiates changes with the next Billing Cycle.
- today initiates 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 is changed with this call CashBox will prorate both credits and charges, and bill or refund immediately; future recurring billings will remain scheduled as before.

---

**Note**: For Seasonal Billing Plans, CashBox requires that the effectiveDate is set to today.

**changeBillingPlanTo**: the BillingPlan to replace the current one 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 Plan, and reset the billing date for the AutoBill to today.

**autoBillItemModifications**: an array of AutoBillItemModification objects that define the changes to be made to the existing AutoBill.

- **editAutoBillItem**: an existing AutoBillItem to modify.
- **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, when set to `true`, returns 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 are made when `dryrun` is `true`.

**transaction**: (Optional) Transaction ID of a pre-authorized transaction (using `transaction.auth()`). Use a pre-authorized transaction to verify and reserve full funds but modify an AutoBill at some arbitrary later date without having to cancel the prior `Auth` or submit a new one.

**Note:** This feature is supported only for the Barclaycard Smartpay Gateway.

**changeBillingDateTo**: (Optional) New billing date. Use this parameter to change the billing date and have CashBox automatically prorate the charges. You can change the billing date to a later date (extend the billing cycle), or change the billing date to an earlier date (shorten the billing cycle). Note that you must also set `billProratedPeriod` to `true` to have CashBox calculate prorated charges.

- Extend the billing cycle. A $30 monthly subscription bills on the 1st of the month. On February 20, `changeBillingDateTo` is set to the 16th of March, effective immediately. CashBox calculates a new charge for the period 3-1 through 3-15, for an immediate prorated charge of $15. Next regular monthly billing is on the 16th of March.
- Shorten the billing cycle. A $30 monthly subscription bills on the 1st of the month. On February 10, `changeBillingDateTo` is set to the 16th of March, effective immediately. CashBox calculates a refund for the period 2-16 through 2-28 and applies it against the initial charge of $30 billed on February 1st. Next regular monthly billing is on the 16th of March.

**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 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Could not load the AutoBill.</td>
</tr>
<tr>
<td></td>
<td>• Invalid value for <code>effectiveDate</code>: <code>value</code>.</td>
</tr>
<tr>
<td></td>
<td>• BillingPlan used in <code>modify</code> must already exist.</td>
</tr>
<tr>
<td></td>
<td>• No <code>AutoBillItem</code> found for <code>criteria</code>.</td>
</tr>
<tr>
<td></td>
<td>• Could not determine a unique item for <code>criteria</code>.</td>
</tr>
<tr>
<td></td>
<td>• The currency field on an added <code>AutoBillItem</code> does not match the currency on the AutoBill.</td>
</tr>
<tr>
<td></td>
<td>• Could not determine start date for <code>AutoBill</code>.</td>
</tr>
<tr>
<td></td>
<td>• Campaign code <code>code</code> could not be redeemed.</td>
</tr>
<tr>
<td></td>
<td>• Cannot redeem a time grant campaign on a <code>BillingPlan</code> via <code>AutoBill::modify()</code>.</td>
</tr>
<tr>
<td></td>
<td>• <code>BillingPlan</code> <code>merchantPlanId</code> is not eligible for campaign code <code>code</code>.</td>
</tr>
<tr>
<td></td>
<td>• Modifications would result in an <code>AutoBill</code> with no active <code>AutoBillItems</code>.</td>
</tr>
<tr>
<td></td>
<td>• Cannot remove items or change the <code>BillingPlan</code> effective today for an <code>AutoBill</code> with a Token PaymentMethod.</td>
</tr>
<tr>
<td></td>
<td>• Could not set <code>AutoBill</code> to 'Processing' status.</td>
</tr>
<tr>
<td></td>
<td>• Saving modified <code>AutoBill</code> failed; rolling back.</td>
</tr>
<tr>
<td></td>
<td>• <code>AutoBillItem</code> to be removed has already been removed.</td>
</tr>
<tr>
<td></td>
<td>• Could not determine start date for <code>AutoBill</code>.</td>
</tr>
<tr>
<td></td>
<td>• Campaign code <code>code</code> could not be redeemed.</td>
</tr>
<tr>
<td></td>
<td>• Cannot redeem a time grant campaign on a single <code>AutoBillItem</code> via <code>AutoBill::modify()</code>.</td>
</tr>
<tr>
<td></td>
<td>• Product <code>merchantProductId</code> is not eligible for campaign code <code>code</code>.</td>
</tr>
<tr>
<td></td>
<td>• Product <code>merchantProductId</code> is not eligible for an active time grant campaign.</td>
</tr>
<tr>
<td>402</td>
<td>• Modify transaction authorization failed.</td>
</tr>
</tbody>
</table>
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 = 'd79ae3429ff102383b76d8f1eae8da52bd7dclaf';
replacementItemIdent = 'upgradedProductItem';
$newItemIdent = 'addedProductItem';

// create and identify the AutoBill to be modified
$autobill = new AutoBill();
$autobill->($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 (where supported). 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

srd: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

autobill: an AutoBill object to which you wish to grant credit, if redemption of the gift card is successful. Use the 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• AutoBill not found.</td>
</tr>
<tr>
<td></td>
<td>• Failed to translate gift card error-description.</td>
</tr>
<tr>
<td></td>
<td>• Failed to redeem gift card error-description.</td>
</tr>
<tr>
<td></td>
<td>• Failed to retrieve gift card error-description.</td>
</tr>
<tr>
<td></td>
<td>• Failed to save AutoBill after gift card redemption attempt.</td>
</tr>
<tr>
<td></td>
<td>• Failed to reload AutoBill after gift card redemption attempt error-description.</td>
</tr>
<tr>
<td></td>
<td>• Redemption attempt failed for Gift Card ID gift-card-details.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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->set('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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
</table>
| 400         | One of the following:  
  - AutoBill not found.  
  - Neither paymentId nor invoiceId: indexNumber provided for reversal attempt.  
  - Failed to add reverse payment: error-description. |
| 404         | Payment ID not found. |
Example

// to reverse a payment made using makePayment
$autobill = new AutoBill();
$autobill->set($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**

`srđ`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srđ`. A null `srđ` returns the complete response.

`autobill`: the `AutoBill` object from which you wish to revoke credit. Use the `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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• AutoBill not found.</td>
</tr>
<tr>
<td></td>
<td>• Failed to translate credit <code>error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Failed to revoke credit <code>error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Failed to save <code>AutoBill</code> after revoking credit.</td>
</tr>
<tr>
<td></td>
<td>• Failed to reload <code>AutoBill</code> after revoking credit <code>error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Data validation error: Missing required parameter <code>credit</code>.</td>
</tr>
</tbody>
</table>
Example

// to revoke credit from an autobill

$abill = new AutoBill();

// autobill id for customer's existing subscription
// to a game
$abill->set('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";
}
The `settlementQuote` method generates a quote for the settlement amount that would result from an `AutoBill.cancel` call made with both the `disentitle` and `settle` parameters set to `True`. You can then provide your customers this preview of how much prorated amount would be refunded upon cancellation and, if the customer agrees, commit the cancellation.

Additionally, advance knowledge of how much would be subject to refund can guide your refund policy decisions for allowing immediate disentitlement and/or choosing to settle.

**Input**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call. A null `srd` returns the complete response.

- **autobill**: `AutoBill` object for which the quote is to be generated. You can identify this object with either its VID or your `AutoBill` ID.

- **force**: Boolean flag that, when set to `true`, forces generation of a quote even if the minimum commitment period for this `AutoBill` object has not yet been satisfied. In the case of an `AutoBill` that has a `BillingPlan` with a `minimumCommitment` period, and the subscription is still in the commitment period, the penalty is calculated as defined in the `cancelFeePolicy`.

- **amountOnly**: Boolean flag that, when set to `true`, restricts the output to just the settlement amount. When set to `false`, or not specified, `settlementQuote` will also generate mock Refunds/Transactions that would be generated in response to an actual `AutoBill.cancel` call with `disentitle` and `settle` parameters set.

- **cancelFeePolicy**: policy for handling the settlement fee. Options for specifying how to cancel the policy include:
  - Waive the Cancel Fee—the default.
  - Prorate Cancel Fee—calculate fee based on time remaining in the commit period.
  - Full Cancel Fee—apply the full early cancellation fee.

**Output**

- **amount**: Settlement amount (Decimal). Negative for Refund, Positive for charge.

- **currency**: ISO 4217 currency code for amount.

- **token**: Vindicia Token for amount when the `AutoBill` uses a Token `BillingPlan/PaymentMethod`.

- **note**: a string providing additional information about the success or failure of the settlement quote.

- **transactions**: An array of `Transaction` objects that provide details of the charges the customer will incur with the settlement.
**refunds**: An array of `Refund` objects that provide details of credits that will be refunded to the customer with the settlement.

**Returns**

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Success</td>
</tr>
</tbody>
</table>
| 400         | AutoBill was not specified  
Error processing settlement quote request: [Failure Details] |
| 403         | Settlement quote attempt failed (Usually a soft failure, such as the tax vendor being down)  
Minimum commitment not fulfilled for this AutoBill |
| 405         | Cannot cancel Upgraded AutoBill |
| 500         | Something went terribly wrong (SYSERROR) |

**Example**

```
$autobillVid = 'ffff2d063iamnotanalien96a284234b006d7b7c'; // for illustration purposes only!

$autobill_factory = new autobill();

//Fetch existing AutoBill
$response = $autobill_factory->fetchByVid($autobillVid);
if($response['returnCode'] == 200)
{
    $autobill = $response['autobill'];
    $force = true;
    $amount_only = false;

    //Make settlementQuote request
    $response = $autobill_factory->settlementQuote(
        $autobill, // Vindicia::Soap::AutoBill
        $force,    // xsd:boolean
        $amount_only // xsd:boolean
    );
    if($response['returnCode'] == 200)
    {
        $settlement_amount = $response['amount'];
        // Settlement quote amount (negative for credit)
        $settlement_currency = $response['currency'];
        // ISO 4217 Currency code
        @settlement_transactions = $response['transactions'];
        // Array of Vindicia::Soap::Transaction objects
        @settlement_refunds = $response['refunds'];
        // Array of Vindicia::Soap::Refund objects
        $settlement_note = $response['note'];
        // String providing note on quote attempt
    }
    else
```
{  
  // Handle settlementQuote failure
}

else
{
  // Handle fetchByVid failure
}
update

The update method is used to create a new AutoBill object. An AutoBill object represents a customer subscription with recurring billing. Be sure to properly construct the AutoBill (see Section 4.1: AutoBill Data Members) before passing it into this call.

Call this method to first risk-screen the related Payment Method for the AutoBill object. Enabling risk screening causes this method to score a Transaction for the related Payment Method. (See the score method.) If the score is below the 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.

If the billing is scheduled for today, CashBox authorizes the full amount immediately. In the event of a failure, CashBox responds according to the immediateAuthFailurePolicy, whose options include the following:

- doNotSaveAutoBill—(default) deletes the AutoBill without saving it
- putAutoBillInRetryCycle—creates AutoBill and retries the authorization
- putAutoBillInRetryCycleIfPaymentMethodIsValid—(recommended)

If billing is scheduled for a future date, the validateForFuturePayment flag is used to determine whether to do a minimal authorization ($0/$1) immediately. To request an immediate validation for a new or existing AutoBill when billing is not scheduled for today, set the validateForFuturePayment flag to true. An immediate minimal authorization will be performed. This flag is ignored when creating a new AutoBill with an immediate billing (scheduled for today) in favor of the full amount authorization.

By default, if a new AutoBill creation or modification requires billing within 25-hours, CashBox does an immediate billing with full authorization. The purpose of this behavior is to minimize processing fees by combining the validation transaction with the billing transaction. This behavior has the added benefit of enabling you to respond to payment failures in real time—that is, while the customer is on line.

If, however, you offer a one-day initial billing cycle, or if you require that the initial transaction date be tomorrow rather than today, use the DelayFullAuthToInitialBillingDate option to prevent immediate billing when an AutoBill is created. This causes CashBox to initiate a validation transaction today, and a billing transaction tomorrow (the specified start date of AutoBill/subscription). This option is false by default. See initialAuth below.

You can also use the AutoBill.update() method to specify a regular billing in advance of, or after, the Service Period start date. You can allow the service dates to be maintained according to the billing plan, while billing for those periods on specified days of the month, before or after the scheduled service date, using the specifiedBillingDay and billingRule parameters.

To create an AutoBill object, initialize the object and set the values for its data members as appropriate, then call the update() method to create the object on the Vindicia server. 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).
A **Product** can have an array of products beneath it (sub-products). When a sub-product is placed on an AutoBill, all attributes of the top-level product will apply to the AutoBill, except for entitlements, which will be the union of the entitlements of all of the products and sub-products.

---

**Note:** The `AutoBill.update` method can not be used to add Products to an AutoBill. To add Products, use `AutoBill.modify`.

---

To apply a Campaign discount to an AutoBill, the Billing Plan must have prices defined in a currency that matches the currency set on the AutoBill. If the Billing Plan price is defined in currencies different 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**

**srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A `null` `srd` returns the complete response.

**autobill:** the AutoBill object to create or update. Identify this object with its VID or .

To specify a regular billing in advance of, or after, the regularly scheduled Service Period start date, use the following parameters:

- **specifiedBillingDay:** an integer (1-31) specifying the day of the month on which to bill (values 29-31 automatically work as the last day of the month for calendar months that do not have 29, 30, or 31 days). Providing no value or null in this parameter instructs CashBox to bill on the service period start date (the default).

- **billingRule:** an enumerated string value (Advance or Arrears), informing CashBox in which direction from the scheduled billing date to select the specified billing date. If no `specified_billing_day` is provided, this parameter is ignored.

You can also use these parameters to change the billing date at migration of an AutoBill object. For more information, see "Advance and Arrears Billing Option for AutoBills" in the CashBox Programming Guide.

**ImmediateAuthFailurePolicy:** if the billing is scheduled for today, CashBox authorizes the full amount immediately. In the event of a failure of the initial transaction for the AutoBill, CashBox responds according to the `immediateAuthFailurePolicy`, whose options include the following:

- **doNotSaveAutoBill**—(default) deletes the AutoBill. Select this setting if you want to start over, ask the consumer for a new credit card number, and create a new AutoBill.

- **putAutoBillInRetryCycle**—creates the AutoBill and places it in a "retry" cycle, in which CashBox continuously retries the credit card number.

- **putAutoBillInRetryCycleIfPaymentMethodIsValid**—(recommended.) Creates the AutoBill and retries the credit card only if CashBox has evidence that the authorization will succeed.
**validateForFuturePayment**: if billing is scheduled for a future date, this flag is used to determine whether to do a minimal authorization ($0/$1) immediately.

**minChargebackProbability**: a number between 0 and 100 by which you specify your fraud risk tolerance level. A chargeback probability (also called the risk-screening score or risk score) of 100 indicates that CashBox is 100% certain a transaction is fraudulent and will result in a chargeback. Specify your acceptable threshold for chargeback probability with this parameter. If the score evaluates to more than your tolerance level, the update call will fail.

If you do not set **minChargebackProbability**, CashBox 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 which, when set to true, causes CashBox to override the AVS policy and update the paymentMethod, regardless of the AVS return code. If set to false or null, (and if **validateForFuturePayment** is set to true) the AVS return code will be used to determine whether to update the paymentMethod.

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

If both **ignoreAvsPolicy** and **ignoreCvnPolicy** are true, no policy evaluation will be done. If only one of the 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.

**Note**: The AutoBill will not be saved if validation is requested and fails.

The AVS and CVN Policy Evaluation policy Results are mapped to the AutoBill and Entitlement creation as follows:

<table>
<thead>
<tr>
<th>Policy Evaluation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>AutoBill is active and entitlement is granted.</td>
</tr>
<tr>
<td>Pending</td>
<td>AutoBill is pending and entitlement is granted.</td>
</tr>
<tr>
<td>Fail</td>
<td>AutoBill is canceled and entitlement is inactive.</td>
</tr>
</tbody>
</table>

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

**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`.)

Note: Do not change the established currency type.

cancelReason: (Optional) reason for canceling the AutoBill. You can use predefined CashBox cancel reason codes, or define additional codes using the CashBox Portal or the API. (See "Canceling AutoBills with Reason Codes" in the CashBox Programming Guide.) Supplying an undefined cancel reason code may result in an error.

initialAuth: (Optional) An enumerated value that specifies how to manage authorization and capture when the scheduled initial billing is within 25 hours. Default behavior is equivalent to `AuthImminentBilling`:

- `AuthImminentBilling`: By default, charges to be billed within 25 hours are authorized and captured immediately to save on processing fees. The immediate full amount authorization provides the validation. Returns.
- `DelayFullAuthToInitialBillingDate`: This value forces the specified validation behavior now, delaying the full authorization and capture attempt until the specified AutoBill Start Date.

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Can not use this reason code: $reasonCode if its reserved for Vindicia use only.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load cancel reason code.</td>
</tr>
</tbody>
</table>

transaction: (Optional) Transaction ID of a pre-authorized transaction (using `transaction.auth()`). Use a pre-authorized transaction to verify and reserve full funds but create or modify an AutoBill at some arbitrary later date without having to cancel the prior Auth or submit a new one.

Note: This feature is supported only for the Barclaycard Smartpay Gateway.

unknownStart: (Optional) a Boolean flag that, when set to `True`, will create an AutoBill with no start date and status Pending Activation. An AutoBill created with an unknown start date must be activated with an `AutoBill.Activate()` call. Otherwise the `AutoBill` will persist in the pending activation state for up to two years from the creation date, and then be automatically canceled.

The unknown start functionality is currently supported only for credit card and debit card payment methods, and for season based entitlements. See the `activate` method for the `AutoBill` object.

fullAuthDelayedCapture: (Note: for future development, not currently implemented.) If true and the AutoBill has a future start date or a free trial period, the AutoBill will be created with an `auth` for the full amount of the first paid-for period.
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:** contains the response from the payment processor (either the initialTransaction auth response or that of the validation response). For example, the Address Verification Service (AVS) and Card Verification Number (CVN) response codes.

**initialTransaction:** The initial transaction for new `AutoBill` objects. Creates an initial transaction for immediate billing, even if the amount is zero. Outputs a transaction object that contains complete information about the transaction, including the date, amount, and currency of the first billing.

For some payment methods that do not support immediate payment (MerchantAcceptedPayment/invoicing), the Transaction is still included to provide the relevant Transaction and item details, including taxes and discounts, to support display in quotes (using `dryrun`) or confirmation pages.

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
</table>
| 400         | One of the following:  
|             | • Failed to translate credit error-description.  
|             | • Unable to create AutoBill: error-description.  
|             | • Data validation error: Failed to create Payment-Type-Specific Payment Record: Credit Card conversion failed: Credit Card failed Luhn check.  
|             | • Unable to create autobill: Must specify product to create autobill with!  
|             | • Campaign code XYZ is not usable: Code XYZ is not valid.  
|             | • No eligible, undiscounted items found for campaign code.  
|             | • This Credit Card already exists—Policy Violation" (eradicate the newly created but failed one or ensure it is set INACTIVE).  
|             | You receive this error message when you have the Credit_Card_Constraints merchant option enabled on your site. This message means that someone is attempting to use a credit card that is already being used as the payment method on an existing account. This is not allowed. |
| 402         | Unable to create AutoBill: error-description.  
|             | (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. |
// 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->set('ab-44822'); // your ID for the AutoBill
$autobill->setCurrency('USD');
$validate = true;
$fraudScore = 100; // do not want to do risk screening
$response = $autobill->update('putAutoBillInRetryCycleIfPaymentMethodIsValid',
  $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");
  }
}
updateCancelReason

The updateCancelReason method enables you to create or edit a reason associated with an AutoBill when you cancel it. A cancel reason contains a reason code and a description.

Input

**srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**cancelReason**: An object that contains the cancel reason code and description. See CancelReason Subobject.

Output

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

Example

```php
$code = "N5001"
$desc = "Account shut down for identity theft"
$soap_cr = New CancelReason($code, $desc );
$response = $autobill->updateCancelReason($soap_cr);
```
writeOffInvoice

Marks an Invoice as writtenOff, the debt unable to be collected.

**Input**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

- **autobill**: the AutoBill associated with the Invoice to be written off.
- **invoiceId**: the ID of the Invoice to be written 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
</table>
| 400         | One of the following:  
|             |   - AutoBill not found.  
|             |   - Invoiced ID not provided.  
|             |   - Failed to write off invoice: error-description.  |
| 404         | Invoice ID not found.  |

**Example**

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

$response = $autobill->writeOffInvoice('inv-bac');

// check $response
```
The periodicity, frequency, and amount of billing 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 might 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 period of time. (You can define Billing Plans in any currency CashBox supports.)

You can also use Billing Plans to customize entitlement access in relation to the Billing Plan. For example, entitlements might be coincident with the plan (a customer is granted access for every month on which they make a payment), or entitlements could be separated from the billing sequence (a customer might 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.:

Note: Use of the forward slash character (/) in identifiers is not allowed. See Merchant Identifiers for more information.
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

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>billingStatementIdentifier</td>
<td>string</td>
<td>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. If GlobalCollect, MeS, Chase Paymentech or Litle is your payment processor, see Appendix A: Custom Billing Statement Identifier Requirements in the <em>CashBox Programming Guide</em>.</td>
</tr>
<tr>
<td>brTaxCategory</td>
<td>string</td>
<td>The Brazilian tax category for this item.</td>
</tr>
<tr>
<td>cancelFees</td>
<td>BillingPlanCancelFee</td>
<td>List of early cancellation fees for various currencies. See the BillingPlanCancelFee Subobject.</td>
</tr>
<tr>
<td>daysBeforeSeasonToBill</td>
<td>int</td>
<td>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.)</td>
</tr>
<tr>
<td>DaysEntitledAfterSeason</td>
<td>int</td>
<td>Defines the number of days after a Season ends that Entitlements will remain Active.</td>
</tr>
<tr>
<td>daysEntitledBeforeSeason</td>
<td>int</td>
<td>Defines the number of days before the Season begins that Entitlements will become Active.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Your description of the Billing Plan.</td>
</tr>
<tr>
<td>endOfLifeTimeStamp</td>
<td>dateTime</td>
<td>Optional. A time stamp that specifies the expiration date for this BillingPlan object. This value is for your information only, and does not affect CashBox operations.</td>
</tr>
<tr>
<td>entitledOffSeason</td>
<td>Boolean</td>
<td>A Boolean flag that, if set to true, sets Entitlements to remain Active in the off-season. (Default is false.)</td>
</tr>
<tr>
<td>entitlementsValidFor</td>
<td>string</td>
<td>The length of time for which Entitlements are valid after the last Billing date.</td>
</tr>
<tr>
<td>gracePeriodOverride</td>
<td>int</td>
<td>Number of days after the billing date when, if payment is not received, the AutoBill is stopped and the customer is disenrolled; overrides the global merchant default setting, if both are set.</td>
</tr>
</tbody>
</table>
### Table 5-1  BillingPlan Object Data Members  (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>merchantBillingPlanId</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
</tbody>
</table>
| merchantEntitlementIds       | MerchantEntitlementId[]| ID and description of the entitlement, specified by you:  
- **id**—Unique, merchant-defined ID of this entitlement. Free-form string of 1024 or fewer bytes.  
- **description**—Merchant-defined description of this entitlement. Free-form string of 1024 or fewer bytes.  

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. CashBox returns these IDs to you inside Entitlement objects along with the dates till which they are considered valid for a given customer.  

| minimumCommitment            | int                   | Number of billing cycles the customer is contractually obligated to complete without possibly incurring a cancellation fee.  
This setting is referenced by AutoBill.cancel(), which requires that the force flag be set to True to succeed if the minimum commitment has not been satisfied.  

| 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. See The NameValuePair Object.  

| periods                      | BillingPlanPeriod[]   | An array of items that describe the billing plan. 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.  
A Billing Plan can define a free trial period, followed by a monthly subscription service; or it can define a Seasonal Billing Plan whereby a customer is billed only during the Season to which they are subscribed.  
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.  

### Table 5-1 BillingPlan Object Data Members (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repeatEvery</td>
<td>string</td>
<td>If a Billing Period set repeats, this value defines the length of time after the first billing that the set should repeat.</td>
</tr>
<tr>
<td>seasonSet</td>
<td>SeasonSet</td>
<td>The SeasonSet to which the Billing Plan applies. (May be null.) See SeasonSet Data Members.</td>
</tr>
<tr>
<td>skipInitial-FreeWhenRepeat-</td>
<td>Boolean</td>
<td>A Boolean flag that, if set to true, excludes initial free periods when repeating a Billing Period set. (Default is true.)</td>
</tr>
<tr>
<td>ing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>status</td>
<td>BillingPlanStatus</td>
<td>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. See the BillingPlanStatus Subobject.</td>
</tr>
<tr>
<td>taxClassification</td>
<td>string</td>
<td>A string that defines your tax classification for this BillingPlan. A null value in this field causes CashBox to use the tax classification associated with the primary Product of the AutoBill unless over-ridden by configuration in an external tax service (for example, Avalara). If you put a dash (--) in this field, no tax classification is sent to the tax service and CashBox reverts to the default behavior configured in your tax service.</td>
</tr>
<tr>
<td>timesToRun</td>
<td>string</td>
<td>The number of times the sequence of Billing Periods should be repeated. Valid input includes positive integers, or &quot;unlimited.&quot; (Default is null.)</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
5.2 BillingPlan Subobjects

The BillingPlan object has several subobjects:

- BillingPlanPeriod Subobject
- BillingPlanPeriodType Subobject
- BillingPlanPrice Subobject
- BillingPlanStatus Subobject
- BillingPlanCancelFee 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

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycles</td>
<td>int</td>
<td>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.</td>
</tr>
<tr>
<td>doNotNotify-FirstBill</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>expireWarning-Days</td>
<td>int</td>
<td>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.</td>
</tr>
<tr>
<td>free</td>
<td>Boolean</td>
<td>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, however, bill for Charges explicitly added during the period. 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.</td>
</tr>
<tr>
<td>prices</td>
<td>BillingPlan-Price[]</td>
<td>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. See the BillingPlanPrice Subobject.</td>
</tr>
</tbody>
</table>
### BillingPlanPeriod Subobject

The unit of time the Period describes.

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantity</td>
<td>int</td>
<td>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.)</td>
</tr>
<tr>
<td>type</td>
<td>BillingPlanPeriodType</td>
<td>An enumerated string that specifies the unit (day, week, month, or year) for the duration of the billing period. Note: For rolling promotion campaigns spanning multiple billing periods, the billing terms must be homogeneous. You cannot offer a rolling campaign if the Billing Plan has varied term lengths. Free cycles are ignored. See the BillingPlanPeriodType Subobject.</td>
</tr>
</tbody>
</table>

### BillingPlanPrice Subobject

A price for the BillingPlan.

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>decimal</td>
<td>The amount to bill. Must be zero (for free trials) or positive.</td>
</tr>
<tr>
<td>currency</td>
<td>string</td>
<td>The ISO 4217 currency code (see <a href="http://www.xe.com/iso4217.htm">www.xe.com/iso4217.htm</a>) of billing. The default is USD.</td>
</tr>
</tbody>
</table>
Table 5-4  BillingPlanPrice Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priceListName</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>tokenAmount</td>
<td>TokenAmount</td>
<td>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.</td>
</tr>
</tbody>
</table>

BillingPlanStatus Subobject

Describes whether the BillingPlan is Active or Suspended. Suspended Billing Plans can not be AutoBill renewed.

Table 5-5  BillingPlanStatus Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>string</td>
<td>The BillingPlan object is active (accessible to the customer).</td>
</tr>
<tr>
<td>Suspended</td>
<td>string</td>
<td>The BillingPlan object is inactive (inaccessible to the customer).</td>
</tr>
</tbody>
</table>

BillingPlanCancelFee Subobject

Defines a fee for early cancellation of an AutoBill subscription that uses this BillingPlan.

Table 5-6  BillingPlanCancelFee Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount</td>
<td>decimal</td>
<td>Amount of the fee. Must be a positive value.</td>
</tr>
<tr>
<td>Currency</td>
<td>string</td>
<td>ISO 4217 currency code for the billing. Defaults to USD.</td>
</tr>
</tbody>
</table>

5.3  BillingPlan Methods

The following table summarizes the methods for the BillingPlan object.

Table 5-7  BillingPlan Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetchAll</td>
<td>Returns all the BillingPlan objects.</td>
</tr>
</tbody>
</table>
## BillingPlan Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetchAllInSeason</td>
<td>Returns all in season Billing Plans.</td>
</tr>
<tr>
<td>fetchAllOffSeason</td>
<td>Returns all off-season Billing Plans.</td>
</tr>
<tr>
<td>fetchByBillingPlanStatus</td>
<td>Returns one or more BillingPlan objects whose status matches the input.</td>
</tr>
<tr>
<td>fetchByMerchantBillingPlanId</td>
<td>Returns a BillingPlan object whose ID assigned by you matches the input.</td>
</tr>
<tr>
<td>fetchByMerchantEntitlementId</td>
<td>Returns one or more BillingPlan objects whose entitlement ID assigned by you (merchantEntitlementId) matches the input.</td>
</tr>
<tr>
<td>fetchByVid</td>
<td>Returns a BillingPlan object whose VID matches the input.</td>
</tr>
<tr>
<td>update</td>
<td>Creates or updates a BillingPlan object.</td>
</tr>
</tbody>
</table>
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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>404</td>
<td>No BillingPlans found for merchant.</td>
</tr>
</tbody>
</table>

**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**

```php
$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

**srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**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

srd: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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

```php
// 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**

srdo: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>404</td>
<td>No BillingPlans found for Merchant Billing Plan ID <code>input-merchantBillingPlanId</code>.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>404</td>
<td>No BillingPlans found for entitlementId input-merchantEntitlementId.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>404</td>
<td>No BillingPlans found for VID input-vid.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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**

Use the `update` method to create a new `BillingPlan`, or to update 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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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->setDescription('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 and/or Billing Plans. Campaigns are discounts given over a period of time for a service or subscription, and may be applied to multiple Products and Billing Plans.

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.

**Note:** Use of the forward slash character (/) in identifiers is not allowed. See Merchant Identifiers for more information.

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

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>campaignId</td>
<td>string</td>
<td>Your unique identifier for this Campaign object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> Read-only once the Campaign has been created.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
<tr>
<td>campaignType</td>
<td>CampaignType</td>
<td>Specifies whether the Campaign is a Coupon or Promotion. Valid CampaignTypes include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Undefined (Used only for errors.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Coupon</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Promotion</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> Read-only once the Campaign is Active.</td>
</tr>
<tr>
<td>couponCodePrefix</td>
<td>string</td>
<td>Defines a prefix for the CashBox randomly generated Coupon Code.</td>
</tr>
<tr>
<td>couponCodeQuantity</td>
<td>integer</td>
<td>The number of Coupon Codes to generate.</td>
</tr>
<tr>
<td>couponCodeRequiresActivation</td>
<td>Boolean</td>
<td>A Boolean flag which, if set to true, creates inactive Coupon Codes. (Inactive Coupon Codes must be individually activated before use.) (Default is false.)</td>
</tr>
<tr>
<td>couponCodeSeparator</td>
<td>string</td>
<td>The (optional) character used to separate the couponCodePrefix from the CouponCode string. This may be any printable, non-alphanumeric ASCII character. <strong>Note:</strong> If you use a separator character between the Coupon Prefix string and the Coupon Code string, do not also use one in your prefix, as that might cause confusion about what is actually the prefix. The coupon prefix and auto-generated code: Holiday-2015-934sd553, for example, is ambiguous. Holiday2015-934sd553 is not.</td>
</tr>
</tbody>
</table>
The Campaign Object

### Table 6-1  Campaign Object Data Members  (*Continued*)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| cycles           | integer         | The number of Billing Cycles to which the Campaign discount will be applied.  
|                  |                 | **Note:** A Campaign must include either the `cycles` or the `expirationDate` data member. It cannot include both. |
|                  |                 | **Note:** CashBox does not support variable length Billing Cycles in Rolling Campaigns. The length of the `cycles` in each of the billing periods in the billing plan must be the same. For example, if the billing plan has three billing periods and the cycle length of the first billing period is 3 days, the cycle length of the second and third billing periods must also be 3 days. |
| description      | string          | Description of the Campaign.  
|                  |                 | **Note:** Read-only once the Campaign is Active. |
| eligible-BillingPlans | BillingPlan   | One or more Billing Plans eligible for this Campaign.  
|                  |                 | See Section 5.1: BillingPlan Data Members. |
| eligibleProducts | Product         | One or more Products eligible for this Campaign.  
|                  |                 | See Section 13.1: Product Data Members. |
| expirationDate   | dateTime        | The date the Campaign discount expires. (If `null`, the `offerEndDate` will be used.)  
|                  |                 | This date may be after the Campaign's `offerEndDate`, but cannot be before it.  
|                  |                 | **Note:** A Campaign must include either the `cycles` or the `expirationDate` data member, and it may not include both. |
| flatAmountDiscount | CurrencyAmount | Defines the discount, or discounts, as a Currency-Amount pair object.  
|                  |                 | **Note:** `flatAmountDiscount` and `percentageDiscount` are mutually exclusive. |
|                  |                 | **Note:** A Campaign must include either the `flatAmountDiscount` or the `percentageDiscount` data member, and it may not include both.  
|                  |                 | See CurrencyAmount Subobject for details. |
| maxRedemptions   | integer         | Sets the maximum number of different AutoBills to which a Campaign Code may be applied. |
| name             | string          | Name of the Campaign.  
|                  |                 | **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. |
### Table 6-1 Campaign Object Data Members (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>percentageDiscount</td>
<td>decimal</td>
<td>Defines the discount as a percentage of the original Product price. &lt;br&gt; <strong>Note:</strong> flatAmountDiscount and percentageDiscount are mutually exclusive. &lt;br&gt; <strong>Note:</strong> A Campaign must include either the flatAmountDiscount or the percentageDiscount data member, and it may not include both.</td>
</tr>
<tr>
<td>promotionCode</td>
<td>string</td>
<td>The redemption code associated with the Promotion.</td>
</tr>
<tr>
<td>promotionCodeAliases</td>
<td>string (0 or more)</td>
<td>An array of alternative redemption codes for the Promotion. &lt;br&gt; <strong>Note:</strong> Setting this array will replace any existing list of aliases; it will not add new values to an existing list.</td>
</tr>
<tr>
<td>restrictToNewSubscription</td>
<td>Boolean</td>
<td>A Boolean flag which, if true, indicates that the Campaign offer applies only to new AutoBills, and may not be applied to existing AutoBills.</td>
</tr>
<tr>
<td>state</td>
<td>CampaignState</td>
<td>State of the campaign: &lt;br&gt; • Undefined &lt;br&gt; • Active &lt;br&gt; • Inactive &lt;br&gt; • Pending &lt;br&gt; • Complete</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>campaignId</td>
<td>string</td>
<td>Read only. A unique identifier for the Campaign object.</td>
</tr>
<tr>
<td>code</td>
<td>string</td>
<td>The Coupon value, which consists of <code>&lt;couponCode-Prefix&gt;&lt;separator (if defined)&gt;&lt;randomly generated string&gt;</code>. This field is available only when creating the Coupon. When retrieving the CouponCode object, this field will always be returned blank.</td>
</tr>
<tr>
<td>note</td>
<td>string</td>
<td>An optional memo regarding the CouponCode.</td>
</tr>
</tbody>
</table>
| redeemedBy     | CouponRedeemedBy   | An array of CouponRedeemedBy objects, which lists the Account and date on which the Coupon was redeemed. Fields include:  
|                |                    | • merchantAccountId (string)  
|                |                    | • accountVID (string)  
|                |                    | • date (dateTime)           |
| sequence       | int                | A unique number for each Coupon Code generated, starting with 1.            |
| state          | CouponCodeState    | The state of the Coupon Code:  
|                |                    | • Not Yet Activated  
|                |                    | • Activated  
|                |                    | • Redeemed  
|                |                    | • Expired  
|                |                    | • Marked Used  
|                |                    | • Retrieved  
|                |                    | • Invalidated  
|                |                    | • 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateCampaign</td>
<td>Sets the state of an Inactive or Pending Campaign to Active.</td>
</tr>
<tr>
<td>activateCode</td>
<td>Activates a CouponCode.</td>
</tr>
<tr>
<td>cancelCampaign</td>
<td>Cancels a Campaign and all of its existing promotionCodes or couponCodes.</td>
</tr>
<tr>
<td>cloneCampaign</td>
<td>Vindicia best practices recommendation is to use the CashBox GUI interface, rather than the API, to clone a Campaign.</td>
</tr>
<tr>
<td>create1Campaign</td>
<td>Vindicia best practices recommendation is to use the CashBox GUI interface, rather than the API, to create new Campaigns.</td>
</tr>
<tr>
<td>deactivateCampaign</td>
<td>Sets the status of an Active or Pending Campaign to Inactive.</td>
</tr>
<tr>
<td>fetchAllCampaigns</td>
<td>Returns an array of Campaign objects, filtered by Campaign-State, if specified.</td>
</tr>
<tr>
<td>fetchByCampaignId</td>
<td>Loads a Campaign by your Campaign ID.</td>
</tr>
<tr>
<td>fetchByVid</td>
<td>Loads a Campaign by its VID.</td>
</tr>
<tr>
<td>generateCouponCodes</td>
<td>Vindicia best practices recommendation is to use the CashBox GUI interface, rather than the API, to generate Coupon Codes.</td>
</tr>
<tr>
<td>markAllCouponsUsed</td>
<td>Vindicia best practices recommendation is to use the CashBox GUI interface, rather than the API, to mark all Coupons Used.</td>
</tr>
<tr>
<td>retrieveCouponCodes</td>
<td>Fetches previously generated CouponCodes.</td>
</tr>
<tr>
<td>updateCampaign</td>
<td>Vindicia best practices recommendation is to use the CashBox GUI interface, rather than the API, to update a Campaign.</td>
</tr>
<tr>
<td>validateCode</td>
<td>Checks if a Coupon or Promotion may be used.</td>
</tr>
</tbody>
</table>
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

srdf: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srdf. A null srdf returns the complete response.

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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Failed to update SOAP object in DB.</td>
</tr>
<tr>
<td></td>
<td>• Current state of <code>&lt;state&gt;</code> is an invalid state for Campaign activation.</td>
</tr>
<tr>
<td></td>
<td>• Failed to obtain SOAP object from DB.</td>
</tr>
<tr>
<td></td>
<td>• Tried to activateCampaign without doing createCampaign first.</td>
</tr>
<tr>
<td></td>
<td>• Can't activate campaign.</td>
</tr>
</tbody>
</table>

Example

```php
$camp = new Campaign();
$campaignId = '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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Code <code>code</code> doesn't represent a CouponCode.</td>
</tr>
<tr>
<td></td>
<td>• Code <code>code</code> can't be activated: <code>error-description</code>.</td>
</tr>
</tbody>
</table>

**Example**

```php
$campaign = new Campaign();
$response = $campaign->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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
</table>
| 400         | One of the following:  
|             | • Failed to update SOAP object in DB.  
|             | • Campaign is not in the right state for cancellation.  
|             | • Failed to obtain SOAP object from DB.  
|             | • Tried to cancelCampaign without doing createCampaign first.  
|             | • Can't cancel campaign until campaign code generation is complete.  
|             | • Can't cancel campaign: unable to invalidate campaign codes  
|             |   Campaign codes requested = `<requested>`  
|             |   Campaign codes generated = `<generated>`.

**Example**

```php  
$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**

*srd*: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a **method** call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the *srd*. A null *srd* returns the complete response.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Tried to deactivateCampaign without doing createCampaign first.</td>
</tr>
<tr>
<td></td>
<td>• Current state of <code>&lt;state&gt;</code> is an invalid state for Campaign deactivation.</td>
</tr>
<tr>
<td></td>
<td>• Failed to update SOAP object in DB.</td>
</tr>
<tr>
<td></td>
<td>• Failed to obtain SOAP object from DB.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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**

srd: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

status: the (optional) CampaignState of the Campaign(s) you wish to have returned. To fetch all Campaigns, set status to MatchAnyState.

date: 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 date gets the results from 1 through 10.
  - Specifying 2 for date 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**

```php
$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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

  - **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Can't load Campaign with ID input-campaignId.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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**

sdr: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the sdr. A null sdr returns the complete response.

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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Can't load Campaign with VID input-vid.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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**

srd: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Page number must be &gt;= 0 and pageSize must be &gt; 0.</td>
</tr>
<tr>
<td></td>
<td>• Can't load Campaign.</td>
</tr>
<tr>
<td></td>
<td>• Claimed to load a campaign by ID but it has no VID.</td>
</tr>
<tr>
<td></td>
<td>• Can't retrieve campaign codes when campaign state is 'Active.'</td>
</tr>
<tr>
<td></td>
<td>• Can't retrieve campaign codes:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Coupon codes not initialized.</td>
</tr>
<tr>
<td></td>
<td>Number of campaign codes requested not yet set up.</td>
</tr>
<tr>
<td></td>
<td>Number of campaign codes requested is zero.</td>
</tr>
<tr>
<td></td>
<td>Campaign codes requested = <code>&lt;requested&gt;</code>;</td>
</tr>
<tr>
<td></td>
<td>campaign codes generated = <code>&lt;generated&gt;</code></td>
</tr>
</tbody>
</table>

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**

**srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Code <code>input-code</code> doesn't represent a CouponCode.</td>
</tr>
<tr>
<td></td>
<td>• Code <code>input-code</code> can't be activated: <code>error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Code <code>input-code</code> is not redeemable: <code>error-description</code>.</td>
</tr>
</tbody>
</table>

**Example**

```php
$camp = new Campaign();
$response = $camp->validateCode(
    'promo123', // the campaign code
    true       // activate now
);

// check $response
```
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:

**Note:** Use of the forward slash character (/) in identifiers is not allowed. See Merchant Identifiers for more information.
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>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>decimal</td>
<td>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.) Note: Given exchange-rate fluctuations, transactions across currencies might be charged back at amounts that differ from the original amounts.</td>
</tr>
<tr>
<td>caseNumber</td>
<td>string</td>
<td>Your bank’s case number for this Chargeback object, if any.</td>
</tr>
<tr>
<td>currency</td>
<td>string</td>
<td>The ISO 4217 currency code (see <a href="http://www.xe.com/iso4217.htm">www.xe.com/iso4217.htm</a>) of this Chargeback object. This currency applies to the settlement amount (see the amount attribute). The default is USD.</td>
</tr>
<tr>
<td>divisionNumber</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>merchantNumber</td>
<td>string</td>
<td>Your bank’s merchant number, which identifies you as the merchant.</td>
</tr>
<tr>
<td>merchantTransactionId</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>merchantTransactionTimestamp</td>
<td>dateTime</td>
<td>A time stamp that specifies the date and time when the original transaction occurred.</td>
</tr>
<tr>
<td>merchantUserId</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>nameValues</td>
<td>NameValuePair[]</td>
<td>Optional. An array of name–value pairs for the Chargeback object. See Section 10: The NameValuePair Object.</td>
</tr>
<tr>
<td>note</td>
<td>string</td>
<td>Notes on the Chargeback object. Vindicia personnel might make entries here during the dispute process.</td>
</tr>
<tr>
<td>postedTimestamp</td>
<td>dateTime</td>
<td>A time stamp 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 time stamp, will depend on the frequency at which Vindicia downloads chargebacks from your bank or payment processor.</td>
</tr>
<tr>
<td>Data Members</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>presentmentAmount</td>
<td>decimal</td>
<td>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.</td>
</tr>
<tr>
<td>presentmentCurrency</td>
<td>string</td>
<td>The ISO 4217 currency code (see <a href="http://www.xe.com/iso4217.htm">www.xe.com/iso4217.htm</a>) of this transaction at presentment. The default is USD.</td>
</tr>
<tr>
<td>processorReceivedTimestamp</td>
<td>dateTime</td>
<td>A time stamp that specifies the date and time when your bank received the chargeback from the customer.</td>
</tr>
<tr>
<td>reasonCode</td>
<td>string</td>
<td>The reason code reported by your bank for this Chargeback object. Reason codes vary from bank to bank.</td>
</tr>
<tr>
<td>referenceNumber</td>
<td>string</td>
<td>Your bank’s reference number for this Chargeback object, if any.</td>
</tr>
<tr>
<td>status</td>
<td>ChargebackStatus</td>
<td>The current chargeback status in ChargeGuard. A chargeback goes through a life cycle as Vindicia disputes the chargeback on your behalf. See Table 7-3: ChargebackStatus Object Values.</td>
</tr>
<tr>
<td>statusChangedTimestamp</td>
<td>dateTime</td>
<td>A time stamp that specifies the date and time for the last status change.</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
7.2 Chargeback Methods

The following table summarizes the methods for the Chargeback object.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetchByAccount</td>
<td><em>(This method is not in use.)</em></td>
</tr>
<tr>
<td>fetchByCaseNumber and fetchByReferenceNumber</td>
<td>Returns one or more Chargeback objects whose case or reference number matches the input.</td>
</tr>
<tr>
<td>fetchByMerchantTransactionId</td>
<td>Returns one or more Chargeback objects for the transaction whose ID assigned by you (merchantTransactionId) matches the input.</td>
</tr>
<tr>
<td>fetchByStatus</td>
<td>Returns one or more Chargeback objects whose status matches the input.</td>
</tr>
<tr>
<td>fetchByStatusSince</td>
<td>Returns one or more Chargeback objects whose status has changed since the specified time stamp.</td>
</tr>
<tr>
<td>fetchByVid</td>
<td>Returns a Chargeback object whose VID matches the input.</td>
</tr>
<tr>
<td>fetchDelta</td>
<td>Returns the Chargeback objects whose status has changed since this call was last made.</td>
</tr>
<tr>
<td>fetchDeltaSince</td>
<td>Returns the Chargeback objects whose status has changed since the specified time stamp.</td>
</tr>
<tr>
<td>report</td>
<td>Reports a batch of Chargeback objects to ChargeGuard.</td>
</tr>
<tr>
<td>update</td>
<td>Creates or updates a Chargeback object in the Vindicia database.</td>
</tr>
</tbody>
</table>
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

srdf: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load chargebacks by case number input-case-number: No match.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load chargebacks by case number input-case-number: error-description.</td>
</tr>
<tr>
<td></td>
<td>• Must specify a case number to load by!</td>
</tr>
</tbody>
</table>
In addition to those listed in Table 1: Standard Return Codes, `fetchByReferenceNumber` also returns:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load chargebacks by reference number <code>input-reference-number</code>: No match.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load chargebacks by reference number <code>input-reference-number</code>: error-description.</td>
</tr>
<tr>
<td></td>
<td>• Must specify a reference number to load by!</td>
</tr>
</tbody>
</table>

**Example**

```php
// 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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

  - **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
</table>
| 400         | One of the following:  
  - Unable to load chargebacks by merchantTransactionId
    **input-merchantTransactionId**: No match.  
  - Unable to load chargebacks by merchantTransactionId  
    **input-merchantTransactionId**: error-description.  
  - Must specify merchant transaction id to load by!

**Example**

```php
$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**

- **sr**d: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `sr`. A null `sr` returns the complete response.

- **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**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Challenged</td>
<td>Vindicia has submitted rebuttal documents to your payment processor to dispute this chargeback.</td>
</tr>
<tr>
<td>CollectionsNew</td>
<td>An inactive status.</td>
</tr>
<tr>
<td>CollectionsWon</td>
<td>An inactive status.</td>
</tr>
<tr>
<td>CollectionsLost</td>
<td>An inactive status.</td>
</tr>
<tr>
<td>Duplicate</td>
<td>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.</td>
</tr>
<tr>
<td>Expired</td>
<td>The related documents or transaction details you reported were received too late by Vindicia to dispute this chargeback.</td>
</tr>
<tr>
<td>Incomplete</td>
<td>Vindicia has received chargeback information from the payment processor but does not have the original transaction details from you.</td>
</tr>
</tbody>
</table>
Table 7-3 ChargebackStatus Object Values (Continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legitimate</td>
<td>A valid chargeback because the original transaction was truly fraudulent. Vindicia does not represent or dispute legitimate transactions.</td>
</tr>
<tr>
<td>Lost</td>
<td>Vindicia challenged this chargeback but lost the case.</td>
</tr>
<tr>
<td>New</td>
<td>The first chargeback received by Vindicia, which is in the process of deciding how to pursue on your behalf.</td>
</tr>
<tr>
<td>NewSecondChargeback</td>
<td>A second chargeback has been received against a transaction that was initially charged back, disputed, and won.</td>
</tr>
<tr>
<td>Pass</td>
<td>Even though all the documentation is available, Vindicia will not dispute this chargeback because of one or more of the following reasons: The chargeback is less than US$5. Not enough evidence exists for a dispute. Regulations do not allow Vindicia to respond. Vindicia does not recommend taking the dispute to arbitration.</td>
</tr>
<tr>
<td>Retrieval</td>
<td>An incoming retrieval or ticket request.</td>
</tr>
<tr>
<td>Responded</td>
<td>Vindicia has responded to the retrieval or ticket request.</td>
</tr>
<tr>
<td>Represented</td>
<td>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.)</td>
</tr>
<tr>
<td>Won</td>
<td>Vindicia challenged this chargeback, which has been reversed in your favor.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load chargebacks by status input-status: No match.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load chargebacks by status input-status: error-description.</td>
</tr>
<tr>
<td></td>
<td>• Must specify a status to load by!</td>
</tr>
</tbody>
</table>
**Example**

```php
$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);
```
The `fetchByStatusSince` method returns one or more Chargeback objects whose statuses match the input and have changed since the specified time stamp. This call is similar to 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**

- **srdf:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

- **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 time stamp specified in the input.

**Returns**

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

```php
$cb = new Chargeback();
$page = 0;
$pageSize = 50;
// Assume a function is available that returned time stamp 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

srdf: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srdf. A null srdf returns the complete response.

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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load chargebacks by VID input-vid: No match.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load chargebacks by VID input-vid: error-description.</td>
</tr>
<tr>
<td></td>
<td>• Must specify VID to load by!</td>
</tr>
</tbody>
</table>
Example

```php
$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 time stamp 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

srdd: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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

```php
$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 time stamp.

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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`timestamp`: the date and time on or after which a chargeback changed its status.

`endTimestamp`: a time stamp that specifies the date and time before which a chargeback changed its status. If null, CashBox applies only `timeStamp` as the search criterion.

| Note: | To reduce high loads on the system, the maximum span for a `fetchDeltaSince()` call is 61 days. If you want to fetch an extended amount of data, best practice suggests that you do so via a loop that pulls a day's worth of data at a time, with another loop internally that handles fetching each resultant page for that day. |

`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 time stamp 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);
```
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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Error saving transaction: error-description.</td>
</tr>
</tbody>
</table>
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**

`srdf`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

*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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• OK.</td>
</tr>
<tr>
<td></td>
<td>• Chargeback object was unchanged.</td>
</tr>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Failed to save chargeback.</td>
</tr>
<tr>
<td></td>
<td>• Error saving chargeback disposition log entry: <code>error-description</code>.</td>
</tr>
</tbody>
</table>
Example

```php
// 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:

  ```
  Note: Use of the forward slash character (/) in identifiers is not allowed. See Merchant Identifiers for more information.
  ```

- 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, this 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.
# Entitlement Data Members

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

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>Account</td>
<td>The Account object with which this Entitlement object is associated. See Section 1.2: Account Data Members.</td>
</tr>
<tr>
<td>active</td>
<td>Boolean</td>
<td>A Boolean flag that, if set to true, indicates that the Entitlement is currently active.</td>
</tr>
<tr>
<td>autoBillVid</td>
<td>string</td>
<td>The AutoBill VID associated with this Entitlement.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Your description for the Entitlement.</td>
</tr>
<tr>
<td>endTimestamp</td>
<td>dateTime</td>
<td>The date on which the Entitlement will expire, plus a grace period for the final billing transaction. (Blank for no end date.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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). 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 time stamp.</td>
</tr>
<tr>
<td>string</td>
<td></td>
<td>Your AutoBill ID associated with this Entitlement.</td>
</tr>
<tr>
<td>merchantEntitlementId</td>
<td>string</td>
<td>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. <strong>Note:</strong> use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
<tr>
<td>merchantProductId</td>
<td>string</td>
<td>Your Product ID associated with this Entitlement, if any.</td>
</tr>
<tr>
<td>productVid</td>
<td>string</td>
<td>The Product VID associated with this Entitlement, if any.</td>
</tr>
<tr>
<td>startTimestamp</td>
<td>dateTime</td>
<td>The time the entitlement begins.</td>
</tr>
</tbody>
</table>
### 8.2 Entitlement Methods

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fetchByAccount</code></td>
<td>Returns one or more <code>Entitlement</code> objects for the <code>Account</code> object specified in the input.</td>
</tr>
<tr>
<td><code>fetchByEntitlementIdAndAccount</code></td>
<td>Returns the <code>Entitlement</code> object with the entitlement ID for the <code>Account</code> object specified in the input.</td>
</tr>
<tr>
<td><code>fetchDeltaSince</code></td>
<td>Returns one or more <code>Entitlement</code> objects that have changed since the specified time stamp.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>customer_id</th>
<th>entitlement_id</th>
<th>Last Update</th>
<th>Active Till</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jdoe1970</td>
<td>VideoDownloadSpecial</td>
<td>2009-09-18</td>
<td>null</td>
</tr>
<tr>
<td>Jdoe1970</td>
<td>LiveTechSupport</td>
<td>2009-08-23</td>
<td>2009-09-01</td>
</tr>
</tbody>
</table>

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**

$srd$: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the $srd$. A null $srd$ returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>404</td>
<td>Account not found.</td>
</tr>
</tbody>
</table>
Example

```php
$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**

srdf: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Account not specified.</td>
</tr>
<tr>
<td>404</td>
<td>Account not found.</td>
</tr>
</tbody>
</table>
Example

```php
$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 time stamp. 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 time stamp 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**

- `srdd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdd`. A null `srdd` returns the complete response.

- `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.

**Note:** To reduce high loads on the system, the maximum span for a `fetchDeltaSince()` call is 61 days. If you want to fetch an extended amount of data, best practice suggests that you do so via a loop that pulls a day’s worth of data at a time, with another loop internally that handles fetching each resultant page for that day.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Invalid value or values of timestamp, and/or page, and/or page size.</td>
</tr>
</tbody>
</table>

**Example**

```php
$ent = new Entitlement();
$pg = 0;
$pageSize = 200;
$count = 0;
$endTimestamp = '2010-01-02T22:34:32.265Z';
$start = '2010-01-01T22:34:32.265Z';

do {
    $ret = $ent->fetchDeltaSince($start, $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
        }
        // 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 (where supported). 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>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hashType</td>
<td>HashType</td>
<td><em>(This data member is not in use.)</em></td>
</tr>
<tr>
<td>lastDigits</td>
<td>string</td>
<td><strong>Read-only.</strong> 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.</td>
</tr>
<tr>
<td>paymentProvider</td>
<td>string</td>
<td>Gift card processor company that CashBox should contact to check the status of a gift card and redeem the gift card.</td>
</tr>
<tr>
<td>pin</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>pinHash</td>
<td>string</td>
<td><em>(This data member is not in use.)</em></td>
</tr>
<tr>
<td>pinLength</td>
<td>integer</td>
<td><strong>Read-only.</strong> 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.</td>
</tr>
<tr>
<td>product</td>
<td>Product</td>
<td><strong>Read-only.</strong> 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. See Section 13.1: Product Data Members.</td>
</tr>
</tbody>
</table>
| 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().
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)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>GiftCardStatus</td>
<td><strong>Read-only.</strong> 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. See the GiftCardStatus Subobject.</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| nameValues       | NameValue-Pair[] | An array of name–value pairs.  
                      (This data member is not in use.) |
| providerResponseCode | string     | Code that CashBox received from the gift card processor when it set the current status.  
                      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.  
                      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.  
                      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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• You may redeem the GiftCard whose status you checked.</td>
</tr>
<tr>
<td></td>
<td>• An earlier call to reverse redemption of a GiftCard was successful and you can redeem the GiftCard again.</td>
</tr>
<tr>
<td>Deactive</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• The gift card processor rejected the GiftCard.</td>
</tr>
<tr>
<td></td>
<td>• A call to redeem a GiftCard was unsuccessful.</td>
</tr>
<tr>
<td></td>
<td>• An attempt to reverse a redemption on a GiftCard was not authorized by the gift card processor.</td>
</tr>
<tr>
<td>Redeemed</td>
<td>Your call for redemption of a GiftCard was successful.</td>
</tr>
<tr>
<td>RedemptionPending</td>
<td>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.</td>
</tr>
<tr>
<td>Suspended</td>
<td><em>(This status is not in use.)</em></td>
</tr>
<tr>
<td>Unknown</td>
<td>CashBox cannot determine the status of the GiftCard for one of two reasons:</td>
</tr>
<tr>
<td></td>
<td>• It could not contact the processor.</td>
</tr>
<tr>
<td></td>
<td>• It could not interpret a response from the processor.</td>
</tr>
</tbody>
</table>
9.3 GiftCard Methods

The following table lists and summarizes the methods for the GiftCard object.

Table 9-4 GiftCard Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reverse</td>
<td>Reverses status of a GiftCard from a previous redemption attempt.</td>
</tr>
<tr>
<td>statusInquiry</td>
<td>Returns the latest status of a GiftCard</td>
</tr>
<tr>
<td></td>
<td>Use this method to determine whether a GiftCard can be redeemed.</td>
</tr>
</tbody>
</table>
**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**

`srdf`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

*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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Failed to retrieve gift card <em>error-description</em>.</td>
</tr>
<tr>
<td></td>
<td>• Reversal attempt failed <em>error-description</em>.</td>
</tr>
<tr>
<td></td>
<td>• Reversal attempt rejected by GiftCard Processor.</td>
</tr>
</tbody>
</table>
Example

```php
$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**

**srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Gift Card could not be saved prior to status. Status Inquiry attempt failed error-description.</td>
</tr>
</tbody>
</table>
$gc = new GiftCard();

    // set the PIN provided by the customer
$gc->setPin('683092298403');
$gc->setPaymentProvider('your gift card processor');

    // 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 you create and associate with text string values. You use these name-value pairs to store custom data for your own internal tracking purposes, or to store CashBox specific data for defined CashBox purposes. Both the name and the value components of a name-value pair are each limited to 255 characters. Both custom and predefined name-value pairs are restricted to the 255 character limit. If you create a custom name-value pair in which one or both components exceeds the 255 character limit, CashBox returns a 400-series error/return code, to ensure that your data is not lost without warning.

Note: Currently some Methods still silently truncate name value pairs in excess of 255 characters.

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, Transaction, and AutoBill, CashBox automatically generates certain name-value pairs, designated by the prefix vin:. These pairs are listed and defined in the nameValue data member table for the specified object. (For more information, see the chapter “Working with Name-Value Pairs” in the CashBox Programming Guide.)
10.1 **NameValuePair Data Members**

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

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The name for the name/value pair.</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>The value for the name/value pair.</td>
</tr>
</tbody>
</table>
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**

- `srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- `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, 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Invalid Type.</td>
</tr>
<tr>
<td></td>
<td>• Failed to retrieve name/value names.</td>
</tr>
</tbody>
</table>
$nvp = new NameValuePair();
$response = $nvp->fetchNameValueNames('Account');

if ($response['returnCode'] == 200) {
    $names = $response['names'];
    foreach ($names as $name) {
        print "$name
";
    }
} 
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, 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

```php
$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 includes 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 on, 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.

---

**Note:** Use of the forward slash character (/) in identifiers is not allowed. See [Merchant Identifiers](#) for more information.
# 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

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountHolderName</td>
<td>string</td>
<td>The name of the account holder.</td>
</tr>
<tr>
<td>active</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>amazon</td>
<td>Amazon</td>
<td>A subobject that specifies the details of an Amazon payment. You must populate this attribute if you set the type attribute (described later in this table) to Amazon. See the Amazon Subobject</td>
</tr>
<tr>
<td>applePay</td>
<td>ApplePay</td>
<td>A subobject that specifies the details of payment accepted via ApplePay. You must specify this attribute if you set the type attribute to ApplePay. See the ApplePay Subobject.</td>
</tr>
<tr>
<td>billingAddress</td>
<td>Address</td>
<td>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. See the The Address Object.</td>
</tr>
<tr>
<td>boleto</td>
<td>Boleto</td>
<td>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 to Boleto. See the Boleto Subobject.</td>
</tr>
<tr>
<td>carrierBilling</td>
<td>CarrierBilling</td>
<td>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. See the CarrierBilling Subobject.</td>
</tr>
<tr>
<td>creditCard</td>
<td>CreditCard</td>
<td>A subobject that specifies the details of a credit card. You must populate this attribute if you set the type attribute to CreditCard. See the CreditCard Subobject.</td>
</tr>
<tr>
<td>currency</td>
<td>string</td>
<td>The ISO 4217 currency code (see <a href="http://www.xe.com/iso4217.htm">www.xe.com/iso4217.htm</a>) 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. 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.</td>
</tr>
</tbody>
</table>
## Table 11-1 PaymentMethod Object Data Members (Continued)

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customerDescription</td>
<td>string</td>
<td>Optional. The customer’s description for this payment method.</td>
</tr>
</tbody>
</table>
| customerSpecifiedType | string          | A customer-specified arbitrary string that describes the payment method type. This field is optional for most credit cards, but required for the following card types, which must be specified exactly as listed:  
- Switch  
- Solo  
- Dankort  
- Laser, and  
- CarteBleue. |
| 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.        |
| 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. |
| extendedVerification | ExtendedVeri-    | Data needed to enable extended verification systems, such as 3D Secure.                                                                   |
|                   | fication        | See the ExtendedVerification Subobject.                                                                                                         |
| externalbilling   | ExternalBilling | A subobject that specifies the details of an external billing system. You must specify this attribute if you set the type attribute to ExternalBilling. |
|                   | string          | The externalToken data member accepts and stores the externally provided token or identifier for payment methods, usually credit cards that use an external abstraction vendor like a card or network token. Support is limited to certain payment processors and method types (for example, Edgil and Credit Card). When set, the value passed is used to represent this paymentMethod when presenting to your processor or gateway. |
| googlePay         | GooglePay       | Details of payment accepted via GooglePay.                                                                                                  |
|                   |                 | See the GooglePay Subobject.                                                                                                                   |
| hostedPage        | HostedPage      | A subobject that contains the details of a payment accepted or applied using payment provider billing pages.                                    |
|                   |                 | **Note:** Your customer’s Account must exist before any Hosted Page related call references that Account.                                     |
|                   |                 | 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. |
|                   |                 | See the MerchantAcceptedPayment Subobject.                                                                                                   |
### Table 11-1 PaymentMethod Object Data Members (Continued)

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>merchantPaymentMethodId</td>
<td>string</td>
<td>Your unique identifier for this PaymentMethod object. Once you’ve created this object, you may refer to it with this identifier. <strong>Note:</strong> use of the forward slash character (/) in merchant identifiers is not allowed. See <a href="#">Merchant Identifiers</a> for more information.</td>
</tr>
<tr>
<td>militaryStar</td>
<td>MilitaryStar</td>
<td>A subobject that specifies the details of a Military Star card. You must specify this attribute if you set the type attribute to MilitaryStar. See the MilitaryStar Subobject.</td>
</tr>
</tbody>
</table>
| nameValues           | NameValuePair[]    | Optional. An array of name–value pairs that provides additional information on the PaymentMethod object, as follows:

  - 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.
  - A name–value pair with Name: issueNumber. The value for issueNumber is the issue number on the customer’s Switch or Solo card.
  - 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.

See [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. See the PayPal Subobject.                                                                                                                      |
| skrill               | Skrill             | The Skrill account details for a Payment Method of type Skrill. See the Skrill 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).

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.

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.

| 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. See Section 17.1: Token Data Members.                                                                                                     |
Table 11-1 PaymentMethod Object Data Members *(Continued)*

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>PaymentMethod-Type</td>
<td><strong>Required.</strong> 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. See the PaymentMethodType Subobject.</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
11.2 PaymentMethod Subobjects

The PaymentMethod object has several subobjects:

- Amazon Subobject
- Boleto Subobject
- CarrierBilling Subobject
- CreditCard Subobject
- DirectDebit Subobject
- ECP Subobject
- ExtendedCardAttributes Subobject
- ExtendedVerification Subobject
- ExternalBilling Subobject
- HostedPage Subobject
- MerchantAcceptedPayment Subobject
- MilitaryStar Subobject
- PaymentMethodType Subobject
- PayPal Subobject
- PhoneNumber Subobject
- PriceCriteria Subobject
- Skrill Subobject
- ApplePay Subobject
- GooglePay Subobject

Amazon Subobject

List details for an Amazon payment.

Table 11-2 Amazon Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>billingAgreementId</td>
<td>String</td>
<td>The Amazon billing agreement identifier. This value is returned by the OffAmazonPayments getAmazonBillingAgreementId API call and should be passed to Vindicia here.</td>
</tr>
<tr>
<td>buyerName</td>
<td>String</td>
<td>The name Amazon associates with the billing agreement identifier.</td>
</tr>
<tr>
<td>buyerEmail</td>
<td>String</td>
<td>The email Amazon associates with the billing agreement identifier.</td>
</tr>
<tr>
<td>physicalAddress</td>
<td>Address</td>
<td>The address Amazon associates with the billing agreement identifier.</td>
</tr>
</tbody>
</table>
Table 11-2  Amazon Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>billingAddress</td>
<td>Address</td>
<td>The billing address Amazon associates with the buyer’s payment method.</td>
</tr>
<tr>
<td>address_verification_code</td>
<td>String</td>
<td>The address verification code Amazon received when validating the buyer’s payment method.</td>
</tr>
</tbody>
</table>

**Boleto Subobject**

Lists details for a Boleto Bancario payment.

Table 11-3  Boleto Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| fiscalNumber         | string    | The fiscal number that appears on the customer’s Boleto Bancario payment slip. This number, formally called Casadastro de Pessoas, is formatted in a specific pattern (modulo 11).
|                      |           | **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 11-4  CarrierBilling Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>currency</td>
<td>string</td>
<td>ISO 4217 Currency Code for either the <code>static_price_inc_salestax</code>, or the <code>dynamic_target_price</code>. (For dynamic pricing, the customer currency will be determined by the customer region/countryCode.)</td>
</tr>
<tr>
<td>encodedPhoneNumber</td>
<td>string</td>
<td>The (read only) payment provider-encoded phone number used in the Transaction.</td>
</tr>
<tr>
<td>paymentProvider</td>
<td>PaymentProvider</td>
<td>The payment provider selected for the Transaction. (CashBox currently supports BOKU.) See Section 12.1: PaymentProvider Data Members.</td>
</tr>
</tbody>
</table>
Table 11-4  CarrierBilling Object Data Members  (Continued)

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phoneNumber</td>
<td>PhoneNumber</td>
<td>The customer phone number used for the payment. See the PhoneNumber Subobject.</td>
</tr>
<tr>
<td>priceCriteria</td>
<td>PriceCriteria</td>
<td>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. See the PriceCriteria Subobject.</td>
</tr>
<tr>
<td>carrierId</td>
<td>string</td>
<td>The identifier for the carrier, either read from the phone or entered by the customer.</td>
</tr>
</tbody>
</table>

CreditCard Subobject

Lists details for a Credit Card.

Table 11-5  CreditCard Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>string</td>
<td>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. 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: An encrypted value of the credit-card account number in the accountHash field (see the item below), or 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).</td>
</tr>
<tr>
<td>accountLength</td>
<td>int</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table 11-5CreditCard Object Data Members (Continued)

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auResponseCode</td>
<td>string</td>
<td>Status code received from the payment processors for Account Updater changes.</td>
</tr>
<tr>
<td>auResponseMessage</td>
<td>string</td>
<td>Status code Description.</td>
</tr>
<tr>
<td>bin</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>extendedCardAttributes</td>
<td>ExtendedCardAttributes</td>
<td>Enhanced authentication details returned from Payment Provider. See the ExtendedCardAttributes Subobject.</td>
</tr>
<tr>
<td>expirationDate</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>hashType</td>
<td>HashType</td>
<td>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.</td>
</tr>
<tr>
<td>accountHash</td>
<td>string</td>
<td>A hash of the full account number. Remove any non-numeric characters prior to hashing. If the account number is provided, this may be left blank and the hash will be calculated by Vindicia. This value is not returned in fetches.</td>
</tr>
<tr>
<td>lastAuRequestDate</td>
<td>dateTime</td>
<td>Date of the last Account Updater request or CardRefresh registration.</td>
</tr>
<tr>
<td>lastDigits</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>lastUpdateDate</td>
<td>dateTime</td>
<td>Date of the last change to the credit card information.</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
**DirectDebit Subobject**

Lists details for a Direct Debit account.

Table 11-6  DirectDebit Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>string</td>
<td>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 <strong>must</strong> 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. 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.</td>
</tr>
<tr>
<td>accountHash</td>
<td>string</td>
<td>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 747417f2206148a3118d0f3adfd2b5e4139baac. Specify this string only if you are <strong>not</strong> specifying the full account number for security reasons when reporting transactions to Vindicia for ChargeGuard. You need not specify this field when using the associated payment method in a Transaction processed through CashBox.</td>
</tr>
<tr>
<td>accountLength</td>
<td>int</td>
<td>The number of digits in the full account number. Specify this attribute only if you are <strong>not</strong> specifying the full account number but are specifying the accountHash field only for security reasons when reporting transactions to Vindicia for ChargeGuard. You need not specify this field for the associated payment method for a Transaction processed through CashBox.</td>
</tr>
<tr>
<td>bankSortCode</td>
<td>string</td>
<td>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. You <strong>must</strong> 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.</td>
</tr>
</tbody>
</table>
ECP Subobject

Lists details for an ECP account.

Table 11-7  ECP Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>string</td>
<td>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. <strong>Note:</strong> CashBox does not validate ECP accounts algorithmically, and partially masks the account number when returning it in response to your call.</td>
</tr>
<tr>
<td>accountHash</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>accountLength</td>
<td>integer</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 11-7 ECP Object Data Members  *(Continued)*

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountType</td>
<td>AccountType</td>
<td>The type of bank account that issues this electronic check. The allowed values are ConsumerChecking, ConsumerSavings, and CorporateChecking. Note: This field is required.</td>
</tr>
<tr>
<td>allowedTransactionType</td>
<td>ECPTransactionType</td>
<td>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.</td>
</tr>
<tr>
<td>hashType</td>
<td>HashType</td>
<td>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.</td>
</tr>
<tr>
<td>lastDigits</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>routingNumber</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
ExtendedCardAttributes Subobject

This object is read-only, and lists auth response details returned from your Payment Provider.

Table 11-8 ExtendedCardAttributes Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affluent</td>
<td>int</td>
<td>Possible values: Y (true), N (false), U (undefined). Applicable Processors: CPT, Little.</td>
</tr>
<tr>
<td>CardDescription</td>
<td>string</td>
<td>The returned description for the card. Applicable Processors: Little, MeS.</td>
</tr>
<tr>
<td>CommercialCard</td>
<td>int</td>
<td>Possible values: Y (true), N (false), U (undefined). Applicable Processors: CPT, Little.</td>
</tr>
<tr>
<td>ConsumerCard</td>
<td>int</td>
<td>Possible values: Y (true), N (false), U (undefined). Applicable Processor: Little.</td>
</tr>
<tr>
<td>CountryOfIssuance</td>
<td>string</td>
<td>Possible values: USA, etc. Applicable Processors: CPT, Little.</td>
</tr>
<tr>
<td>CreditCard</td>
<td>int</td>
<td>Possible values: Y (true), N (false), U (undefined). Applicable Processors: Little, MeS, Chase.</td>
</tr>
<tr>
<td>DebitCard</td>
<td>int</td>
<td>Possible values: Y (true), N (false), U (undefined). Applicable Processors: Little, MeS, Chase.</td>
</tr>
<tr>
<td>DurbinRegulated</td>
<td>int</td>
<td>Possible values: Y (true), N (false), U (undefined). Applicable Processor: CPT.</td>
</tr>
<tr>
<td>GiftCard</td>
<td>int</td>
<td>Possible values: Y (true), N (false), U (undefined). Applicable Processors: Little, MeS.</td>
</tr>
<tr>
<td>HealthcareCard</td>
<td>int</td>
<td>Possible values: Y (true), N (false), U (undefined). Applicable Processors: CPT, Little, MeS.</td>
</tr>
<tr>
<td>MassAffluent</td>
<td>int</td>
<td>Possible values: Y (true), N (false), U (undefined). Applicable Processor: Little.</td>
</tr>
<tr>
<td>PayrollCard</td>
<td>int</td>
<td>Possible values: Y (true), N (false), U (undefined). Applicable Processors: CPT, Little.</td>
</tr>
<tr>
<td>PINlessDebitCard</td>
<td>int</td>
<td>Possible values: Y (true), N (false), U (undefined). Applicable Processor: CPT.</td>
</tr>
<tr>
<td>PrepaidCard</td>
<td>int</td>
<td>Possible values: Y (true), N (false), U (undefined). Applicable Processors: CPT, Little, MeS.</td>
</tr>
<tr>
<td>Reloadable</td>
<td>int</td>
<td>Possible values: Y (true), N (false), U (undefined). Applicable Processor: Little.</td>
</tr>
</tbody>
</table>
### ExtendedVerification Subobject

Details for an ExtendedVerification payment method.

#### Table 11-9 ExtendedVerification Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>browserAcceptHeader</td>
<td>String</td>
<td>Contents of the customer browser HTTP Accept header.</td>
</tr>
<tr>
<td>browserUserAgentHeader</td>
<td>String</td>
<td>Contents of the customer browser HTTP User-Agent header.</td>
</tr>
<tr>
<td>returnUrl</td>
<td>String</td>
<td>URL merchant wants customer redirected to after verification is complete.</td>
</tr>
</tbody>
</table>

### ExternalBilling Subobject

List details for an ExternalBilling payment method.

#### Table 11-10 ExternalBilling Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BillReceiver</td>
<td>Enum</td>
<td>The payment provider. In this case the payment provider is one of the Amdocs Billing systems.—CES, Ensemble, or Optima. Currently CES is the only billing system implemented in the integration.</td>
</tr>
<tr>
<td>PaymentChannelId</td>
<td>String</td>
<td>The customer paymentChannelId in the billing system. Best practice recommends that this is the same as the merchantPaymentMethodId in the Payment-Method Object. (See PaymentMethod Data Members).</td>
</tr>
<tr>
<td>FinancialAccountID</td>
<td>String</td>
<td>The customer financial account ID in the billing system.</td>
</tr>
</tbody>
</table>
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 11-11 HostedPage Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>countryCode</td>
<td>string</td>
<td>The ISO 3166 ( alpha-2 ) country code for customer’s location, \</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: The combination country+processorPaymentMethodId+merchantId must be set at GlobalCollect</td>
</tr>
<tr>
<td>language</td>
<td>string</td>
<td>Required. The ISO 639-1 language matrix code for the payment pages.</td>
</tr>
<tr>
<td>paymentProvider</td>
<td>PaymentProvider</td>
<td>The payment provider selected for the Transaction. (CashBox supports GlobalCollect.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Section 12.1: PaymentProvider Data Members.</td>
</tr>
</tbody>
</table>
**MerchantAcceptedPayment Subobject**

Lists details for a payment entered manually by a merchant.

### Table 11-12 MerchantAcceptedPayment Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>string</td>
<td>The full account number.</td>
</tr>
<tr>
<td>accountHash</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>

**processorPaymentMethodId**

A string representing the payment method to use for the Transaction. (These values correspond to GlobalCollect’s payment product ID.) CashBox supports the following values:

- Moneybookers: 843
- Paysafecard: 830
- Ukash: 1400
- Direct Debit (Germany): 702
- Recurring Direct Debit (Germany): 712
- Direct Debit (Austria): 703
- Recurring Direct Debit (Austria): 713
- Direct Debit (Netherlands): 701
- Recurring Direct Debit (Netherlands): 711
- Direct Debit (Spain): 709
- Recurring Direct Debit (Spain): 719
- PayPal: 840
- iDEAL: 809
- Sofortuberweisung: 836
- Yandex: 849
- Webmoney: 841
- CashU: 845
- Alipay: 861

**returnUrl**

A string representing the URL to which you would like customers to be redirected after they have successfully completed the HostedPage transaction. (This is often your confirmation page.)
### Table 11-12 MerchantAcceptedPayment Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountLength</td>
<td>string</td>
<td>Length of the total account number. If the full account number is submitted, this field may be left blank, and CashBox will calculate it.</td>
</tr>
<tr>
<td>amount</td>
<td>decimal</td>
<td>The amount paid by a customer. This value must be 0 for PaymentMethods attached to AutoBills. <strong>Note:</strong> MerchantAcceptedPayment PaymentMethods may be attached to AutoBills to indicate that the customer should be invoiced (rather than automatically charged).</td>
</tr>
<tr>
<td>currency</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>hashType</td>
<td>HashType</td>
<td>The algorithm used to hash the account number. If this value is not provided, CashBox will use assume SHA1.</td>
</tr>
<tr>
<td>lastDigits</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>note</td>
<td>string</td>
<td>An optional memo regarding the payment made.</td>
</tr>
<tr>
<td>paymentId</td>
<td>string</td>
<td>The ID of the payment accepted by the merchant.</td>
</tr>
<tr>
<td>paymentType</td>
<td>string</td>
<td>The type of payment accepted by the merchant.</td>
</tr>
<tr>
<td>timestamp</td>
<td>dateTime</td>
<td>The time that payment occurred.</td>
</tr>
<tr>
<td>token</td>
<td>Token</td>
<td>The Token associated with the amount (if this is a Token-based AutoBill). See Section 17.1: Token Data Members.</td>
</tr>
</tbody>
</table>

### MilitaryStar Subobject

Lists details for a Military Star payment.

### Table 11-13 MilitaryStar Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>string</td>
<td>MilitaryStar account number.</td>
</tr>
<tr>
<td>cid</td>
<td>string</td>
<td>Customer Identification Number of the account holder.</td>
</tr>
</tbody>
</table>
### 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 11-14 PaymentMethodType Object Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>The payment method is Amazon.</td>
</tr>
<tr>
<td>Boleto</td>
<td>The payment method is Boleto Bancário.</td>
</tr>
<tr>
<td>CarrierBilling</td>
<td>The payment method is Carrier Billing.</td>
</tr>
<tr>
<td>CreditCard</td>
<td>The payment method is credit card.</td>
</tr>
<tr>
<td>DirectDebit</td>
<td>The payment method is direct debit. CashBox supports direct debit payment methods in the Netherlands, Germany, and Austria.</td>
</tr>
<tr>
<td>ECP</td>
<td>The payment method is electronic check through the ACH network.</td>
</tr>
<tr>
<td>ExternalBilling</td>
<td>A subobject that specifies the details of an external billing system.</td>
</tr>
<tr>
<td>HostedPage</td>
<td>The payment method is HostedPage.</td>
</tr>
<tr>
<td>MerchantAcceptedPayment</td>
<td>The payment is manually entered by the merchant.</td>
</tr>
<tr>
<td>PayPal</td>
<td>The payment method is PayPal.</td>
</tr>
<tr>
<td>Token</td>
<td>The payment method is Tokens.</td>
</tr>
</tbody>
</table>

### PayPal Subobject

Lists details for a PayPal account.

#### Table 11-15 PayPal Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancelUrl</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>hashType</td>
<td>HashType</td>
<td><em>(This data member is not in use.)</em></td>
</tr>
<tr>
<td>password</td>
<td>string</td>
<td><em>(This data member is not in use.)</em></td>
</tr>
<tr>
<td>passwordHash</td>
<td>string</td>
<td><em>(This data member is not in use.)</em></td>
</tr>
<tr>
<td>payerId</td>
<td>string</td>
<td>Unique PayPal customer account identification number in PayPal ExpressCheckout.</td>
</tr>
<tr>
<td>returnUrl</td>
<td>string</td>
<td>The URL to which you would like customers to be redirected after they have successfully completed payment transactions on the PayPal site.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>(This is often your confirmation page, on which the customer confirms the order and payment or the billing agreement.)</em></td>
</tr>
</tbody>
</table>
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 11-16 PhoneNumber Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>areaCode</td>
<td>string</td>
<td>Required. The area code segment of the phone number.</td>
</tr>
<tr>
<td>countryCode</td>
<td>string</td>
<td>The Country Code segment of the phone number.</td>
</tr>
<tr>
<td>extension</td>
<td>string</td>
<td>The phone number’s extension.</td>
</tr>
<tr>
<td>localNumber</td>
<td>string</td>
<td>Required. The local number (excluding extension).</td>
</tr>
<tr>
<td>phoneType</td>
<td>PhoneType</td>
<td>The type of phone.</td>
</tr>
<tr>
<td>rawInput</td>
<td>string</td>
<td>Raw, unfiltered data input by customer.</td>
</tr>
</tbody>
</table>

Note: PayPal has one email address, `payerId`, that identifies the PayPal account of the 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>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>countryCode</td>
<td>string</td>
<td>ISO 3166-1 alpha-2 countryCode for customer location. This value will override the CarrierBilling object’s countryCode data member.</td>
</tr>
<tr>
<td>currency</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>A description for the Price Criteria.</td>
</tr>
<tr>
<td>dynamicDeviation</td>
<td>int</td>
<td>The % deviation (+/- 1000) from the target value that is acceptable as a price point selection.</td>
</tr>
<tr>
<td>dynamicMatch</td>
<td>int</td>
<td>The % deviation (+/- 1000) from the target value that is classified as an exact match.</td>
</tr>
<tr>
<td>dynamicPriceMode</td>
<td>DynamicPriceMode</td>
<td>Defines which price point element is matched by the dynamic pricing algorithm.</td>
</tr>
<tr>
<td>dynamicTargetPrice</td>
<td>decimal</td>
<td>The target price in the specified currency for dynamic pricing.</td>
</tr>
<tr>
<td>fwdUrl</td>
<td>string</td>
<td>Overrides both the successful transaction forward-to URL, and the failed transaction forward-to URL.</td>
</tr>
<tr>
<td>merchantServiceIdentifier</td>
<td>string</td>
<td>Your service identifier for the payment provider.</td>
</tr>
</tbody>
</table>
Table 11-17 PriceCriteria Object Data Members  (Continued)

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>paymentProvider</td>
<td>PaymentProvider</td>
<td>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.) See Section 12.1: PaymentProvider Data Members.</td>
</tr>
</tbody>
</table>
| pricePointDeviationPolicy | PricePointDeviationPolicy | The allowed price deviation policy for Carrier-Billing payments using dynamic price selection. PricePointDeviationPolicy may be one of three values:  
  HiPreferred: A solution higher than the target value will be favored over a lower solution.  
  HiOnly: Only solutions higher than the target value will be returned.  
  LowPreferred: A solution lower than the target value will be favored over a higher solution.  
  LowOnly: Only solutions lower than the target value will be returned.  
  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. (Note that use of the forward slash character (/) in merchant identifiers is not allowed.  
  Note: use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information. |

**Skrill Subobject**

Lists details for a Skrill account.

Table 11-18 Skrill Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>billingAgreementId</td>
<td>string</td>
<td>The Skrill 1-Tap billing agreement identifier (rec_payment_id).</td>
</tr>
<tr>
<td>cancelUrl</td>
<td>string</td>
<td>The URL to which you would like to redirect customers if they do not approve payment via Skrill.</td>
</tr>
</tbody>
</table>
ApplePay Subobject

List details for an ApplePay payment.

Table 11-19 ApplePay Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>paymentInstrumentName</td>
<td>String</td>
<td>The paymentInstrumentName field from PKPayment.</td>
</tr>
<tr>
<td>paymentNetwork</td>
<td>String</td>
<td>The paymentNetwork from PKPayment.</td>
</tr>
<tr>
<td>transactionIdentifier</td>
<td>String</td>
<td>The transactionIdentifier field from PKPayment.</td>
</tr>
<tr>
<td>paymentData</td>
<td>Address</td>
<td>The paymentData field from PKPayment.</td>
</tr>
<tr>
<td>expirationDate</td>
<td>Address</td>
<td>Returns the expiration date of the credit card used for ApplePay. Will be formatted as YYYYMM. For example, July, 2008 will be returned as 200807.</td>
</tr>
</tbody>
</table>

GooglePay Subobject

List details for an GooglePay payment.

Table 11-20 GooglePay Object Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>paymentToken</td>
<td>String</td>
<td>The encrypted paymentToken you receive after calling GooglePaymentDataRequest.</td>
</tr>
</tbody>
</table>
### 11.3 PaymentMethod Methods

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

#### Table 11-21 PaymentMethod Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetchByAccount</td>
<td>Returns one or more <code>PaymentMethod</code> objects whose <code>Account</code> object matches the input.</td>
</tr>
<tr>
<td>fetchByMerchantPaymentMethodId</td>
<td>Returns a <code>PaymentMethod</code> object whose <code>merchantPaymentMethodId</code> matches the input.</td>
</tr>
<tr>
<td>fetchByVid</td>
<td>Returns a <code>PaymentMethod</code> object whose VID matches the input.</td>
</tr>
<tr>
<td>fetchByWebSessionVid</td>
<td>Returns a <code>PaymentMethod</code> object whose <code>WebSessionVid</code> matches the input.</td>
</tr>
<tr>
<td>update</td>
<td>Creates or updates a <code>PaymentMethod</code> object.</td>
</tr>
<tr>
<td>validate</td>
<td>Validates but does not store a <code>PaymentMethod</code> object.</td>
</tr>
</tbody>
</table>
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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Account parameter is required.</td>
</tr>
<tr>
<td>404</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• No PaymentMethods found for account.</td>
</tr>
<tr>
<td></td>
<td>• Account not found.</td>
</tr>
</tbody>
</table>
Example

```php
$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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- `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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Missing required parameter paymentMethodId.</td>
</tr>
<tr>
<td>404</td>
<td>Unable to find requested PaymentMethod: <strong>error-description</strong>.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Missing required parameter 'vid'.</td>
</tr>
<tr>
<td>404</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to find requested PaymentMethod: error-description.</td>
</tr>
<tr>
<td></td>
<td>• Unable to find requested PaymentMethod: No matches.</td>
</tr>
</tbody>
</table>
Example

```php
$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**

`srdf`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Missing required parameter 'vid'.</td>
</tr>
<tr>
<td>404</td>
<td>Unable to find requested PaymentMethod: No matches.</td>
</tr>
</tbody>
</table>

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 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. CashBox uses either $0 or $1 (or other currency) for the amount, depending on the payment method type, credit card type, and payment processor for the validation transaction. If the validation amount is not $0, CashBox might later void this transaction.

Note: CashBox does not support validation for PayPal or Merchant Accepted payment method types.

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 11-22 Validation Process by Payment Method Type

<table>
<thead>
<tr>
<th>Payment Method Type</th>
<th>Validation Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>Amazon Payments methods are automatically validated upon creation/update in CashBox. If you specify Amazon as the Payment Method while creating an AutoBill object, CashBox and Amazon require that the customer agree to the recurring billing on the merchant’s site (using the Amazon-provided widgets or API). Regardless of the specified update/validate parameter, the system confirms the Amazon status for the underlying payment method selected by the customer. Where enabled for you by Amazon, a full Address Verification Response is available. It is provided by Amazon’s latest value for the selected underlying payment method. Amazon does not currently provide for on-demand validation later. <strong>Note:</strong> After initial approval, an existing Billing Agreement can be used for any charges against that customer, one-time or recurring, as long as it is within the established monthly limits (with Amazon). No further validation or approval is required.</td>
</tr>
<tr>
<td>Credit card</td>
<td>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 in the CashBox Portal.</td>
</tr>
<tr>
<td>Direct debit</td>
<td>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. 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.</td>
</tr>
</tbody>
</table>
Electronic check (ECP)  CashBox supports ECP if your payment processor is Chase Paymentech or Litle.  

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.  

(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.)  

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.

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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>261</td>
<td>All active AutoBills were updated. AutoBills which are both expired and Suspended cannot be updated.</td>
</tr>
</tbody>
</table>
| 400         | One of the following:  
  - *Error-description.* (Returned if CashBox cannot map a PaymentMethod object that is passed into a database record.)  
  - Data validation error Failed to create Payment-Type-Specific Payment Record: Credit Card conversion failed: Credit Card failed Luhn check.  
  - Unable to save payment method: *error-description*.  
  - This Credit Card already exists—Policy Violation" (eradicate the newly created but failed one or ensure it is set INACTIVE).  
  You receive this error message when you have the Credit_Card_Constraints merchant option enabled on your site. This message means that someone is attempting to use a credit card that is already being used as the payment method on an existing account. This is not allowed. |
| 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.* |
// 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";
    }
}

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";
    }
}

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**

`srdd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdd`. A null `srdd` returns the complete response.

`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`, 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`. 
**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>VS/CVN policy not evaluated. 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.</td>
</tr>
</tbody>
</table>
| 400         | One of the following:  

  - Invalid parameters: *error-description*.  
  - Error-description.  

  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>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authCurrencyOverride</td>
<td>string</td>
<td>The currencies for which authorization currency may be overridden by USD.</td>
</tr>
<tr>
<td>authExpirationDays</td>
<td>int</td>
<td>The number of days before the payment provider expires authorizations.</td>
</tr>
<tr>
<td>disputeAddress</td>
<td>Address</td>
<td>The payment provider’s dispute address. See Section 3.1: Address Data Members.</td>
</tr>
<tr>
<td>disputeEmail</td>
<td>string</td>
<td>The payment provider’s email address for disputes.</td>
</tr>
<tr>
<td>disputeUri</td>
<td>anyURI</td>
<td>The payment provider’s URI for disputes.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the provider.</td>
</tr>
<tr>
<td>nameValues</td>
<td>NameValuePair</td>
<td>An optional array of name-value pairs to associate with the payment provider.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Section 10: The NameValuePair Object.</td>
</tr>
</tbody>
</table>
12.2 PaymentProvider Methods

The following table summarizes the methods for the PaymentProvider object.

Table 12-2  PaymentProvider Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataRequest</td>
<td>Performs a generic query on the PaymentProvider object.</td>
</tr>
<tr>
<td>fetchByName</td>
<td>Loads a PaymentProvider object by name.</td>
</tr>
</tbody>
</table>
**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**

- **srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- **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**

```perl
$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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- **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**

```php
$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.

**Note:** Use of the forward slash character (/) in identifiers is not allowed. See Merchant Identifiers for more information.

A product can be a single item or a bundled collection of multiple products. For example, a product can be a monthly magazine subscription, or it can 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 associated entitlements. (See also Section 3.1: Creating Products in the CashBox Programming Guide.)
- Define a bundled product by 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 purchase products that grant their account token credits. Tokens can be used as currency in proprietary transaction systems (such as the purchase of a sword in an online game), or they can 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>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>billingStatementIdentifier</code></td>
<td>string</td>
<td><strong>Optional.</strong> The transaction description on the customer’s billing statement that is sent by the bank when the customer is charged through this <code>Product</code> object. This field’s value and format are set by your payment processor; consult with Vindicia Client Services before setting the value. If GlobalCollect, Chase Paymentech, MeS, or Litle is your payment processor, see Appendix A: Custom Billing Statement Identifier Requirements in the <em>CashBox Programming Guide</em>. <strong>Note:</strong> If this identifier is also defined in a <code>BillingPlan</code> object associated with the <code>AutoBill</code> object for this <code>Product</code> object, the billing statement identifier on <code>BillingPlan</code> takes precedence.</td>
</tr>
<tr>
<td><code>brTaxCategory</code></td>
<td>string</td>
<td>The Brazilian tax category for this item.</td>
</tr>
<tr>
<td><code>bundledProducts</code></td>
<td>Product</td>
<td>Zero or more products “bundled” or grouped with this <code>Product</code>.</td>
</tr>
<tr>
<td><code>creditGranted</code></td>
<td>Credit</td>
<td>The credit(s) to be granted upon purchase of this <code>Product</code>. See the <code>Credit Subobject</code>.</td>
</tr>
<tr>
<td><code>defaultBillingPlan</code></td>
<td>BillingPlan</td>
<td><strong>Optional.</strong> Recurring pricing is governed by this attribute if a billing plan is not explicitly associated with the <code>AutoBill</code> object for this <code>Product</code> object.</td>
</tr>
<tr>
<td><code>defaultRatePlan</code></td>
<td>RatePlan</td>
<td><strong>Optional.</strong> A default <code>RatePlan</code> for the <code>Product</code>.</td>
</tr>
<tr>
<td><code>descriptions</code></td>
<td>ProductDescription</td>
<td><strong>Optional.</strong> Zero or more language/product description pairs. <strong>Note:</strong> In the absence of a product description, the <code>merchantProductId</code> will be used.</td>
</tr>
<tr>
<td><code>endOfLifeTimestamp</code></td>
<td>dateTime</td>
<td><strong>Optional.</strong> A time stamp that specifies the expiration date for this <code>Product</code> object. Use this attribute to filter your product list, and present only those products with future expiration dates as currently available for subscription. (This attribute is for your information only, and does not affect CashBox operations.)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>string</td>
<td>A description of this product written in language. A free-form string of less than 256 characters.</td>
</tr>
<tr>
<td>language</td>
<td>string</td>
<td>The language in which the product description is written.</td>
</tr>
</tbody>
</table>

ProductPrice Subobject

Lists a currency and/or Token value for the product.

Table 13-3 ProductPrice Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>decimal</td>
<td>Value of the currency amount.</td>
</tr>
<tr>
<td>currency</td>
<td>string</td>
<td>ISO 4217 currency code to be used for this ProductPrice Amount. Defaults to USD.</td>
</tr>
<tr>
<td>token</td>
<td>Token</td>
<td>Details of pricing using tokens.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>string</td>
<td>Product is currently active (available to the customer).</td>
</tr>
<tr>
<td>Suspended</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
## 13.3 Product Methods

The following table summarizes the methods for the Product object.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetchAll</td>
<td>Returns all the Product objects.</td>
</tr>
<tr>
<td>fetchByAccount</td>
<td>Returns one or more Product objects whose Account object matches the input.</td>
</tr>
<tr>
<td>fetchByMerchantEntitlementId</td>
<td>Returns all the Product objects whose entitlement ID assigned by you (merchantEntitlementId) matches the input.</td>
</tr>
<tr>
<td>fetchByMerchantProductId</td>
<td>Returns the Product object whose merchantProductId matches the input.</td>
</tr>
<tr>
<td>fetchByVid</td>
<td>Returns a Product object whose VID matches the input.</td>
</tr>
<tr>
<td>update</td>
<td>Creates or updates a Product object.</td>
</tr>
</tbody>
</table>
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

srdf: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srdf. A null srdf returns the complete response.

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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>404</td>
<td>No products found for merchant.</td>
</tr>
</tbody>
</table>
Example

```php
$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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>404</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load account to search by: <code>error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load account by: No matches.</td>
</tr>
</tbody>
</table>
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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Must specify entitlement id.</td>
</tr>
<tr>
<td>404</td>
<td>Could not load product for entitlement id <code>input-merchant-EntitlementId</code>.</td>
</tr>
</tbody>
</table>

**Example**

```php
// 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**

**srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the **srd**. A null **srd** returns the complete response.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load product by sku <strong>input-merchantProductId</strong>: No match.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load product by merchantProductId <strong>input-merchantProductId</strong>: error-description.</td>
</tr>
<tr>
<td></td>
<td>• Must specify merchantProductId to load by!</td>
</tr>
</tbody>
</table>

**Example**

```
$prodMerchantId = '5w3320dj';

// Create a SOAP caller product object
$product = new Product();
$response = $product->fetchByMerchantProductId($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**

- `srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- `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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load product by VID <code>input-vid: error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Must specify VID to load by!</td>
</tr>
<tr>
<td>404</td>
<td>Unable to load product by VID <code>input-vid: No match</code>.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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

* `srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- `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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• <code>Error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• CashBox encountered a general error while mapping the input <code>Product</code> object to the CashBox database.</td>
</tr>
<tr>
<td></td>
<td>• Unable to save product: <code>error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Unable to retrieve saved product: <code>error-description</code>.</td>
</tr>
</tbody>
</table>
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:

**Note:** Use of the forward slash character (/) in identifiers is not allowed. See [Merchant Identifiers](#) for more information.
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

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>string</td>
<td>Optional. A description for the Rate Plan.</td>
</tr>
<tr>
<td>hasEventRecorded</td>
<td>Boolean</td>
<td>Read-only field that indicates whether or not this RatePlan has any Events recorded against it.</td>
</tr>
<tr>
<td>includedUnits</td>
<td>decimal</td>
<td>The number of Rated Units automatically included with each Billing Cycle.</td>
</tr>
<tr>
<td>maximumFee</td>
<td>RatePlanPrice</td>
<td>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. The RatePlanPrice object is an (amount, currency) pair used in a RatePlan, and contains two members: amount: the number of currency units. currency: the ISO 4217 currency code to be used for this Fee.</td>
</tr>
<tr>
<td>merchant-RatePlanId</td>
<td>string</td>
<td>Required. Your unique ID for this Rate Plan. Note: use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
<tr>
<td>minimumFee</td>
<td>RatePlanPrice</td>
<td>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. The RatePlanPrice object is an (amount, currency) pair used in a RatePlan, and contains two members: amount: the number of currency units. currency: the ISO 4217 currency code to be used for this Fee.</td>
</tr>
</tbody>
</table>
The calculation method by which this RatePlan will determine the price to bill for a Billing Cycle.

The MultiplyRatedUnitsBy object contains two values:

- EachRespectiveTier: multiplies the number of Events for the Billing Cycle by the ratePrice for the Tier in which they occurred.
- HighestApplicableTier: multiplies the total number of Events for the Billing Cycle by the price for the highest Tier in which any reported Event occurred.

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.

For the same use, HighestApplicableTier would calculate the charge by multiplying 15 units by $1, for a total charge of $15.

### nameValues

Optional. An array of name-value pair items specific to this RatePlan.

See Section 10: The NameValuePair Object.

### ratedUnit

Required. The names for the Unit included in this Rate Plan.

The RatedUnit object contains two data members:

- nameSingular (string): defines the singular name for the Unit, as displayed in CashBox pages, reports, and customer emails.
- namePlural (string): defines the plural name for the Unit.

### ratePlanModel

Required. Defines the mode of use for the RatePlan.

The RatePlanModel object contains one of two values:

- UsageBased: calculates the fee per Billing Cycle based on the number of Rated Units consumed during the Billing Cycle.
- LicenseBased: calculates the fee per Billing Cycle based on a defined number of licenses per Billing Cycle.

**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.
The RatePlan Object 345

Table 14-1  RatePlan Object Data Members  *(Continued)*

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| roundingDecimals  | integer     | Defines the rounding logic for returned Unit values. 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: 
  0: rounds to the nearest integer. (346)
  2: rounds to the nearest hundredth. (346.27)
  -2: rounds to the nearest hundred. (300) |
| status            | RatePlanStatus | Defines the status of the RatePlan: 
  Active: the Rate Plan is available for use.
  Suspended: the Rate Plan is not available for use. |
| tier              | RatePlanTier | An array of pricing levels used in the Rate Plan. 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 time stamp 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

<table>
<thead>
<tr>
<th>Value</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountVid</td>
<td>string</td>
<td>Lists the VID for the Account associated with the Event. (Returned for Event fetches.)</td>
</tr>
<tr>
<td>amount</td>
<td>decimal</td>
<td>The number of Rated Units (as defined by the Rate Plan) used by this Event.</td>
</tr>
<tr>
<td>autoBillItemVid</td>
<td>string</td>
<td>Vindicia’s unique name (VID) for the AutoBillItem.</td>
</tr>
<tr>
<td>autoBillVid</td>
<td>string</td>
<td>Vindicia’s unique name (VID) for the AutoBill.</td>
</tr>
<tr>
<td>billedStatus</td>
<td>BilledStatus</td>
<td>A read-only object of type BilledStatus, which describes whether the Event has been billed, and which includes one of two values: Billed: the Event has been included in a Billing Statement. Unbilled: the Event has not yet been included in a Billing Statement.</td>
</tr>
<tr>
<td>Value</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dateReceived</td>
<td>dateTime</td>
<td>A read-only field which lists the date/time that CashBox received the Event.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Optional. A description of the Event.</td>
</tr>
<tr>
<td>merchantAccountId</td>
<td>string</td>
<td>Lists the merchantAccountId associated with the Event. (Returned for Event fetches.)</td>
</tr>
<tr>
<td>string</td>
<td></td>
<td>Your unique ID for the AutoBill associated with the Event.</td>
</tr>
<tr>
<td>merchantAutoBillItemId</td>
<td>string</td>
<td>Your unique ID for the AutoBillItem associated with the Event. Either merchantAutoBillItemId or autoBillItemVid must be defined for each Event.</td>
</tr>
<tr>
<td>merchantProductId</td>
<td>string</td>
<td>Your unique ID for the Product associated with the Event.</td>
</tr>
<tr>
<td>merchantEventId</td>
<td>string</td>
<td>Optional. Your unique ID for the Event. Each merchantEventId must be unique. If omitted, CashBox will automatically populate this field.</td>
</tr>
<tr>
<td>nameValues</td>
<td>NameValuePair</td>
<td>Optional. An array of name-value pair items specific to this Event. See Section 10: The NameValuePair Object.</td>
</tr>
<tr>
<td>productVid</td>
<td>string</td>
<td>Vindicia's unique name (VID) for the Product.</td>
</tr>
<tr>
<td>eventDate</td>
<td>dateTime</td>
<td>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.</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
**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

<table>
<thead>
<tr>
<th>Value</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountVid</td>
<td>string</td>
<td>Vindicia’s unique identifier for the Account. (Returned for Event fetches.)</td>
</tr>
<tr>
<td>autoBillItemVid</td>
<td>string</td>
<td>Vindicia’s unique identifier for the AutoBillItem.</td>
</tr>
<tr>
<td>autoBillVid</td>
<td>string</td>
<td>Vindicia’s unique identifier for the AutoBill.</td>
</tr>
<tr>
<td>currentTier</td>
<td>string</td>
<td>The Rate Plan Tier to which the summary refers. (The top Tier for which an Event is recorded at the moment of the query.)</td>
</tr>
<tr>
<td>currentTotalRatedUnitsBill</td>
<td>decimal</td>
<td>The total current charge for the Billing Cycle, in the currency specified on the AutoBill.</td>
</tr>
<tr>
<td>eventCount</td>
<td>int</td>
<td>The number of Events included in this summary for this AutoBillItem.</td>
</tr>
<tr>
<td>merchantAccountId</td>
<td>string</td>
<td>Your unique identifier for the Account. This is a read-only field. (Returned for RatedUnitSummary fetches.)</td>
</tr>
<tr>
<td>merchantAutoBillItemId</td>
<td>string</td>
<td>Your unique identifier for the AutoBillItem.</td>
</tr>
<tr>
<td>merchantProductId</td>
<td>string</td>
<td>Your unique identifier for the Product.</td>
</tr>
<tr>
<td>merchantRatePlanId</td>
<td>string</td>
<td>Your unique identifier for the RatePlan.</td>
</tr>
<tr>
<td>productVid</td>
<td>string</td>
<td>Vindicia’s unique identifier for the Product.</td>
</tr>
<tr>
<td>ratedUnit</td>
<td>ratedUnit</td>
<td>The name of the Rated Unit for which the summary is returned.</td>
</tr>
<tr>
<td>ratedUnitTotal</td>
<td>decimal</td>
<td>The total number of all Rated Units included in this summary.</td>
</tr>
<tr>
<td>ratePlanVid</td>
<td>string</td>
<td>Vindicia’s unique identifier for the RatePlan.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Value</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>Required. The descriptive name for the Tier.</td>
</tr>
</tbody>
</table>
| ratePrice  | RatePlanPrice| Required. An array of RatePlanPrice objects, which define the Price (or prices) for this Tier (one price for each currency used). The RatePlanPrice object is an (amount, currency) pair, which contains two data members:
  - amount: the number of currency units.
  - 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:
  - FlatFee: charges the customer a defined price per Tier.
  - PerUnit: charges the customer a defined price per Rated Unit.

  FlatFee defines a stepped pricing structure, in which the customer is charged the ratePrice per Tier.

  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.

  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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deductEvent</td>
<td>Deducts Events from the unbilled Unit balance.</td>
</tr>
<tr>
<td>fetchAll</td>
<td>Fetches all existing Rate Plans.</td>
</tr>
<tr>
<td>fetchByMerchantRatePlanId</td>
<td>Fetches an existing RatePlan by its merchantRatePlanId.</td>
</tr>
<tr>
<td>fetchByVid</td>
<td>Fetches an existing RatePlan by its VID.</td>
</tr>
<tr>
<td>fetchEventById</td>
<td>Fetches an Event by its merchantEventId.</td>
</tr>
<tr>
<td>fetchEventByVid</td>
<td>Fetches an Event by its VID.</td>
</tr>
<tr>
<td>fetchEvents</td>
<td>Fetches all Events by the specified Account, AutoBill, RatePlan, or Product.</td>
</tr>
<tr>
<td></td>
<td>If none of these are specified, fetches all Events.</td>
</tr>
<tr>
<td>fetchUnbilledEvents</td>
<td>Returns unbilled Events for the input AutoBill, Account, RatePlan, or Product.</td>
</tr>
<tr>
<td></td>
<td>If none of these are specified, fetches all unbilled Events.</td>
</tr>
<tr>
<td>fetchUnbilledRatedUnits-Total</td>
<td>Returns an array of RatedUnitSummary objects, broken out by AutoBillItem.</td>
</tr>
<tr>
<td>recordEvent</td>
<td>Records Events against a defined Account, AutoBill, or AutoBillItem.</td>
</tr>
<tr>
<td>reverseEvent</td>
<td>Reverses one or more existing unbilled Events.</td>
</tr>
</tbody>
</table>
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**

- **srdf:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srdf. A null srdf returns the complete response.

- **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**

```php
$rateplan = new RatePlan;
$event = new Event;
$event->setMerchantEventId('rating_129');
$event->set('ab_715');
$event->setAmount(2);
$response = $rateplan->deductEvent(array($event));
// check $response
```
fetchAll

The `fetchAll` method returns all available RatePlan objects.

**Input**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Must specify page and pageSize!</td>
</tr>
</tbody>
</table>

**Example**

```php
$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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

- **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**

```php
$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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

- **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**

```php
$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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

  - **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**

```php
$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**

*srd*: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the *srd*. A null *srd* returns the complete response.

*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**

```php
$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

srdf: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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.

cratePlan: the RatePlan for which Events should be fetched. (Optional.)

startTimestamp: the starting time stamp (lower limit) for the range of Events you wish to retrieve.

cendTimestamp: the ending time stamp (upper limit) for the range of Events you wish to retrieve.

cpage: (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.

cpageSize: (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

srd: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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 time stamp (lower limit) for the range of Events you wish to retrieve.

endTimestamp: the ending time stamp (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**

- **srds**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srds`. A null `srds` returns the complete response.

- **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**

**sr**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the sr. A null sr returns the complete response.

**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**

```php
$rateplan = new RatePlan;
$event = new Event;

$event->setMerchantEventId('rating_129');
$event->set('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

srdf: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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

```php
$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 funds you return to a customer for a previously conducted Transaction, in which the customer paid you for a product or service.

You can generate the Refund object in CashBox in one of two ways

• You can issue a Refund through CashBox to reverse a one-time or recurring Transaction, using either the CashBox API or the Portal.

• You can create a refund to report a transaction that occurred outside the CashBox system. This enables the Vindicia ChargeGuard team to effectively dispute a chargeback against the transaction for which you issued a refund.

Note: Use of the forward slash character (/) in identifiers is not allowed. See Merchant Identifiers for more information.
## 15.1 Refund Data Members

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

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>decimal</td>
<td>A decimal representation of a monetary amount for the refund. Even though <code>amount</code> is a financial unit, its actual value and meaning depend on the value you set in the <code>currency</code> data member. This amount must not exceed that on the <code>Transaction</code> for which you are issuing this refund.</td>
</tr>
<tr>
<td>amountIncludeTax</td>
<td>boolean</td>
<td>Indicates whether the amount is tax-inclusive, or tax-exclusive.</td>
</tr>
<tr>
<td>credit</td>
<td>Credit</td>
<td>A credit(s) reversed as part of the refund. This is a read-only field.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See the <code>Credit Subobject</code>.</td>
</tr>
<tr>
<td>currency</td>
<td>string</td>
<td>The ISO 4217 currency code (see <a href="http://www.xe.com/iso4217.htm">www.xe.com/iso4217.htm</a>) for this transaction.</td>
</tr>
<tr>
<td>merchantRefundId</td>
<td>string</td>
<td>A string of a maximum of 255 characters that represents your unique identifier for this <code>Refund</code> 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. <strong>Note:</strong> use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
<tr>
<td>nameValues</td>
<td>NameValuePair</td>
<td>An optional array of name-value pair items specific to this refund.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See The NameValuePair Object.</td>
</tr>
<tr>
<td>note</td>
<td>string</td>
<td>An optional memo regarding the refund.</td>
</tr>
<tr>
<td>referenceString</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>refundItems</td>
<td>RefundItem</td>
<td>An array of <code>RefundItem</code> objects. May be left empty if the <code>refundDistributionStrategy</code> is not set to <code>SpecifiedItems</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See the <code>RefundItem Subobject</code>.</td>
</tr>
</tbody>
</table>
Refund Data Members

Table 15-1 Refund Object Data Members (Continued)

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| refundDistributionStrategy| string      | Defines how refunds against a payment transaction are processed. If unspecified, a default value of None will be used. The following values are currently supported:  
  - None—applies the refund using the older, Transaction-level approach, which applies a refund against a Transaction as a whole, grouping all individual purchases (or line items) into a single Transaction amount. Note that once you partially refund a particular Transaction using the Transaction-level approach, you must continue to use this same approach on subsequent refunds on that original Transaction. Any attempt at a SpecifiedItems refund against a Transaction in which the older Transaction-level refund has been performed will return an error. If unspecified, a default value of None will be used.  
  - SpecifiedItems—designates a refund for any or all of the purchased items on the original transaction. When applying the refund, you can either specify TaxOnly for the item, or you can specify an amount to affect a full or partial refund at the item level. If you specify an amount, it cannot exceed the original TransactionItem amount (minus any discounts, and minus the sum of all prior refunds against this item). Also, when you specify items, you must pass RefundItem with the details of how to refund each item—if you select either None or RemainingBalance, you do not need to provide RefundItems.  
  - RemainingBalance—refunds the balance remaining (at the line item-level) for the original Transaction. |
| status                    | RefundStatus| Status of the refund.  
  - Reported—refund has been issued but has not been processed by the payment processor/issuer.  
  - Processing—refund call has been requested for a transaction that has not yet been captured. The refund will start processing after capture (refunds are batched and sent daily).  
  - Failed—payment processor has rejected the refund.  
  - Complete—refund has been sent and processed by payment processor. |
| timestamp                 | dateTime    | A time stamp 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. |
Table 15-1  Refund Object Data Members  *(Continued)*

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tokenAction</td>
<td>RefundTokenAction</td>
<td>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. See the RefundTokenAction Subobject.</td>
</tr>
<tr>
<td>transaction</td>
<td>Transaction</td>
<td>The original Transaction to which this refund applies, which must have been successfully captured through CashBox. To process this Refund through CashBox, populate this field with the VID or your transaction ID (merchantTransactionId) to identify the transaction. 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. See Section 18.1: Transaction Data Members.</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
15.2 Refund Subobject

The Refund object has two subobjects:

- RefundItem Subobject
- RefundTokenAction Subobject.

RefundItem Subobject

Describes a line item in a Refund that corresponds to a TransactionItem.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>Amount of the requested refund for this item, in the currency of the overall transaction. This amount must not exceed the original TransactionItem amount, minus any discounts and minus the sum of all prior refunds against this item. This field may be left blank if the taxOnly field is set to TRUE.</td>
</tr>
<tr>
<td>brTaxCategory</td>
<td>The (read-only) Brazilian tax category for this item.</td>
</tr>
<tr>
<td>description</td>
<td>Output only. A description of the TransactionItem, which will be filled in by CashBox from line item description on the original tns:Transaction.</td>
</tr>
<tr>
<td>sku</td>
<td>SKU on the original transaction line item (sku need not be passed in if the transactionItemIndexNumber is passed in). If the sku is passed in, and can be used to uniquely identify the transaction item, it will be used, otherwise an error will be returned.</td>
</tr>
<tr>
<td>taxes</td>
<td>Read-only array to list the taxes from the original TransactionItem that are being refunded.</td>
</tr>
<tr>
<td>taxOnly</td>
<td>If this is true, the amount parameter is ignored and the refund amount will be the total of the as-yet unrefunded taxes for this item.</td>
</tr>
<tr>
<td>transactionItemIndexNumber</td>
<td>Sequence number assigned upon creation to a TransactionItem by CashBox. Either the transactionItemIndexNumber, or the sku, must be specified for each RefundItem.</td>
</tr>
</tbody>
</table>
RefundTokenAction Subobject

Describes the action taken on a Transaction refund, which caused a customer to be granted or to receive tokens.

Table 15-3  RefundTokenAction Object Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancelNegative-Balance</td>
<td>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.</td>
</tr>
<tr>
<td>CancelZeroBalance</td>
<td>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.</td>
</tr>
<tr>
<td>None</td>
<td>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.</td>
</tr>
</tbody>
</table>
## 15.3 Refund Methods

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

### Table 15-4 Refund Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetchByAccount</td>
<td>Returns one or more <code>Refund</code> objects that represent the refunds for the Transactions whose <code>Account</code> object matches the input.</td>
</tr>
<tr>
<td>fetchByTransaction</td>
<td>Returns one or more <code>Refund</code> objects that are associated with the <code>Transaction</code> object specified in the input.</td>
</tr>
<tr>
<td>fetchByVid</td>
<td>Returns a <code>Refund</code> object whose VID matches the input.</td>
</tr>
<tr>
<td>fetchDeltaSince</td>
<td>Returns one or more <code>Refund</code> objects whose time stamp falls on or after the time stamp specified in the input.</td>
</tr>
<tr>
<td>perform</td>
<td>Issues one or more refunds.</td>
</tr>
<tr>
<td>report</td>
<td>Reports the refunds to Vindicia for chargeback processing.</td>
</tr>
</tbody>
</table>
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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load account to search by: No matches.</td>
</tr>
<tr>
<td></td>
<td>• No account specified to load refunds by!</td>
</tr>
<tr>
<td>404</td>
<td>Unable to load account to search by: <code>error-description</code>.</td>
</tr>
</tbody>
</table>
**Example**

```php
$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**

- **srdf**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

  - **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>No transaction specified to load by!</td>
</tr>
<tr>
<td>404</td>
<td>Unable to load refund: No match for transaction.</td>
</tr>
</tbody>
</table>
Example

```php
$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

srd: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>No VID specified to load refund by.</td>
</tr>
<tr>
<td>404</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load refund: No match for VID input-vid.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load refund by VID input-vid: error-description.</td>
</tr>
</tbody>
</table>

Example

$vid='cddbda93f509e2bf8e6d0e7918b0cee2e03cc175'
$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 time stamp falls on or after the time stamp specified in the input. Limit the number of objects returned by specifying an upper limit on the time stamp as well, using the `endTimestamp` parameter.

**Input**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`timestamp`: the search criterion for selecting `Refund` objects to be returned. The time stamps of those selected objects are less than or equal to this value.

`endTimestamp`: the end-date and time stamp. `Refunds` with time stamps greater than this value will not be returned.

**Note:** To reduce high loads on the system, the maximum span for a `fetchDeltaSince()` call is 61 days. If you want to fetch an extended amount of data, best practice suggests that you do so via a loop that pulls a day’s worth of data at a time, with another loop internally that handles fetching each resultant page for that day.

`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 time stamp 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>404</td>
<td>Unable to find payment method in database.</td>
</tr>
<tr>
<td>400</td>
<td>Must specify a time stamp to find refunds newer than ...</td>
</tr>
</tbody>
</table>
Example

```php
refund = new Refund();
$pm = null;
$ = '2009-11-28T12:40:51-0800';
$endTimeStamp = '2009-12-28T12:40:50-0800';
$response = $refund->fetchDeltaSince($,$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**

srđ: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srđ. A null srđ returns the complete response.

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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>206</td>
<td>Some (or all) refunds failed; check VIDs, notes.</td>
</tr>
<tr>
<td>404</td>
<td>Cannot refund transaction: error-description.</td>
</tr>
</tbody>
</table>

Example

Note that use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.

```
// Create a SpecifiedItems 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->setRefundDistributionStrategy('SpecifiedItems');
$refundItem1 = new RefundItem();
$refundItem1->setTransactionItemIndexNumber(1);
$refundItem1->setAmount(5.99);
$refund1->setRefundItems(array($refundItem1));
$refund1->setNote('Refunding due to customer complaint about outage');

// Create another refund object (No Distribution Strategy)
$refund2 = new Refund();
$refund2->setMerchantRefundId('REF102');
$transaction2 = new Transaction();
// merchant ID of a successful transaction that we wish to refund
$transaction2->setMerchantTransactionId('TX102');
$refund2->setTransaction($transaction2);
$refund2->setRefundDistributionStrategy('None');
$refund2->setAmount(10.99);
$refund2->setNote('Customer charged twice');

// Create a tax-only refund object
$refund3 = new Refund();
$refund3->setMerchantRefundId('REF103');
$transaction3 = new Transaction();
// merchant ID of a successful transaction that we wish to refund
$transaction3->setMerchantTransactionId('TX103');
$refund3->setTransaction($transaction3);
$refund3->setRefundDistributionStrategy('SpecifiedItems');
```
$refundItem3 = new RefundItem();
$refundItem3->setTransactionItemIndexNumber(1);
$refundItem3->setTaxOnly(true);
$refund3->setRefundItems(array($refundItem3));
$refund3->setNote('Customer asserts they are tax-exempt');

// Create a RemainingBalance refund object
$refund4 = new Refund();
$refund4->setMerchantRefundId('REF104');
$transaction4 = new Transaction();
// merchant ID of a successful transaction that we wish to refund
$transaction4->setMerchantTransactionId('TX104');
$refund4->setTransaction($transaction4);
$refund4->setRefundDistributionStrategy('RemainingBalance');
$refund4->setNote('Customer fled the country');

$soap_refund = new Refund();
$response = $soap_refund->perform(array($refund1, $refund2, $refund3, $refund4));
if($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**

`srdf`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Unable to save refunds: <code>error-description</code>.</td>
</tr>
</tbody>
</table>
Example

Note that use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.

// 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');</p>

$refund1->setTransaction($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:

---

**Note:** Use of the forward slash character (/) in identifiers is not allowed. See Merchant Identifiers for more information.
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

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>merchantSeason-SetId</td>
<td>string</td>
<td>Your unique ID for this SeasonSet. Free-form string 255 characters or fewer.</td>
</tr>
<tr>
<td>nameValues</td>
<td>NameValuePair</td>
<td>An array of name-value pairs to associate with the SeasonSet.</td>
</tr>
<tr>
<td>seasons</td>
<td>Season</td>
<td>An array of Seasons that make up the SeasonSet.</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>Vindicia's Globally Unique Identifier (GUID) for this object.</td>
</tr>
</tbody>
</table>

Note: use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.

The Season object contains three values:
- description: your description for the Season.
- startDate: its start date.
- endDate: its end date.

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>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fetchAll</code></td>
<td>Returns all <code>SeasonSets</code>.</td>
</tr>
<tr>
<td><code>fetchAllInSeason</code></td>
<td>Returns all in season <code>SeasonSets</code>.</td>
</tr>
<tr>
<td><code>fetchAllOffSeason</code></td>
<td>Returns all off-season <code>SeasonSets</code>.</td>
</tr>
<tr>
<td><code>fetchByMerchantSeasonSetId</code></td>
<td>Returns the <code>SeasonSet</code> specified by the input Merchant ID. Note that use of the forward slash character (/) in merchant identifiers is not allowed. See <a href="#">Merchant Identifiers</a> for more information.</td>
</tr>
<tr>
<td><code>fetchByVid</code></td>
<td>Returns the <code>SeasonSet</code> specified by the input VID.</td>
</tr>
<tr>
<td><code>fetchCurrentSeason</code></td>
<td>Returns the current <code>Season</code> for the input <code>SeasonSet</code>.</td>
</tr>
<tr>
<td><code>fetchNextSeason</code></td>
<td>Returns the next <code>Season</code> for the input <code>SeasonSet</code>.</td>
</tr>
<tr>
<td><code>isInSeason</code></td>
<td>Returns a Boolean flag, which indicates whether the input <code>SeasonSet</code> is in season.</td>
</tr>
<tr>
<td><code>update</code></td>
<td>Creates a new <code>SeasonSet</code> or updates an existing one.</td>
</tr>
</tbody>
</table>
fetchAll

The fetchAll method returns all existing SeasonSets.

Input

srdf: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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

```php
$ss_factory = new SeasonSet();
$page = 0;
$pageSize = 10;
while ($count > 0) {
    $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++;
}
```
fetchAllInSeason

The fetchAllInSeason method returns all existing SeasonSet objects that are in season during the input nowDate.

Input

sr드: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the sr드. A null sr드 returns the complete response.

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

```php
$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**

- `srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

  - `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**

```php
$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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

  - **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**

```php
$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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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**

```php
$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**

- **srdf**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

  - `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**

```php
$ss = new SeasonSet();
$response = $ss->fetchCurrentSeason();

// check $response

$season = $response['season'];
```
fetchNextSeason

The `fetchNextSeason` method returns the next `Season` for the input `SeasonSet`.

**Input**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- **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**

```php
$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

srd: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

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

- `srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- `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

```php
$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.: 

Note: Use of the forward slash character (/) in identifiers is not allowed. See Merchant Identifiers for more information.
### 17.1 Token Data Members

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

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>string</td>
<td>Optional. A description of this token type in your application.</td>
</tr>
<tr>
<td>merchantTokenId</td>
<td>string</td>
<td>Required. Your unique identifier for this Token object. This ID is also referred to as the <code>token type</code>. For example, an airline might identify a <code>Token</code> object with the ID <code>FREQ_FLIER_MILES_2010</code> to denote the number of frequent-flier miles accumulated by a customer account in 2010. A cell-phone company might use <code>ANYTIME_PHONE_MINUTES</code> to identify <code>Token</code> objects that specify a customer’s balance of anytime minutes. <strong>Note:</strong> use of the forward slash character (/) in merchant identifiers is not allowed. See <a href="#">Merchant Identifiers</a> for more information.</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>Vindicia’s Globally Unique Identifier (GUID) for this object. When creating a new <code>Token</code> object, leave this field blank; it will be automatically populated by CashBox.</td>
</tr>
</tbody>
</table>
17.2 Token Methods

The following table lists and summarizes the methods for the Token object.

Table 17-2 Token Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetch</td>
<td>Returns an existing Token object.</td>
</tr>
<tr>
<td>update</td>
<td>Creates or updates a Token object.</td>
</tr>
</tbody>
</table>
**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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>No token specified to load!</td>
</tr>
</tbody>
</table>

**Example**

```php
$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: ";
    print $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**

**srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the **srd**. A null **srd** returns the complete response.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Unable to save token: <code>error-description</code>.</td>
</tr>
<tr>
<td>501</td>
<td><code>Error-description</code>. (Returned if the call cannot map the SOAP <strong>Token</strong> object to CashBox's database representation of the token.)</td>
</tr>
</tbody>
</table>

**Example**

```php
$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";
} else {
    print $response['errorDetails']->errorDescription . "\n";
}
```

© 2019 Vindicia, Inc.  Table of Contents  The Token Object  400
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 one-time billing or recurring billing, or for a transaction occurring outside of CashBox but reported to Vindicia for chargeback dispute through ChargeGuard. Note however that it is the Refund object, not the Transaction object, that encapsulates 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.:

Note: Use of the forward slash character (/) in identifiers is not allowed. See Merchant Identifiers for more information.

A Transaction object might also represent a potential, rather than a completed, financial transaction. For example, you can 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 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 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>
<thead>
<tr>
<th>Transaction Type</th>
<th>Description</th>
<th>Initiated By</th>
<th>Transaction API Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migrated</td>
<td>Transactions generated by a billing system other than CashBox, which 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.</td>
<td>You, the merchant</td>
<td>AutoBill.migrate or Transaction.migrate</td>
</tr>
<tr>
<td>Real-time (one-time)</td>
<td>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.</td>
<td>You, the merchant</td>
<td>Transaction.auth() or Transaction.capture()</td>
</tr>
<tr>
<td>Recurring billing</td>
<td>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.</td>
<td>CashBox</td>
<td>None. An active AutoBill object must exist.</td>
</tr>
<tr>
<td>Reported</td>
<td>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.</td>
<td>You, the merchant</td>
<td>Transaction.report() or Transaction.score()</td>
</tr>
</tbody>
</table>
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.

### Table 18-1 Uses for the Transaction Object (Continued)

<table>
<thead>
<tr>
<th>Transaction Type</th>
<th>Description</th>
<th>Initiated By</th>
<th>Transaction API Call</th>
</tr>
</thead>
</table>
| **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.  
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.  
Once you have migrated the transaction, CashBox will enter the missing details.                                                                                                           | Indirectly by you, the merchant   | None.                |
| **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.  
If the Transaction is authorized, the payment method is considered valid, and the transaction status becomes AuthorizedForValidation; if not, the status is Cancelled.  
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.                |
18.1 Transaction Data Members

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

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>Account</td>
<td>The Account object that represents the customer to which this Transaction object applies. See Section 1.2: Account Data Members.</td>
</tr>
<tr>
<td>amount</td>
<td>decimal</td>
<td>Required. The monetary amount for this Transaction object, that is, the total cost of one or more line items purchased. When you process</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the transaction through CashBox by calling auth or authCapture, CashBox fills in this attribute based on the total of the line items (see the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TransactionItem attributes) added to the transaction. When reporting the transaction to Vindicia, ensure that the transaction amount</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matches the total of the line items. For CashBox-generated Transactions, this field is generated automatically.</td>
</tr>
<tr>
<td>autoBillCycle</td>
<td>int</td>
<td>The AutoBill Billing Cycle during which this Transaction occurred.</td>
</tr>
<tr>
<td>billingPlanCycle</td>
<td>int</td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the most recent recurring billing event, as determined by the AutoBill and its BillingPlan. Note: This data member will increment for free</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cycles.</td>
</tr>
<tr>
<td>billingStatementIdentifier</td>
<td>string</td>
<td>The string that is displayed on a customer's billing statement. For one-time transactions, CashBox supports this value for only certain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>payment processors. Because this value and its format are constrained by your payment processor, consult with Vindicia Client Services</td>
</tr>
<tr>
<td></td>
<td></td>
<td>before setting its value. If GlobalCollect, MeS, Chase Paymentech or Litle is your payment processor, see Appendix A: Custom Billing Statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Identifier Requirements in the CashBox Programming Guide.</td>
</tr>
<tr>
<td>currency</td>
<td>string</td>
<td>The ISO 4217 currency code (see <a href="http://www.xe.com/iso4217.htm">www.xe.com/iso4217.htm</a>) for this Transaction object. The default is USD. To determine the actual monetary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value, set the values for both amount and currency.</td>
</tr>
<tr>
<td>destPaymentMethod</td>
<td>PaymentMethod</td>
<td>The payment method for deposits to a customer account for this Transaction object. This field is used to make outbound ECP payments or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>transfers. See Section 11.1: PaymentMethod Data Members.</td>
</tr>
<tr>
<td>divisionNumber</td>
<td>string</td>
<td>The number of your division or group with your payment processor for this Transaction. Chase Paymentech refers to this number as the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Division Number; Litle calls it the Report Group; MeS calls it the Profile ID. Do not specify this attribute for one-time transactions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you subscribe to ChargeGuard, complete this field when reporting transactions to CashBox. CashBox will use this value to match the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transaction to the appropriate chargeback from the payment processor.</td>
</tr>
</tbody>
</table>
Table 18-2 Transaction Object Data Members *(Continued)*

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ecpTransaction-</td>
<td>ECPTransaction-Type</td>
<td>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.</td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mandate</td>
<td>Mandate</td>
<td>Mandate used to authorize this transaction. See the Mandate Subobject for details.</td>
</tr>
<tr>
<td>merchantAffiliateId</td>
<td>string</td>
<td>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. <strong>Note:</strong> use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
<tr>
<td>merchantAffiliateSubId</td>
<td>string</td>
<td>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. <strong>Note:</strong> use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
<tr>
<td>merchantTransactionId</td>
<td>string</td>
<td>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. 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. 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. <strong>Note:</strong> use of the forward slash character (/) in merchant identifiers is not allowed. See Merchant Identifiers for more information.</td>
</tr>
</tbody>
</table>
CashBox 25.0.0: API Reference Guide

Transaction Data Members

Table 18-2 Transaction Object Data Members  *(Continued)*

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nameValues</td>
<td>NameValuePair[]</td>
<td>Optional. An array of name–value pairs, which are useful in tracking the associated AutoBill object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CashBox provides the following name-value pairs for use with EDD/SEPA European Direct Debit payment methods:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Use name vin:MandateFlag and value 1 to associate the EDD/SEPA European Direct Debit Payment Method with the AutoBill.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Use name vin:MandateVersion and value 1.0.1, to associate a mandate document of version 1.0.1 with the object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Use name vin:MandateID to pass the Mandate ID field of the EDD/SEPA European Direct Debit Extension record to Chase Paymntech.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Use name vin:MandateApprovalDate to pass the Signature Date field of the EDD/SEPA European Direct Debit Extension Record to Chase Paymntech.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Use vin:ValidateFullAmt to pass whether full amount authorization needs to be done before charging. This must be enabled if the processor is Global Collect.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vin:PayPalNoShippingAddress—if set to 1 passes the NOSHIP-PING=1 flag in the PayPal ExpressCheckOut request, masking the shipping address for customers buying digital goods.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>paypal:NoCommitUserAction—if set to 1, removes the useraction=commit setting from the PayPal ExpressCheckOut request, resulting in a the PayPal checkout page showing Agree to Continue instead of Agree and Pay. Use this setting if you want to do additional checking after the customer exits this checkout page and final authorization doesn't immediately follow. CashBox supports this feature only for one-time transactions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>paypal:SuppressItems—if set to 1, suppresses display of line item details of a purchase during a PayPal ExpressCheckOut request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To prevent showing item detail and pricing, provide the name–value pairs: paypal:SuppressItems and paypal:NoCommitUserAction, both with a value of 1, in AutoBill or Transaction objects.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you use Vindicia's legacy Tax engine or any variety of Avalara Tax service, you can use the vin:NegativeAmountTaxClassification data option on any Transaction (name = vin:NegativeAmountTaxClassification, value = xyz, where xyz is a tax class). This instructs CashBox to calculate a negative tax amount on the part of the total that was paid for using currency credits. This is useful if the customer paid taxes when purchasing the currency credits.</td>
</tr>
</tbody>
</table>
The following name-value pairs are automatically populated by CashBox for AutoBill-generated Transactions:

- **vin:AutoBillVID** — the VID of the AutoBill for which this Transaction was generated.
- **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.)
- **vin:entifier** — your unique ID for the AutoBill for which this Transaction was generated.
- **vin:RetryNumber** — the attempt number (in retry cycles) of the Transaction.
- **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.

If Global Collect is your payment processor, you can set `vin:recurring` to **1** to have CashBox flag one-time-Transactions as recurring Transactions. CashBox then passes the values: `orderId`, `EffortID`, and `AttemptID` to Global Collect, as set by you in the `vin:orderId`, `vin:EffortID`, and `vin:AttemptID` fields.

You can send these name-value pairs on `transaction.auth()`, `transaction.capture()` and `transaction.authcapture()` calls for one-time transactions to indicate that those transactions should be flagged as recurring.

**Table 18-2 Transaction Object Data Members (Continued)**

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| **nameValues**       | NameValuePair[] | Note: All name-value pairs included with the Transaction-generating AutoBill will be automatically copied to the resultant Transaction. The following name-value pairs are automatically populated by CashBox for AutoBill-generated Transactions:  
- **vin:AutoBillVID** — the VID of the AutoBill for which this Transaction was generated.  
- **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.)  
- **vin:entifier** — your unique ID for the AutoBill for which this Transaction was generated.  
- **vin:RetryNumber** — the attempt number (in retry cycles) of the Transaction.  
- **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.  

See Section 10: The NameValuePair Object.  
If Global Collect is your payment processor, you can set `vin:recurring` to **1** to have CashBox flag one-time-Transactions as recurring Transactions. CashBox then passes the values: `orderId`, `EffortID`, and `AttemptID` to Global Collect, as set by you in the `vin:orderId`, `vin:EffortID`, and `vin:AttemptID` fields.  
You can send these name-value pairs on `transaction.auth()`, `transaction.capture()` and `transaction.authcapture()` calls for one-time transactions to indicate that those transactions should be flagged as recurring. |
| **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.  
**Note:** If CashBox handles the billing, do not fill in this field. |
| **paymentProcessorTransactionId** | string        | The name Vindicia uses to identify a CashBox Transaction to the associated payment processor. This is usually the CashBox transactionId. In some cases however, due to the format requirements of some payment processors, CashBox alters the Transaction name before sending it to the processor. In those cases, the name of a transaction in a report by the payment processor will differ from the name of the same transaction in a report by Vindicia. The paymentProcessorTransactionId data member enables you to reconcile the original CashBox Transaction name with the altered Transaction name CashBox sends to the payment processor, to identify the Transaction. |
### Table 18-2 Transaction Object Data Members (Continued)

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preferredNotificationLanguage</td>
<td>string</td>
<td>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 CashBox database as part of your configuration. This value overrides any language setting in the Account object for this transaction.</td>
</tr>
<tr>
<td>previousMerchantTransactionId</td>
<td>string</td>
<td>Your unique identifier for a previous transaction referenced by this Transaction object.</td>
</tr>
<tr>
<td>salesTaxAddress</td>
<td>Address</td>
<td>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. Note: If CashBox calculates sales tax for you, leave this field empty. See Section 3.1: Address Data Members.</td>
</tr>
<tr>
<td>shippingAddress</td>
<td>Address</td>
<td>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. See Section 3.1: Address Data Members.</td>
</tr>
<tr>
<td>sourceIp</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>sourceMacAddress</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>sourcePaymentMethod</td>
<td>PaymentMethod</td>
<td>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. 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. For recurring transactions generated by CashBox, this attribute is the PaymentMethod object associated with the corresponding AutoBill object. When reporting a transaction for ChargeGuard, partially mask the payment method’s data members, for added security. See Section 11.1: PaymentMethod Data Members.</td>
</tr>
<tr>
<td>sourcePhoneNumber</td>
<td>string</td>
<td>Optional. The phone number from which this Transaction object originated. This information may be useful in chargeback disputes.</td>
</tr>
</tbody>
</table>

© 2019 Vindicia, Inc.  
Table of Contents  
The Transaction Object  
408
The Transaction Object

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statusLog</td>
<td>TransactionStatus()</td>
<td>An array of the statuses this Transaction object has gone through, with the first entry being the most recent status. Each Transaction-Status 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. 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. For recurring transactions, CashBox sets this attribute when capturing the transaction with the payment processor. 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. See the TransactionStatus Subobject.</td>
</tr>
<tr>
<td>taxExemptions</td>
<td>TaxExemption</td>
<td>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. See the TaxExemption Subobject.</td>
</tr>
<tr>
<td>timestamp</td>
<td>dateTime</td>
<td>A time stamp 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.</td>
</tr>
<tr>
<td>transaction-Items</td>
<td>TransactionItem</td>
<td>A TransactionItem array that lists the line items that comprise this Transaction object. Each item is a separate data structure of type TransactionItem. For migrated transactions, CashBox does not validate that the subitem amounts listed here add up to the total transaction amount (see the amount attribute). 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. 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. To add sales tax when migrating transactions, include the tax as a line item here. See the TransactionItem Subobject.</td>
</tr>
<tr>
<td>userAgent</td>
<td>string</td>
<td>Optional. Your customer’s user agent from whom this Transaction originated.</td>
</tr>
<tr>
<td>Data Member</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>verification-Code</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>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. <strong>Note:</strong> 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.</td>
</tr>
<tr>
<td>originalActivityDate</td>
<td>string:dateTime</td>
<td>The date of the original transaction attempt for the governing AutoBill that spawned the transaction—null for one-time transactions or any transaction not part of an AutoBill.</td>
</tr>
<tr>
<td>retryNumber</td>
<td>string</td>
<td>The number of retry attempts to bill for this transaction. This index begins with 0 for the initial attempt to charge for a particular billing. A value of 1 indicates the first retry, 3 indicates the third retry, and so on. For one-time transactions, this value is null.</td>
</tr>
<tr>
<td>autobillVID</td>
<td>string</td>
<td>The Vindicia Record ID (or VID) of the AutoBill that &quot;owns&quot; the transaction—null for one-time transactions or any transaction not associated with an AutoBill.</td>
</tr>
<tr>
<td>identifier</td>
<td>string</td>
<td>The merchant-specified identifier of the AutoBill that &quot;owns&quot; the transaction—null for one-time transactions.</td>
</tr>
</tbody>
</table>
18.2 Transaction Subobjects

The Transaction object has several subobjects:

- AVSMatchType Subobject
- Mandate Subobject
- MigrationTaxItem Subobject
- MigrationTransaction Subobject
- MigrationTransactionItem Subobject
- MigrationTransactionType Subobject
- TransactionItem Subobject
- TransactionStatus Subobject
- TransactionStatusBoleto Subobject
- TransactionStatusCreditCard Subobject
- TransactionStatusECP Subobject
- TransactionStatusHostedPage Subobject
- TransactionStatusPayPal Subobject
- TransactionStatusSkrill Subobject
- TransactionStatusType Subobject
- TransactionValidationResponse Subobject

**AVSMatchType Subobject**

Defines the AVS Match type.

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FullMatch</td>
<td>string</td>
<td>The billing address from the customer matches the one on file with the bank.</td>
</tr>
<tr>
<td>IssuerError</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>NoMatch</td>
<td>string</td>
<td>The billing address from the customer does not match the one on file with the bank.</td>
</tr>
<tr>
<td>NoOpinion</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
### AVSMatchType Object Data Members (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotSupported</td>
<td>string</td>
<td>The AVS match type requested is not supported.</td>
</tr>
<tr>
<td>PartialMatch</td>
<td>string</td>
<td>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 <code>creditCardStatus</code> attribute.</td>
</tr>
</tbody>
</table>

**Mandate Subobject**

See the Mandate Subobject for details.

**MigrationTaxItem Subobject**

Defines a tax line-item in a MigrationTransactionItem.

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>decimal</td>
<td>Tax amount in the currency of the overall transaction.</td>
</tr>
<tr>
<td>jurisdiction</td>
<td>string</td>
<td>Sales tax jurisdiction for the Transaction.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>Sales tax name.</td>
</tr>
</tbody>
</table>
**MigrationTransaction Subobject**

Defines a Transaction migrated to CashBox from a different billing system.

Table 18-5 MigrationTransaction Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>Account</td>
<td>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. 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.</td>
</tr>
<tr>
<td>amount</td>
<td>decimal</td>
<td>Required. The amount of the transaction, as a decimal. Must be non-negative, and add up to the total value of all the associated TransactionItems.</td>
</tr>
<tr>
<td>autoBillCycle</td>
<td>int</td>
<td>Required. The billing sequence number for the Transaction within the life of the AutoBill. (Note: The first CashBox billing = 0.)</td>
</tr>
<tr>
<td>billingDate</td>
<td>dateTime</td>
<td>Required. The AutoBill's Billing Plan Period start date/time associated with the Transaction.</td>
</tr>
<tr>
<td>billingPlanCy-</td>
<td>int</td>
<td>Required. The billing sequence for the Transaction within the specified Billing Plan.</td>
</tr>
<tr>
<td>cle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>currency</td>
<td>string</td>
<td>Required. The ISO 4217 currency code used for this Transaction. Defaults to USD if not specified.</td>
</tr>
<tr>
<td>divisionNumber</td>
<td>string</td>
<td>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; Little calls it the Report Group; MeS calls it the Profile ID. If you provide a value for divisionNumber when migrating a Transaction into CashBox, this value will be set on that imported Transaction, and used in all subsequent activities, such as refunds. If you do not provide a divisionNumber, CashBox uses the global default. (See also vin:OriginalDivisionNumber in the CashBox Programming Guide, chapter 17 Working with Name-Value Pairs.)</td>
</tr>
<tr>
<td>merchantAffili-</td>
<td>string</td>
<td>Optional. Your ID (a free-form string of 128 characters or less) for the affiliate that submitted this Transaction object, if any.</td>
</tr>
<tr>
<td>ateId</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 18-5 MigrationTransaction Object Data Members  (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>merchantAffiliateSubId</td>
<td>string</td>
<td>Optional. Your ID (a free-form string of 128 characters or less) for the sub-affiliate that submitted this Transaction object, if any.</td>
</tr>
<tr>
<td>merchantBillingPlanId</td>
<td>string</td>
<td>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. For more information, see Section 5: The BillingPlan Object.</td>
</tr>
<tr>
<td>merchantTransactionId</td>
<td>string</td>
<td>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. Note: For PayPal transactions, this value must be the INVNUM or INVOICEID field sent to PayPal for the transaction.</td>
</tr>
<tr>
<td>migrationTransactionItems</td>
<td>MigrationTransactionItem</td>
<td>Required. An array of MigrationTransactionItems included with the Transaction. For more information, see the MigrationTransactionItem Subobject.</td>
</tr>
<tr>
<td>nameValues</td>
<td>NameValuePair</td>
<td>An optional array of name-value pairs you wish to associate with the Transaction. Transactions generated as a result of the AutoBill.modify call will include a name-value pair with name vin:type and value modify. See Section 10: The NameValuePair Object.</td>
</tr>
<tr>
<td>paymentMethod</td>
<td>PaymentMethod</td>
<td>Required. The Payment Method (e.g., a credit card) used for this Transaction.</td>
</tr>
<tr>
<td>paymentProcessor</td>
<td>string</td>
<td>The payment processor for this Transaction. Possible values include FDMS, GlobalCollect, Little, MeS, Orbital, PayFlowPro, PayPal, Paymentech, and Other. If the Payment Processor is not supported by CashBox, the migrationTransaction will be imported, but other actions (such as 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.</td>
</tr>
<tr>
<td>paymentProcessorTransId</td>
<td>string</td>
<td>The identifier assigned to this Transaction by your Payment Processor.</td>
</tr>
</tbody>
</table>
Table 18-5  MigrationTransaction Object Data Members  (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preferredNotificationLanguage</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>retryNumber</td>
<td>integer</td>
<td>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. If the migrationTransaction is included in an AutoBill.migrate request, but retryNumber is not specified, this field will default to 0.</td>
</tr>
<tr>
<td>salesTaxAddress</td>
<td>Address</td>
<td>The address used to calculate sales tax on this Transaction. This field should be included if you include taxes in the Transaction.</td>
</tr>
<tr>
<td>shippingAddress</td>
<td>Address</td>
<td>Optional. The shipping address for this Transaction object. Note: While optional, this field is useful in resolving chargebacks. See Section 3.1: Address Data Members.</td>
</tr>
<tr>
<td>sourceIp</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>statusLog</td>
<td>TransactionStatus</td>
<td>Required. A log of TransactionStatus entries (with accurate time stamps) associated with this Transaction. At least one TransactionStatus object with a time stamp and status set to Cancelled, Captured or Settled must be included with every AutoBill.migrate call. For CreditCard and ECP PaymentMethod objects, enter the avsCode and cvnCode to help the CashBox Chargeback team fight Chargebacks. See the TransactionStatus Subobject.</td>
</tr>
<tr>
<td>taxExemptions</td>
<td>TaxExemption</td>
<td>An array of Exemptions that apply to this Transaction. Multiple tax exemptions may be defined. See the TaxExemption Subobject.</td>
</tr>
</tbody>
</table>
Table 18-5 MigrationTransaction Object Data Members (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taxInclusive</td>
<td>Boolean</td>
<td>A Boolean flag which defines whether the price listed for the Transaction is inclusive or exclusive of tax. If true, CashBox treats the MigrationTransactionItem 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.</td>
</tr>
<tr>
<td>type</td>
<td>MigrationTransac-</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>tionType</td>
<td></td>
</tr>
<tr>
<td>verification-Code</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemType</td>
<td>MigrationTransac-</td>
<td>The Migrated Transaction Item's type, which may be one of three values:</td>
</tr>
<tr>
<td></td>
<td>tionItemType</td>
<td>• Credit: a one-time charge (not necessarily associated with an AutoBillItem).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• NonRecurringCharge: an after tax Credit applied to a Transaction.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RecurringCharge: a one-time charge (not necessarily associated with an AutoBillItem).</td>
</tr>
<tr>
<td></td>
<td>Optional. Your</td>
<td>If unspecified, the type defaults to RecurringCharge.</td>
</tr>
<tr>
<td>merchantAuto-</td>
<td>unique identifier</td>
<td></td>
</tr>
<tr>
<td>BillItemId</td>
<td>for the AutoBill-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Item associated</td>
<td></td>
</tr>
<tr>
<td></td>
<td>with this</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MigrationTransac-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tionItem. Use this</td>
<td></td>
</tr>
<tr>
<td></td>
<td>data member to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>distinguish</td>
<td></td>
</tr>
<tr>
<td></td>
<td>between two or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>more AutoBillItems</td>
<td></td>
</tr>
<tr>
<td></td>
<td>for the same</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Product.</td>
<td></td>
</tr>
</tbody>
</table>
MigrationTransactionItem Subobject

Defines the migrated Transaction’s type.

Table 18-7 MigrationTransactionType Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonRecurring</td>
<td>string</td>
<td>A one-time charge (not necessarily associated with an AutoBill).</td>
</tr>
<tr>
<td>recurring</td>
<td>string</td>
<td>A recurring charge (associated with an AutoBill).</td>
</tr>
</tbody>
</table>
Transaction Item 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>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoBillItem-Vid</td>
<td>string</td>
<td>Vindicia’s unique identifier for the associated Auto-BillItem.</td>
</tr>
<tr>
<td>brTaxCategory</td>
<td>string</td>
<td>The Brazilian tax category for this item.</td>
</tr>
<tr>
<td>campaignCode</td>
<td>string</td>
<td>Campaign code redeemed on this Transaction. To apply a Campaign, use this field to pass in a valid Coupon or Promotion code. Note: This data member will not be returned.</td>
</tr>
<tr>
<td>campaignDescription</td>
<td>string</td>
<td>Description for the Campaign used to discount this transaction, if any.</td>
</tr>
<tr>
<td>campaignId</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>discount</td>
<td>decimal</td>
<td>Read only. A field describing the discount amount on this transaction.</td>
</tr>
<tr>
<td>indexNumber</td>
<td>int</td>
<td>Unique index number for this item. Index numbers start at 1, not 0.</td>
</tr>
<tr>
<td>itemRefunds</td>
<td>TransactionItemRefundSummary</td>
<td>Contains a summary of Refunds against a TransactionItem.</td>
</tr>
</tbody>
</table>
  * amount—Full or partial amount of the refund. |
  * taxAmount—Full or partial amount of tax refunded. |
  * taxOnly—If set to true, CashBox refunded only the tax portion of a TransactionItem. |
  * timestamp—Timestamp when the refund was issued. |
  * refundID—the merchantRefundId of the Refund object where the refund was issued. |
**Table 18-8 TransactionItem Object Data Members (Continued)**

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemType</td>
<td>Transaction-ItemType</td>
<td>Contains one of the following values. If unspecified, the type will default to Purchase.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Purchase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Credit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TaxableCredit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DiscountBeforeTax</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ModificationRefund</td>
</tr>
<tr>
<td></td>
<td></td>
<td>As of this release, passing in negative-amount Transaction items to affect an ad hoc custom discount is no longer supported. To achieve the same effect, pass in the absolute value in the amount field for the item and specify DiscountBeforeTax here.</td>
</tr>
<tr>
<td>merchantAutoBillItemId</td>
<td>string</td>
<td>Your identifier for the associated AutoBillItem.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>A description of the item. For CashBox-generated re-bill transactions in which this Transaction item is derived from a Product or BillingPlan object used with an AutoBill object, this value maps to the Product or BillingPlan object's description attribute. For TransactionItems which reflect Campaign discounts, this data member will be populated by CashBox with the text &quot;Discount for description,&quot; where description is the description data member for the ProductDescription subobject of the Product receiving the discount or the BillingPlan-Description data member of the Billing Plan receiving the discount.</td>
</tr>
<tr>
<td>nameValues</td>
<td>NameValuePair</td>
<td>Optional. An array of name-value pairs the merchant wishes to associate with a Transaction item. Name-Values associated with an AutoBill Item will automatically be populated on corresponding Transaction Items.  See Section 10: The NameValuePair Object</td>
</tr>
<tr>
<td>price</td>
<td>decimal</td>
<td>The item price, denominated by the currency data member of this Transaction object. price is omitted on rated items, because they do not have a fixed, per unit price but are dependent on the tier of the rate plan.</td>
</tr>
<tr>
<td>quantity</td>
<td>decimal</td>
<td>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. For rated items, this number can be a fraction.</td>
</tr>
<tr>
<td>Data Members</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>relatedTransactions</td>
<td>string</td>
<td>List of merchantTransactionIds for all Transactions related to this item. If you have a credit resulting from the removal of an AutoBillItem or BillingPlan, relatedTransactions contains the list of all Transactions originally billed for the item within the timeframe covered by the credit.</td>
</tr>
<tr>
<td>servicePeriodEndDate</td>
<td>dateTime</td>
<td>The start date for the service provided by this TransactionItem. 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. 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. <strong>Note:</strong> 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.</td>
</tr>
<tr>
<td>servicePeriodStartDate</td>
<td>dateTime</td>
<td>The start date for the time period reflected by this TransactionItem.</td>
</tr>
<tr>
<td>sku</td>
<td>string</td>
<td>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. For TransactionItems which reflect Campaign discounts, this data member will be populated by CashBox with the text &quot;Discount formermerchantProductId merchantProductId,&quot; where merchantProductId is the merchantProductId data member for the Product object receiving the discount. This data member will also be populated by CashBox with the text &quot;Discount for merchantBillingPlanId merchantBillingPlanId,&quot; where merchantBillingPlanId is the merchantBillingPlanId data member for the BillingPlan object receiving the discount.</td>
</tr>
<tr>
<td>subtotal</td>
<td>decimal</td>
<td>Read only. The subtotal of the values on the TransactionItem, excluding any discounts and taxes, if tax-exclusive. If tax is inclusive, for example a value-added tax (VAT), it will be included.</td>
</tr>
</tbody>
</table>

Table 18-8 TransactionItem Object Data Members (Continued)
### Table 18-8 TransactionItem Object Data Members  (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tax</td>
<td>Tax()</td>
<td>An array of Tax objects, which include the following data members:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• jurisdiction: (string) the TransactionItem sku for the tax.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• name: (string) the description for the tax.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• amount: (decimal) the amount for the tax.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• externalTaxName: The name for the tax or fee as provided by the tax service; corresponds to “Imposition” in Vertex and “TaxName” in Avatax.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• externalTaxCategory: The category of tax or fee as provided by the tax service; corresponds to “ImpositionType” in Vertex and “TaxGroup” in Avatax.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• externalJurisdictionName: The name of the tax jurisdiction for this tax or fee as provided by the tax service; corresponds to “Jurisdiction” in Vertex and “JurisName” in Avatax.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• taxRate: The effective rate used to calculate this tax as provided by the tax service; corresponds to “EffectiveRate” in Vertex and “Rate” in Avatax.</td>
</tr>
<tr>
<td>taxClassification</td>
<td>string</td>
<td>A string that defines the tax classification for this TransactionItem.</td>
</tr>
<tr>
<td>taxType</td>
<td>string</td>
<td>This data member will be automatically populated by CashBox with applied tax information for the TransactionItem.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Possible values include Inclusive Sales, Exclusive Sales, Inclusive Use, and Exclusive Use.</td>
</tr>
<tr>
<td>tokens</td>
<td>TokenAmount()</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Section 17.1: Token Data Members.</td>
</tr>
<tr>
<td>total</td>
<td>decimal</td>
<td>Read only. The total charge for the transaction, including discounts and taxes. CashBox calculates discount, subtotal, and total. If you pass these in, they will be ignored.</td>
</tr>
</tbody>
</table>
### 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 time stamp and a status of `Cancelled`, `Captured`, or `Settled`.

#### Table 18-9 TransactionStatus Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amazonStatus</td>
<td>TransactionStatusAmazon</td>
<td>Status details for an Amazon Transaction.</td>
</tr>
<tr>
<td>boletoStatus</td>
<td>TransactionStatusBoleto</td>
<td>The status of a Boleto Bancário-based transaction. This field is populated with the <code>uri</code> received for this Transaction (the URL your payment processor received in response to a presentment of the fiscal number for the Transaction). <strong>Note:</strong> CashBox does not support the Boleto Payment method for migrated Transactions. See the <code>TransactionStatusBoleto Subobject</code>.</td>
</tr>
<tr>
<td>carrierBillingStatus</td>
<td>TransactionStatusCarrierBilling</td>
<td>The status for a Carrier Billing based Transaction. This object contains two data members: authCode: Result code for the requested action. buyUrl: URL which (when sourced on a customer’s browser) generates HTML elements to facilitate processing of a CarrierBilling payment. <strong>Note:</strong> CashBox does not support the Carrier Billing payment method for migrated Transactions.</td>
</tr>
<tr>
<td>creditCardStatus</td>
<td>TransactionStatusCreditCard</td>
<td>The most recently returned status of the credit-card-based transaction. For migrated Transactions, populate this field with the payment-processor-specific details, such as the authorization code. When reporting transactions to Vindicia for ChargeGuard, specify this attribute to help Vindicia dispute chargebacks. See the <code>TransactionStatusCreditCard Subobject</code>.</td>
</tr>
<tr>
<td>directDebitStatus</td>
<td>TransactionStatusDirectDebit</td>
<td><em>(This data member is not in use.)</em></td>
</tr>
</tbody>
</table>
### Table 18-9 TransactionStatus Object Data Members (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ecpStatus</td>
<td>Transaction-StatusECP</td>
<td>The status of an ECP-based transaction. For migrated Transactions, populate this field with the most recent status received from your Payment Processor. When reporting transactions to Vindicia for ChargeGuard, specify this attribute, to help Vindicia dispute chargebacks. See the TransactionStatusECP Subobject.</td>
</tr>
<tr>
<td>fundingSource-Balance</td>
<td>decimal</td>
<td>The outstanding available balance on the submitted PaymentMethod.</td>
</tr>
<tr>
<td>hostedPageStatus</td>
<td>Transaction-StatusHostedPage</td>
<td>Status details for a HostedPage Transaction. Note: The customer’s Account must exist before any Hosted Page related call references that Account. See the TransactionStatusHostedPage Subobject.</td>
</tr>
<tr>
<td>paymentMethod-Type</td>
<td>PaymentMethod-Type</td>
<td><strong>Optional.</strong> 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. 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. See the PaymentMethodType Subobject.</td>
</tr>
<tr>
<td>payPalStatus</td>
<td>Transaction-StatusPayPal</td>
<td>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. See the TransactionStatusPayPal Subobject.</td>
</tr>
<tr>
<td>skrillStatus</td>
<td>Transaction-StatusSkrill</td>
<td>Status details for a Skrill Transaction.</td>
</tr>
<tr>
<td>status</td>
<td>Transaction-StatusType</td>
<td>An enumerated string that specifies the transaction status. 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. See the TransactionStatusType Subobject.</td>
</tr>
</tbody>
</table>
### TransactionStatusBoleto Subobject

Defines the status for a Credit Card transaction.

#### Table 18-10 TransactionStatusBoleto Object Data Member

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uri</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>

### TransactionStatusCreditCard Subobject

Defines the status for a Credit Card transaction.

#### Table 18-11 TransactionStatusCreditCard Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authCode</td>
<td>string</td>
<td>The reason code returned by the payment processor when this Transaction object is authorized, captured, or cancelled.</td>
</tr>
<tr>
<td>avsCode</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
### TransactionStatusCreditCard Subobject

Defines the status for a Boleto Bancario transaction.

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvnCode</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>authApprovalCode</td>
<td>string</td>
<td>Authorization Code provided by the payment processor approving the credit card transaction. Any CashBox Method that returns a Transaction Object returns the Authorization Code as part of the Transaction Object.</td>
</tr>
</tbody>
</table>

### TransactionStatusECP Subobject

Defines the status for a Boleto Bancario transaction.

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authCode</td>
<td>string</td>
<td>The reason code returned by the payment processor when this Transaction object is authorized, captured, or cancelled.</td>
</tr>
</tbody>
</table>

### TransactionStatusHostedPage Subobject

Defines the status for a Hosted Page transaction.

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authCode</td>
<td>string</td>
<td>The result code for the status update.</td>
</tr>
<tr>
<td>redirectUrl</td>
<td>string</td>
<td>The Hosted Pages URL to which your customer should be redirected to complete a HostedPage Transaction.</td>
</tr>
</tbody>
</table>
## TransactionStatusPayPal Subobject

Defines the status for a PayPal transaction.

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authCode</td>
<td>string</td>
<td>The success or failure return code received from PayPal after authorization is finalized.</td>
</tr>
<tr>
<td>payerId</td>
<td>string</td>
<td>Unique PayPal customer account identification number in PayPal's ExpressCheckout</td>
</tr>
<tr>
<td>redirectUrl</td>
<td>string</td>
<td>The PayPal URL to which you must redirect your customer to complete a PayPal transaction.</td>
</tr>
<tr>
<td>token</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>

## TransactionStatusSkrill Subobject

Defines the status for a Skrill transaction.

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authCode</td>
<td>string</td>
<td>The success or failure return code received from Skrill after authorization is finalized.</td>
</tr>
<tr>
<td>redirectUrl</td>
<td>string</td>
<td>The Skrill URL to which you must redirect your customer to complete a Skrill transaction.</td>
</tr>
<tr>
<td>token</td>
<td>string</td>
<td>The session ID for Skrill Gateway.</td>
</tr>
</tbody>
</table>
**TransactionStatusType Subobject**

Defines the Transaction Status.

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthExpired</td>
<td>string</td>
<td>The transaction was not captured and the auth has expired. The transaction must be re-authorized and then captured.</td>
</tr>
<tr>
<td>Authorization-Pending</td>
<td>string</td>
<td>A PayPal-based transaction in CashBox that is awaiting further action by the customer on the PayPal site. 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.</td>
</tr>
<tr>
<td>Authorized</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>AuthorizedFor-Validation</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>Authorized-Pending</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>Cancelled</td>
<td>string</td>
<td>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().</td>
</tr>
<tr>
<td>Captured</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>New</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
TransactionValidationResponse Subobject

Returned from the Transaction.migrate call, this object describes a specific validation issue with a submitted transaction.

Table 18-17 TransactionValidationResponse Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>string</td>
<td>Required. A numerical code indicating the type of issue that was encountered. Specific codes are listed below.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Required. A human readable description of the issue encountered.</td>
</tr>
<tr>
<td>merchantTransactionId</td>
<td>string</td>
<td>Required. Your unique ID for the submitted transaction.</td>
</tr>
</tbody>
</table>
Table 18-18 TransactionValidationResponse Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>The call succeeded.</td>
</tr>
</tbody>
</table>
| 400         | Your call failed, which could be due to an authentication failure or a CashBox failure to find any objects that match your input. 400 may also be one of the following:  
  • Billing has already been attempted for Transaction ID merchantTransactionId.  
  • Failed to deserialize Transaction.  
  • Invalid Arguments - No transaction object. |
| 403         | The Vindicia server cannot authenticate your request. |
| 404         | One of the following:  
  • Unable to load transaction: no match for merchantTransactionId merchantTransactionId.  
  • 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 unavailable. Retry your call later. |
# 18.3 Transaction Methods

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

## Table 18-19 Transaction Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addressAndSalesTaxFromPayPalOrder</code></td>
<td>Allows you to fetch the billing and shipping addresses from PayPal.</td>
</tr>
<tr>
<td><code>auth</code></td>
<td>Sends this <code>Transaction</code> object to the payment processor for pre-authorization.</td>
</tr>
<tr>
<td><code>authCapture</code></td>
<td>Authorizes and captures this <code>Transaction</code> object in one call.</td>
</tr>
<tr>
<td><code>calculateSalesTax</code></td>
<td>Calculates the sales tax for this <code>Transaction</code> object.</td>
</tr>
<tr>
<td><code>cancel</code></td>
<td>Cancels a batch of previously authorized but not yet captured <code>Transaction</code> objects.</td>
</tr>
<tr>
<td><code>capture</code></td>
<td>Captures a batch of previously authorized <code>Transaction</code> objects.</td>
</tr>
<tr>
<td><code>fetchByAccount</code></td>
<td>Returns one or more <code>Transaction</code> objects whose <code>Account</code> object matches the input.</td>
</tr>
<tr>
<td><code>fetchByAutobill</code></td>
<td>Returns all the <code>Transaction</code> objects for an <code>AutoBill</code> object.</td>
</tr>
<tr>
<td><code>fetchByMerchantTransactionId</code></td>
<td>Returns a <code>Transaction</code> object whose transaction ID assigned by you (<code>merchantTransactionId</code>) matches the input.</td>
</tr>
<tr>
<td><code>fetchByPaymentMethod</code></td>
<td>Returns all the <code>Transaction</code> 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.</td>
</tr>
<tr>
<td><code>fetchByVid</code></td>
<td>Returns a <code>Transaction</code> object whose VID matches the input.</td>
</tr>
<tr>
<td><code>fetchByWebSessionVid</code></td>
<td>Returns a <code>Transaction</code> object whose <code>WebSession</code> VID matches the input.</td>
</tr>
<tr>
<td><code>fetchDelta</code></td>
<td>Returns the <code>Transaction</code> objects whose status has changed since the last <code>fetchDelta</code> call.</td>
</tr>
<tr>
<td><code>fetchDeltaSince</code></td>
<td>Returns the <code>Transaction</code> objects that have been modified since the specified time stamp. (<code>endTimeStamp</code> may also be specified.)</td>
</tr>
<tr>
<td><code>finalizeBokuAuthCapture</code></td>
<td><em>(This method is not in use.)</em></td>
</tr>
<tr>
<td><code>finalizeCarrierBilling</code></td>
<td>Completes the processing the a <code>Transaction</code> with payment method <code>CarrierBilling</code>.</td>
</tr>
<tr>
<td><code>finalizeCustomerAction</code></td>
<td>Completes Transaction processing after your customer finishes payment activities at the payment provider-hosted web pages and is redirected to your site.</td>
</tr>
<tr>
<td><code>finalizePayPalAuth</code></td>
<td>Informs CashBox about the final authorization status of a transaction paid for with a PayPal-based payment method.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>migrate</td>
<td>Allows you to migrate Transactions from a previous billing system to CashBox.</td>
</tr>
<tr>
<td>report</td>
<td><em>(This method is not in use. Use Transaction.migrate to report Transactions to CashBox that have been processed in other billing systems.)</em></td>
</tr>
<tr>
<td>score</td>
<td>Evaluates the risk score or chargeback probability score for this Transaction object.</td>
</tr>
</tbody>
</table>
addressAndSalesTaxFromPayPalOrder

The `addressAndSalesTaxFromPayPalOrder` method allows you to fetch customer billing and shipping addresses from PayPal. This enables CashBox to correctly calculate and apply tax on the current PayPal Transaction. This call is intended for merchants who do not collect address information from their customers. CashBox updates Payment Methods on the Account object with the billing and shipping addresses returned by this call, to be used on the current, and on future, one-time and recurring PayPal Transactions.

---

**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, work with your Vindicia Client Services representative to enable the feature for your CashBox account.

The Shipping Address will always be returned by this call, even without completing these required steps for the Billing Address return.

---

**Input**

`srp`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srp`. A null `srp` returns the complete response.

`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.

```php
$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 can 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 this 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, 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 sufficient funds are available 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, 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 return a qualified success if the otherwise successful transaction experienced a tax-based timeout, and return a 202 error. Given this error, you may choose to proceed with the transaction as is or re-attempt the billing with a new transaction. If you ignore this error and subsequently capture the transaction as is, the related capture will proceed according to your configured handling for Tax Not Available scenarios. To learn about additional options for handling taxes when the tax vendor is off-line, see Appendix B “Handling ‘Tax Service Not Available’ Scenarios” in the `CashBox Programming Guide`.

**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**

`srdf`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.
**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 (applied to all eligible Transaction items and applicable to each individual Transaction item).

**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. (It is not necessary to specify a payment method, as 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 18-22: 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>202</td>
<td>Taxes temporarily unavailable.</td>
</tr>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Must specify line items in transaction to calculate sales tax for auth!</td>
</tr>
<tr>
<td></td>
<td>• Data validation error error-description.</td>
</tr>
<tr>
<td></td>
<td>• Must specify transaction to authorize!</td>
</tr>
<tr>
<td></td>
<td>• Auth attempt failed to return a valid Transaction.</td>
</tr>
<tr>
<td></td>
<td>• Vindicia fault fault-code encountered.</td>
</tr>
<tr>
<td></td>
<td>• Internal-error-description.</td>
</tr>
<tr>
<td></td>
<td>• Data validation error Failed to create Payment-Type-Specific Payment Record: Credit Card conversion failed: Credit Card failed Luhn check.</td>
</tr>
<tr>
<td></td>
<td>• This Credit Card already exists—Policy Violation&quot; (eradicate the newly created but failed one or ensure it is set INACTIVE).</td>
</tr>
</tbody>
</table>

You receive this error message when you have the Credit_Card_Constraints merchant option enabled on your site. This message means that someone is attempting to use a credit card that is already being used as the payment method on an existing account. This is not allowd.

| 407         | Failed AVS policy evaluation. |
| 408         | Failed CVN policy evaluation. |

<table>
<thead>
<tr>
<th>402</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Can't call auth on Boleto associated transaction. Please call authCapture!</td>
</tr>
<tr>
<td></td>
<td>• The transaction ID merchantTransactionId collides with reserved Vindicia namespace, which is: namespace.</td>
</tr>
<tr>
<td></td>
<td>• Unable to create transaction ID consistent with reserved Vindicia namespace, which is: namespace.</td>
</tr>
<tr>
<td></td>
<td>• No payment method found in transaction or account.</td>
</tr>
<tr>
<td></td>
<td>• Transaction already previously authorized!</td>
</tr>
</tbody>
</table>

| 406         | Chargeback risk score is higher than minChargebackProbability, transaction not authorized. |
|             | (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
$chargebackProbability = 70;
$response = $tx->auth($chargebackProbability, 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 
";
}
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, and minChargebackProbability scoring can cause this call to fail even if a successful authorization is received.

**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 return a qualified success if the otherwise successful Transaction experienced a tax-based timeout, and return a 202 error. Given this error, the automatic capture is postponed by a configurable delay (defaults to one hour) during which you may proceed by taking no action, or you can forcibly cancel the transaction. If you ignore this error and subsequently capture the transaction as is, the related capture will proceed according to your configured handling for Tax Not Available scenarios. To learn about additional options for handling taxes when the tax vendor is off line, see Appendix B “Handling ‘Tax Service Not Available’ Scenarios” in the CashBox Programming Guide.

This call is used to process one-time (real-time) transactions through CashBox. Call auth() to preauthorize a customer 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**

**srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**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 (applied to all eligible Transaction items and applicable to each individual Transaction item).

**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. (It is not necessary to specify a payment method, as no payment method validations, authorizations or charges will be performed if dryrun is true.)

**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 authCapture call will fail.
If you do not set `minChargebackProbability`, CashBox defaults to 100, indicating that all transactions are acceptable and no risk screening occurs. For more information on CashBox risk screening, see Section 15: Common ChargeGuard Programming Tasks in the CashBox Programming Guide

**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.

`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**

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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>202</td>
<td>Taxes temporarily unavailable.</td>
</tr>
</tbody>
</table>
| 400         | One of the following:  
  • Must specify line items in transaction to calculate sales tax for auth!  
  • Data validation error error-description.  
  • Must specify transaction to authorize!  
  • Auth attempt failed to return a valid Transaction.  
  • Vindicia fault fault-code encountered.  
  • Internal-error-description.  
  • Data validation error Failed to create Payment-Type-Specific Payment Record: Credit Card conversion failed: Credit Card failed Luhn check.  
  • This Credit Card already exists—Policy Violation" (eradicate the newly created but failed one or ensure it is set INACTIVE). |
| 402         | One of the following:  
  • The transaction ID merchantTransactionId collides with reserved Vindicia namespace, which is: namespace.  
  • Unable to create transaction ID consistent with reserved Vindicia namespace, which is: namespace.  
  • No payment method found in transaction or account.  
  • Transaction already previously authorized! |
| 406         | There are two conditions under which a 406 response code can be returned on a Transaction.capture() and/or Transaction.authCapture() SOAP method call.  
  • The first condition applies to both calls when the risk score calculated by MaxMind exceeds the minChargebackProbability parameter, which the merchant set in the method call. In this situation, CashBox returns the 406 response code with a returnString value: Chargeback risk score is higher than minChargebackProbability, transaction not authorized.  
  • The second condition applies only to the Transaction.capture() method call. It occurs when a merchant calls the method on a transaction that has already been authorized and is pending capture (that is, CashBox billing processes are waiting to initialize a capture cycle to the payment processor that the transaction will be sent to). Repeated capture calls will not speed up the process. |
| 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.
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.

```php
$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
$response = $tx->authCapture($sendEmailNotification, $ignoreAvsPolicy, $ignoreCvnPolicy, $minChargebackProbability);

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 $tx->statusLog[0]->status->creditCardStatus->authCode . "\n";
    }
} else ($response['returnCode'] == 406) {
    print "Transaction not authorized, Chargeback score is ". $response['score'] . "\n";
}
```
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.

```php
$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;
$minChargebackProbability = 30; // 30 percent max
$response = $tx->authCapture($sendEmailNotification, $minChargebackProbability);
if($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
}
if ($response['returnCode'] == 406) {
    print "Transaction not authorized, Chargeback score is " . $response['score'] . "\n";
```
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.

```php
$txn = new Transaction();

// populate the transaction as shown in the previous example
$paymentMethod = new PaymentMethod();
$paymentMethod->setBillingAddress($address);
$paymentMethod->setType('ECP');
$ecp = new ECP();
$ecp->setAccount('123456789');
$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);
```

**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.
$tx->setEcpTransactionType('Inbound');
$sendEmailNotification = false;
$minChargebackProbability = 30; // 30 percent max
$response = $tx->authCapture($sendEmailNotification,
             $minChargebackProbability);
if($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 $ret_tx->statusLog[0]->status->ecpStatus->authCode . "\n";
    }
}
if ($response['returnCode'] == 406) {
    print "Transaction not authorized, Chargeback score is ".
$response['score'] . "%\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.

```php
$tx = new Transaction();
// populate the transaction as shown in earlier examples

$paymentMethod = new PaymentMethod();
pagmentMethod->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');
$tx->setSourcePaymentMethod($paymentMethod);
$sendEmailNotification = false;
$minChargebackProbability = 30; // 30 percent max
$response = $tx->authCapture($sendEmailNotification,
$minChargebackProbability);

if($response['returnCode']==200) {
    if($tx->statusLog[0]->status=='AuthorizationPending') {
        $payPalUrl = $tx->statusLog[0]->payPalStatus->redirectUrl;
        // send customer to a URL for completion of payment
        // process at PayPal's site
    }
} if ($response['returnCode'] == 406) {
    print "Transaction not authorized, Chargeback score is ".
$response['score'] . "\n";
}
```

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.

```php
$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:
$sucess = true;
$response =
    $soap_caller->finalizePayPalAuth($payPalTxId, $success);

if($response['returnCode'] == 200) {
```
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 `calculateSalesTax` call will handle a tax-based timeout, returning a 503 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, sales tax is not calculated.

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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>404</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Address not specified on transaction, and unable to load it from customer accounts - unable to calculate sales tax!</td>
</tr>
<tr>
<td></td>
<td>• Must specify line items in transaction to calculate sales tax!</td>
</tr>
<tr>
<td>503</td>
<td>Tax service temporarily unavailable.</td>
</tr>
</tbody>
</table>
Example

```php
$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**

`srdf`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

- `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 18-20 `CancelResult` Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>merchantTransactionId</td>
<td>string</td>
<td>Your unique identifier for the <code>Transaction</code> object you asked to cancel.</td>
</tr>
<tr>
<td>returnCode</td>
<td>integer</td>
<td>The reason for the success or failure:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 200: <code>cancel()</code> succeeded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 402: The <code>Transaction</code> object has expired and cannot be cancelled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 404: <code>cancel()</code> cannot load the <code>Transaction</code> object, likely because</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the VID or your transaction ID (merchantTransactionId) is invalid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 405: You did not specify an authorized transaction.</td>
</tr>
</tbody>
</table>

### Returns

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Must specify transaction.</td>
</tr>
<tr>
<td></td>
<td>• Unable to save transactions: <strong>error-description</strong>.</td>
</tr>
</tbody>
</table>

### 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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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 18-21 CaptureResult Object Data Members

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>merchantTransactionId</td>
<td>string</td>
<td>Your unique identifier for this <code>Transaction</code> object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Although you normally assign this value, Vindicia might assign it for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>transactions it generates for re-authorization.</td>
</tr>
<tr>
<td>originalMerchantTransactionId</td>
<td>string</td>
<td>Your unique identifier for the original <code>Transaction</code> object in the case of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a reauthorization.</td>
</tr>
<tr>
<td>returnCode</td>
<td>integer</td>
<td>The reason for the success or failure:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 200: capture() succeeded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 402: The Transaction object has expired and cannot be reauthorized by</td>
</tr>
<tr>
<td></td>
<td></td>
<td>capture().</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 404: capture() cannot load the Transaction object, likely because the VID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or your transaction ID (merchantTransactionId) is invalid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 405: You did not specify an authorized transaction.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 500: capture() encountered an internal failure.</td>
</tr>
</tbody>
</table>
In addition to those listed in Table 1: Standard Return Codes, this call returns:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
</table>
| 400         | One of the following:  
|             | • Must specify transaction.  
|             | • Unable to save transactions: error-description. |
| 406         | There are two conditions under which a 406 response code can be returned on a Transaction.capture() and/or Transaction.authCapture() SOAP method call.  
|             | • The first condition applies to both calls when the risk score calculated by MaxMind exceeds the minChargebackProbability parameter, which the merchant set in the method call. In this situation, CashBox returns the 406 response code with a returnString value: Chargeback risk score is higher than minChargebackProbability, transaction not authorized.  
|             | • The second condition applies only to the Transaction.capture() method call. It occurs when a merchant calls the method on a transaction that has already been authorized and is pending capture (that is, CashBox billing processes are waiting to initialize a capture cycle to the payment processor that the transaction will be sent to). Repeated capture calls will not speed up the process. |

**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 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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

*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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load account to search by: No matches.</td>
</tr>
<tr>
<td></td>
<td>• No account specified to load transaction by!</td>
</tr>
<tr>
<td>404</td>
<td>Unable to load account to search by: <code>error-description</code>.</td>
</tr>
</tbody>
</table>
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 = $soap_tx->fetchByAccount($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**

**srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**autobill:** the AutoBill object that serves as the search criterion. You can identify this object with either its VID or your AutoBill ID ()

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load autobill to search by: No matches.</td>
</tr>
<tr>
<td></td>
<td>• No autobill specified to load transaction by!</td>
</tr>
<tr>
<td>404</td>
<td>Unable to load autobill to search by: error-description.</td>
</tr>
</tbody>
</table>
Example

// Create an AutoBill object to represent an existing customer subscription by its id
$autobill = new AutoBill();
$autobill->set('AB101');

// create a transaction object to make the call
$soap_tx = new Transaction();

// fetch the record(s)
$response = $soap_tx->fetchByAutobill($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**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>404</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to load transaction: No match for <code>merchantTransactionId</code> <code>input-merchantTransactionId</code>.</td>
</tr>
<tr>
<td></td>
<td>• Unable to load transaction by <code>merchantTransactionId</code> <code>input-merchantTransactionId</code>. <code>error-description</code>.</td>
</tr>
</tbody>
</table>
// Known transaction id
$txId = "MERCH42202";

// create a transaction object to make the call
$soap_tx = new Transaction();

// fetch the record(s)
$response = $soap_tx->fetchByMerchantTransactionId($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**

`srdf`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

`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:

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

Example

```php
$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**

`srdf`, a sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

`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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>No VID specified to load transaction by.</td>
</tr>
</tbody>
</table>
| 404         | One of the following:  
|             | • Unable to load transaction: No match for VID `input-vid`.  
|             | • Unable to load transaction by VID `input-vid: error-description`. |
Example

```php
// 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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Missing required parameter 'vid'.</td>
</tr>
<tr>
<td>404</td>
<td>Unable to find requested Transaction: No matches.</td>
</tr>
</tbody>
</table>
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 time stamp 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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

- **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 time stamp for the range of Transaction objects fetched.

- **endDate**: the ending time stamp for the range of Transaction objects fetched.

**Returns**

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

```php
$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 time stamp. 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 time stamp 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**

- **srdf**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

- **timestamp**: a time stamp that specifies the date and time on or after which the `Transaction` objects have changed status.

- **endTimestamp**: a time stamp that specifies the upper limit of the date and time before which the `Transaction` objects have changed status.

**Note:** To reduce high loads on the system, the maximum span for a `fetchDeltaSince()` call is 61 days. If you want to fetch an extended amount of data, best practice suggests that you do so via a loop that pulls a day’s worth of data at a time, with another loop internally that handles fetching each resultant page for that day.

- **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 time stamp 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Invalid Arguments - Must specify a valid payment method type, if using that option.</td>
</tr>
<tr>
<td></td>
<td>• Must specify a time stamp to find transactions newer than …</td>
</tr>
<tr>
<td>404</td>
<td>Not Found - No match found for the Payment Method.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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 time stamp when the last time we ran this call.

$since = getLastCallTime();
do {
    // we will not filter returned transactions by end time stamp 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);
**finalizeCarrierBilling**

The `finalizeCarrierBilling` method completes the authorization of a transaction with Carrier Billing as the payment method. Use this method **only** when working with a Transaction that is paid for with a carrier billing-based payment method. When the `Transaction.authCapture()` call returns success, merchants collect, from the customer, the confirmation code sent to the customer by the Carrier. Merchants must then pass this PIN and the `transactionVID` in the `Transaction.finalizeCarrierBilling()` call.

**Input**

- **srdf**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srdf`. A null `srdf` returns the complete response.

- **transactionVid**: Vindicia-generated ID (VID) for the Transaction. This will be available in the Transaction returned by the `AutoBill.update()` or `Transaction.authCapture()` call.

- **confirmationCode**: Set this to the code the carrier sent to your customer’s phone, or to `true` if Boku redirected the customer to your Success web page (`returnUrl`). If Boku redirected the customer to your Failure page, 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. CashBox populates this only if there is no error processing the call.
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**

**srd:** sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the srd. A null srd returns the complete response.

**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**

- `srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

  Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- `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.
$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:

$succeed = true;
$response =
  $soap_caller->finalizePayPalAuth($payPalTxId, $success);

if($response['returnCode'] == 200) {
  $txId = $response['transaction']->getMerchantTransactionId();
  printLog "Transaction authorized: " . $txId;
}

Example
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 can be searched and analyzed, both through the CashBox Portal, 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.

`Transaction.migrate` supports the following payment processors:

- Chase Paymentech
- Merchant e-Solutions
- Litle & Co.
- PayPal

`Transaction.migrate` supports the following payment methods:

- Credit Card
- PayPal
- Merchant Accepted Payment (MAP)

**Input**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

`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:
<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to save transactions: <em>error-description</em>.</td>
</tr>
<tr>
<td></td>
<td>• <em>(This code is returned if an error occurs in the processing of a transaction and it is the only transaction in the batch.)</em></td>
</tr>
<tr>
<td></td>
<td>• One or more Transaction migrations failed.</td>
</tr>
<tr>
<td></td>
<td>• Error descriptions provided in TransactionValidationResponse (contained in the response array of the return).</td>
</tr>
<tr>
<td></td>
<td>• Invalid field(s) for non-recurring Transaction Migration: <em>(invalid fields)</em></td>
</tr>
<tr>
<td></td>
<td>Invalid MigrationTransaction fields (when calling Transaction.migrate): autoBillCycle, merchantBillingPlanId, billingPlanCycle, billingDate, retryNumber.</td>
</tr>
<tr>
<td></td>
<td>Invalid MigrationTransactionItem fields (when calling Transaction.migrate): servicePeriodStartDate, servicePeriodEndDate, merchantAutoBillItemId.</td>
</tr>
<tr>
<td></td>
<td>• Unable to prepare transaction for migration: <em>error</em>.</td>
</tr>
<tr>
<td></td>
<td>• <em>(Details provided in common AutoBill.migrate/Transaction.migrate messages.)</em></td>
</tr>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• MigrationTransaction not provided.</td>
</tr>
<tr>
<td></td>
<td>• Invalid paymentProcessor: <em>paymentProcessor</em>.</td>
</tr>
<tr>
<td></td>
<td>• MigrationTransaction must include at least one statusLog record.</td>
</tr>
<tr>
<td></td>
<td>• Failed to convert salesTaxAddress.</td>
</tr>
<tr>
<td></td>
<td>• Attempt to migrate Transaction which already exists.</td>
</tr>
<tr>
<td></td>
<td>• Unsupported Payment Type: <em>paymentType</em>.</td>
</tr>
<tr>
<td></td>
<td>• Failed to prepare auth_response for Migrated Transactions.</td>
</tr>
<tr>
<td></td>
<td>• Unable to determine currency for migrated Transaction.</td>
</tr>
</tbody>
</table>
|             | • Calculated Transaction amount *(XXX.XX)* does not match input amount *(YYY.YY)* on migrated Transaction.
// 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('ONE TIME CHARGE');
$txItem->setPrice(49.99);
$txItem->setSku('CB-4081');
$txItem->setTaxClassification('DC010500');
// This should be the Avalara tax code associated with this product

Example
$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->setAuthCode('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 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 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 18-22. 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>
<thead>
<tr>
<th>Score Code (ID)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>The city and state in the shipping address do not match the ZIP code.</td>
</tr>
<tr>
<td>15</td>
<td>The city and state in the billing address do not match the ZIP code.</td>
</tr>
<tr>
<td>16</td>
<td>The shipping address is in the database of known risky mail drops.</td>
</tr>
<tr>
<td>21</td>
<td>The country of the issuing bank does not match the country of the billing address.</td>
</tr>
<tr>
<td>31</td>
<td>The password is in the database of high-risk passwords.</td>
</tr>
<tr>
<td>32</td>
<td>The user name is in the database of high-risk user names.</td>
</tr>
<tr>
<td>41</td>
<td>The email address is in the database of high-risk email addresses.</td>
</tr>
<tr>
<td>42</td>
<td>The email address is from a free email provider.</td>
</tr>
<tr>
<td>51</td>
<td>The IP address is in the database of known transparent proxy servers.</td>
</tr>
<tr>
<td>52</td>
<td>The IP address is an anonymous proxy.</td>
</tr>
<tr>
<td>63</td>
<td>The country of the IP address or billing address is a high-risk country.</td>
</tr>
</tbody>
</table>
Table 18-22 Score Code Descriptions  (Continued)

<table>
<thead>
<tr>
<th>Score Code (ID)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>The distance between the IP address and billing address is XX kilometers.</td>
</tr>
<tr>
<td>65</td>
<td>The IP address and billing address are in different countries.</td>
</tr>
<tr>
<td>71</td>
<td>The Account object is associated with known fraudulent (friendly or true-fraud) chargebacks.</td>
</tr>
</tbody>
</table>

**Input**

`srd`: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

**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 18-22: Score Code Descriptions for details.

**Returns**

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Unable to save transactions: <code>error-description</code>.</td>
</tr>
<tr>
<td></td>
<td>• Data validation error: <code>error-description</code>.</td>
</tr>
</tbody>
</table>

**Example**

```php
$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

You create WebSession objects in the context of the Vindicia HOA functionality, in anticipation of submission of the Web order form by a customer who requested it from your server. When 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 by a Vindicia server. Handling such data yourself could mean you would have to be in compliance 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 visible in the form that you serve to the customer, but which 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) that HOA makes 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>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiReturn</td>
<td>Return</td>
<td><strong>Read-only.</strong> 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.</td>
</tr>
<tr>
<td>errorURL</td>
<td>string</td>
<td><strong>Optional.</strong> The URL of your site’s dynamic page, to which HOA redirects the customer’s browser at form submission if initial validation (credit card Luhn check, expiration date does not begin with 20) of the form contents fails. If the HOA form post was made using Ajax, and there was a similar error, this value is returned in a Redirect-URL header instead of the redirect. In either case, HOA includes the VID of the WebSession object when redirecting or returning the headers. 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. If you do not specify this attribute, HOA uses the returnUrl value.</td>
</tr>
<tr>
<td>expireTime</td>
<td>dateTime</td>
<td><strong>Read-only.</strong> The time stamp 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. When you fetch a WebSession object, if the current time is past this time stamp 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.</td>
</tr>
<tr>
<td>ipAddress</td>
<td>string</td>
<td><strong>Required.</strong> 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.</td>
</tr>
<tr>
<td>method</td>
<td>string</td>
<td><strong>Required.</strong> 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. 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.</td>
</tr>
<tr>
<td>Data Members</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>methodParamValues</td>
<td>NameValuePair[]</td>
<td><strong>Optional.</strong> 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. For example, if the call is <code>AutoBill.update</code>, exclude the tolerance threshold in the risk score (<code>minChargebackProbability</code>) at form submission. The name for the value is the flattened object name, method name, and parameter name, concatenated with an underscore, for example, <code>AutoBill_Update_minChargebackProbability</code>. See Section 10: The NameValuePair Object.</td>
</tr>
<tr>
<td>nameValues</td>
<td>NameValuePair[]</td>
<td><strong>Optional.</strong> 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 <code>WebSession</code> object. For example, if that call creates an <code>AutoBill</code> 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 <code>vin:Division</code>. See Section 10: The NameValuePair Object.</td>
</tr>
<tr>
<td>postValues</td>
<td>NameValuePair[]</td>
<td><strong>Read-only.</strong> The name–value pairs stored by HOA in the corresponding <code>WebSession</code> object at form submission by the customer if you include non-Vindicia form elements (elements with no <code>vin</code> prefix in their names) in the order form. On your success or failure page, extract these pairs from the <code>WebSession</code> object you fetch. See Section 10: The NameValuePair Object.</td>
</tr>
</tbody>
</table>
Table 19-1 WebSession Object Data Members (Continued)

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| privateFormValues  | 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. 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. 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. 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. Note: Commas are a special character reserved for use in this data member, and should be used only as separators between multiple possible values for the name of a name-value pair. For example, to create an HOA order form that 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:  

\[
\text{vin\_BillingPlan\_merchantBillingPlanId} = \text{gold}, \text{silver}, \text{platinum}
\]

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. Do not use commas as values in the privateFormValues for any other purpose. See Section 10: The NameValuePair Object.
### Table 19-1 WebSession Object Data Members *(Continued)*

<table>
<thead>
<tr>
<th>Data Members</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>returnURL</td>
<td>string</td>
<td>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. If the HOA form post was made using Ajax, this value is returned in a Redirect-URL header instead of the redirect. When redirecting the customer’s browser to this page, or passing the header, 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.</td>
</tr>
<tr>
<td>version</td>
<td>string</td>
<td>The CashBox API version HOA should use for the call specified in the method attribute. This value must be 3.3 or higher.</td>
</tr>
<tr>
<td>VID</td>
<td>string</td>
<td>The Vindicia Globally Unique Identifier (GUID) for this object. When creating a new WebSession object, leave this field blank; it will be populated automatically by CashBox. 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. This enables HOA to load the corresponding WebSession object when the customer submits the form.</td>
</tr>
</tbody>
</table>
19.2 WebSession Methods

The following table lists and summarizes the methods for the `WebSession` object.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fetchByVid</code></td>
<td>Returns an existing <code>WebSession</code> object whose VID matches the input VID.</td>
</tr>
<tr>
<td><code>finalize</code></td>
<td>Completes HOA activity by instructing HOA to make the API call to create</td>
</tr>
<tr>
<td></td>
<td>CashBox objects containing sensitive payment data. Uses data submitted by</td>
</tr>
<tr>
<td></td>
<td>the order form.</td>
</tr>
<tr>
<td><code>initialize</code></td>
<td>Creates a <code>WebSession</code> object.</td>
</tr>
</tbody>
</table>

**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 once you receive the success or failure status from the form post, which will be 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**

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>No VID specified to load session by.</td>
</tr>
<tr>
<td>404</td>
<td>Unable to load session: No match for VID <code>vid</code>.</td>
</tr>
<tr>
<td>500</td>
<td>Unable to load session by VID <code>vid: error-description</code>.</td>
</tr>
</tbody>
</table>
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. When you call `finalize`, simply pass in the ID of the sparse `webSession` object (the VID). Do not populate any `privateFormValues` or `post` values.

In rare cases, you can pass in `MethodParamValues`—for example if you want to pass in a campaign code to an `AutoBill.update` call after collecting it in the payment form.

Call this method when you receive a success notification from CashBox from the form post within HOA. This will either come as a 302 redirect to the return URL you provided, or in the case of a form post using Ajax, it will be the `returnURL` string in the `Redirect-URL` field of the response header. When the customer submits the form, HOA receives the form data and stores it before redirecting the customer's browser to the success page or passing the return header. 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.

You can pass any CashBox API object attributes with this call that you did not originally pass in the initialize call. For example, if you did not collect the campaign code from the customer in the original purchase page where you made the `WebSession.initialize` call, you can collect it with the payment details, and pass it in with the `WebSession.finalize` call.

When you call `WebSession finalize`, 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

- **srd**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a `method` call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srd`. A null `srd` returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
</table>
| 400         | One of the following:  
  - Unable to load session: \textit{error-description}.  
  - Must specify a WebSession to finalize!  
  - This Credit Card already exists—Policy Violation" (eradicate the newly created but failed one or ensure it is set INACTIVE).  
  - You receive this error message when you have the Credit\_Card\_Constraints merchant option enabled on your site. This message means that someone is attempting to use a credit card that is already being used as the payment method on an existing account. This is not allowed.
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. You only need to pass in the CashBox API object attributes that you have collected thus far in the customer’s flow. Remaining attributes can be passed in with the `WebSession.finalize` call.

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**

- **srds**: sparse response description, a SOAP string (which must be a JSON object), in which you specify the elements you want returned. This parameter enables the calling system to constrain a method call to return only components you specify. This gives you greater control over returned content, and improves response time within the Vindicia platform by reducing the processing needed for the call.

Some fields are required, either practically or in the WSDL, and will be returned regardless of the `srds`. A null `srds` returns the complete response.

- **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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>The following:</td>
</tr>
<tr>
<td></td>
<td>• This Credit Card already exists—Policy Violation&quot; (eradicate the newly created but failed one or ensure it is set INACTIVE).</td>
</tr>
<tr>
<td></td>
<td>You receive this error message when you have the Credit_Card_Constraints merchant option enabled on your site. This message means that someone is attempting to use a credit card that is already being used as the payment method on an existing account. This is not allowed.</td>
</tr>
<tr>
<td>402</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Missing required parameter: version <code>version</code>.</td>
</tr>
<tr>
<td></td>
<td>• Invalid parameter: Unsupported version.</td>
</tr>
<tr>
<td></td>
<td>• Missing required parameter: method.</td>
</tr>
<tr>
<td></td>
<td>• Invalid parameter: Unsupported method.</td>
</tr>
<tr>
<td></td>
<td>• Missing required parameter: <code>returnURL</code>.</td>
</tr>
</tbody>
</table>

**Example**

```bash
// to create a WebSession object
```
```php
$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

// 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
}