def get\_exchange\_info(self):  
 *"""Return rate limits and list of symbols* ***:returns****: list - List of product dictionaries  
  
 .. code-block:: python  
  
 {  
 "timezone": "UTC",  
 "serverTime": 1508631584636,  
 "rateLimits": [  
 {  
 "rateLimitType": "REQUESTS",  
 "interval": "MINUTE",  
 "limit": 1200  
 },  
 {  
 "rateLimitType": "ORDERS",  
 "interval": "SECOND",  
 "limit": 10  
 },  
 {  
 "rateLimitType": "ORDERS",  
 "interval": "DAY",  
 "limit": 100000  
 }  
 ],  
 "exchangeFilters": [],  
 "symbols": [  
 {  
 "symbol": "ETHBTC",  
 "status": "TRADING",  
 "baseAsset": "ETH",  
 "baseAssetPrecision": 8,  
 "quoteAsset": "BTC",  
 "quotePrecision": 8,  
 "orderTypes": ["LIMIT", "MARKET"],  
 "icebergAllowed": false,  
 "filters": [  
 {  
 "filterType": "PRICE\_FILTER",  
 "minPrice": "0.00000100",  
 "maxPrice": "100000.00000000",  
 "tickSize": "0.00000100"  
 }, {  
 "filterType": "LOT\_SIZE",  
 "minQty": "0.00100000",  
 "maxQty": "100000.00000000",  
 "stepSize": "0.00100000"  
 }, {  
 "filterType": "MIN\_NOTIONAL",  
 "minNotional": "0.00100000"  
 } ] } }* ***:raises****: BinanceRequestException, BinanceAPIException  
  
 """* return self.\_get('exchangeInfo')

def get\_symbol\_info(self, symbol):  
 *"""Return information about a symbol* ***:param*** *symbol: required e.g BNBBTC* ***:type*** *symbol: str* ***:returns****: Dict if found, None if not  
  
 .. code-block:: python  
  
 {  
 "symbol": "ETHBTC",  
 "status": "TRADING",  
 "baseAsset": "ETH",  
 "baseAssetPrecision": 8,  
 "quoteAsset": "BTC",  
 "quotePrecision": 8,  
 "orderTypes": ["LIMIT", "MARKET"],  
 "icebergAllowed": false,  
 "filters": [  
 {  
 "filterType": "PRICE\_FILTER",  
 "minPrice": "0.00000100",  
 "maxPrice": "100000.00000000",  
 "tickSize": "0.00000100"  
 }, {  
 "filterType": "LOT\_SIZE",  
 "minQty": "0.00100000",  
 "maxQty": "100000.00000000",  
 "stepSize": "0.00100000"  
 }, {  
 "filterType": "MIN\_NOTIONAL",  
 "minNotional": "0.00100000"  
 }  
 ]  
 }* ***:raises****: BinanceRequestException, BinanceAPIException  
  
 """* res = self.\_get('exchangeInfo')  
  
 for item in res['symbols']:  
 if item['symbol'] == symbol.upper():  
 return item  
  
 return None

def get\_server\_time(self):  
 *"""Test connectivity to the Rest API and get the current server time.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md#check-server-time* ***:returns****: Current server time  
  
 .. code-block:: python  
  
 {  
 "serverTime": 1499827319559  
 }* ***:raises****: BinanceRequestException, BinanceAPIException  
  
 """* return self.\_get('time')

def get\_orderbook\_ticker(self, \*\*params):  
 *"""Latest price for a symbol or symbols.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md#symbol-order-book-ticker* ***:param*** *symbol:* ***:type*** *symbol: str* ***:returns****: API response  
  
 .. code-block:: python  
  
 {  
 "symbol": "LTCBTC",  
 "bidPrice": "4.00000000",  
 "bidQty": "431.00000000",  
 "askPrice": "4.00000200",  
 "askQty": "9.00000000"  
 }  
  
 OR  
  
 .. code-block:: python  
  
 [  
 {  
 "symbol": "LTCBTC",  
 "bidPrice": "4.00000000",  
 "bidQty": "431.00000000",  
 "askPrice": "4.00000200",  
 "askQty": "9.00000000"  
 },  
 {  
 "symbol": "ETHBTC",  
 "bidPrice": "0.07946700",  
 "bidQty": "9.00000000",  
 "askPrice": "100000.00000000",  
 "askQty": "1000.00000000"  
 }  
 ]* ***:raises****: BinanceRequestException, BinanceAPIException  
  
 """* return self.\_get('ticker/bookTicker', data=params, version=self.PRIVATE\_API\_VERSION)

def create\_order(self, \*\*params):  
 *"""Send in a new order  
  
 Any order with an icebergQty MUST have timeInForce set to GTC.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md#new-order--trade* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *side: required* ***:type*** *side: str* ***:param*** *type: required* ***:type*** *type: str* ***:param*** *timeInForce: required if limit order* ***:type*** *timeInForce: str* ***:param*** *quantity: required* ***:type*** *quantity: decimal* ***:param*** *quoteOrderQty: amount the user wants to spend (when buying) or receive (when selling)  
 of the quote asset, applicable to MARKET orders* ***:type*** *quoteOrderQty: decimal* ***:param*** *price: required* ***:type*** *price: str* ***:param*** *newClientOrderId: A unique id for the order. Automatically generated if not sent.* ***:type*** *newClientOrderId: str* ***:param*** *icebergQty: Used with LIMIT, STOP\_LOSS\_LIMIT, and TAKE\_PROFIT\_LIMIT to create an iceberg order.* ***:type*** *icebergQty: decimal* ***:param*** *newOrderRespType: Set the response JSON. ACK, RESULT, or FULL; default: RESULT.* ***:type*** *newOrderRespType: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 Response ACK:  
  
 .. code-block:: python  
  
 {  
 "symbol":"LTCBTC",  
 "orderId": 1,  
 "clientOrderId": "myOrder1" # Will be newClientOrderId  
 "transactTime": 1499827319559  
 }  
  
 Response RESULT:  
  
 .. code-block:: python  
  
 {  
 "symbol": "BTCUSDT",  
 "orderId": 28,  
 "clientOrderId": "6gCrw2kRUAF9CvJDGP16IP",  
 "transactTime": 1507725176595,  
 "price": "0.00000000",  
 "origQty": "10.00000000",  
 "executedQty": "10.00000000",  
 "status": "FILLED",  
 "timeInForce": "GTC",  
 "type": "MARKET",  
 "side": "SELL"  
 }  
  
 Response FULL:  
  
 .. code-block:: python  
  
 {  
 "symbol": "BTCUSDT",  
 "orderId": 28,  
 "clientOrderId": "6gCrw2kRUAF9CvJDGP16IP",  
 "transactTime": 1507725176595,  
 "price": "0.00000000",  
 "origQty": "10.00000000",  
 "executedQty": "10.00000000",  
 "status": "FILLED",  
 "timeInForce": "GTC",  
 "type": "MARKET",  
 "side": "SELL",  
 "fills": [  
 {  
 "price": "4000.00000000",  
 "qty": "1.00000000",  
 "commission": "4.00000000",  
 "commissionAsset": "USDT"  
 },  
 {  
 "price": "3999.00000000",  
 "qty": "5.00000000",  
 "commission": "19.99500000",  
 "commissionAsset": "USDT"  
 },  
 {  
 "price": "3998.00000000",  
 "qty": "2.00000000",  
 "commission": "7.99600000",  
 "commissionAsset": "USDT"  
 },  
 {  
 "price": "3997.00000000",  
 "qty": "1.00000000",  
 "commission": "3.99700000",  
 "commissionAsset": "USDT"  
 },  
 {  
 "price": "3995.00000000",  
 "qty": "1.00000000",  
 "commission": "3.99500000",  
 "commissionAsset": "USDT"  
 }  
 ]  
 }* ***:raises****: BinanceRequestException, BinanceAPIException, BinanceOrderException, BinanceOrderMinAmountException, BinanceOrderMinPriceException, BinanceOrderMinTotalException, BinanceOrderUnknownSymbolException, BinanceOrderInactiveSymbolException  
  
 """* return self.\_post('order', True, data=params)

def order\_limit\_buy(self, timeInForce=TIME\_IN\_FORCE\_GTC, \*\*params):  
 *"""Send in a new limit buy order  
  
 Any order with an icebergQty MUST have timeInForce set to GTC.* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *quantity: required* ***:type*** *quantity: decimal* ***:param*** *price: required* ***:type*** *price: str* ***:param*** *timeInForce: default Good till cancelled* ***:type*** *timeInForce: str* ***:param*** *newClientOrderId: A unique id for the order. Automatically generated if not sent.* ***:type*** *newClientOrderId: str* ***:param*** *stopPrice: Used with stop orders* ***:type*** *stopPrice: decimal* ***:param*** *icebergQty: Used with iceberg orders* ***:type*** *icebergQty: decimal* ***:param*** *newOrderRespType: Set the response JSON. ACK, RESULT, or FULL; default: RESULT.* ***:type*** *newOrderRespType: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 See order endpoint for full response options* ***:raises****: BinanceRequestException, BinanceAPIException, BinanceOrderException, BinanceOrderMinAmountException, BinanceOrderMinPriceException, BinanceOrderMinTotalException, BinanceOrderUnknownSymbolException, BinanceOrderInactiveSymbolException  
  
 """* params.update({  
 'side': self.SIDE\_BUY,  
 })  
 return self.order\_limit(timeInForce=timeInForce, \*\*params)  
  
def order\_limit\_sell(self, timeInForce=TIME\_IN\_FORCE\_GTC, \*\*params):  
 *"""Send in a new limit sell order* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *quantity: required* ***:type*** *quantity: decimal* ***:param*** *price: required* ***:type*** *price: str* ***:param*** *timeInForce: default Good till cancelled* ***:type*** *timeInForce: str* ***:param*** *newClientOrderId: A unique id for the order. Automatically generated if not sent.* ***:type*** *newClientOrderId: str* ***:param*** *stopPrice: Used with stop orders* ***:type*** *stopPrice: decimal* ***:param*** *icebergQty: Used with iceberg orders* ***:type*** *icebergQty: decimal* ***:param*** *newOrderRespType: Set the response JSON. ACK, RESULT, or FULL; default: RESULT.* ***:type*** *newOrderRespType: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 See order endpoint for full response options* ***:raises****: BinanceRequestException, BinanceAPIException, BinanceOrderException, BinanceOrderMinAmountException, BinanceOrderMinPriceException, BinanceOrderMinTotalException, BinanceOrderUnknownSymbolException, BinanceOrderInactiveSymbolException  
  
 """* params.update({  
 'side': self.SIDE\_SELL  
 })  
 return self.order\_limit(timeInForce=timeInForce, \*\*params)  
  
def order\_market(self, \*\*params):  
 *"""Send in a new market order* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *side: required* ***:type*** *side: str* ***:param*** *quantity: required* ***:type*** *quantity: decimal* ***:param*** *quoteOrderQty: amount the user wants to spend (when buying) or receive (when selling)  
 of the quote asset* ***:type*** *quoteOrderQty: decimal* ***:param*** *newClientOrderId: A unique id for the order. Automatically generated if not sent.* ***:type*** *newClientOrderId: str* ***:param*** *newOrderRespType: Set the response JSON. ACK, RESULT, or FULL; default: RESULT.* ***:type*** *newOrderRespType: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 See order endpoint for full response options* ***:raises****: BinanceRequestException, BinanceAPIException, BinanceOrderException, BinanceOrderMinAmountException, BinanceOrderMinPriceException, BinanceOrderMinTotalException, BinanceOrderUnknownSymbolException, BinanceOrderInactiveSymbolException  
  
 """* params.update({  
 'type': self.ORDER\_TYPE\_MARKET  
 })  
 return self.create\_order(\*\*params)  
  
def order\_market\_buy(self, \*\*params):  
 *"""Send in a new market buy order* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *quantity: required* ***:type*** *quantity: decimal* ***:param*** *quoteOrderQty: the amount the user wants to spend of the quote asset* ***:type*** *quoteOrderQty: decimal* ***:param*** *newClientOrderId: A unique id for the order. Automatically generated if not sent.* ***:type*** *newClientOrderId: str* ***:param*** *newOrderRespType: Set the response JSON. ACK, RESULT, or FULL; default: RESULT.* ***:type*** *newOrderRespType: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 See order endpoint for full response options* ***:raises****: BinanceRequestException, BinanceAPIException, BinanceOrderException, BinanceOrderMinAmountException, BinanceOrderMinPriceException, BinanceOrderMinTotalException, BinanceOrderUnknownSymbolException, BinanceOrderInactiveSymbolException  
  
 """* params.update({  
 'side': self.SIDE\_BUY  
 })  
 return self.order\_market(\*\*params)  
  
def order\_market\_sell(self, \*\*params):  
 *"""Send in a new market sell order* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *quantity: required* ***:type*** *quantity: decimal* ***:param*** *quoteOrderQty: the amount the user wants to receive of the quote asset* ***:type*** *quoteOrderQty: decimal* ***:param*** *newClientOrderId: A unique id for the order. Automatically generated if not sent.* ***:type*** *newClientOrderId: str* ***:param*** *newOrderRespType: Set the response JSON. ACK, RESULT, or FULL; default: RESULT.* ***:type*** *newOrderRespType: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 See order endpoint for full response options* ***:raises****: BinanceRequestException, BinanceAPIException, BinanceOrderException, BinanceOrderMinAmountException, BinanceOrderMinPriceException, BinanceOrderMinTotalException, BinanceOrderUnknownSymbolException, BinanceOrderInactiveSymbolException  
  
 """* params.update({  
 'side': self.SIDE\_SELL  
 })  
 return self.order\_market(\*\*params)  
  
def create\_oco\_order(self, \*\*params):  
 *"""Send in a new OCO order  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md#new-oco-trade* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *listClientOrderId: A unique id for the list order. Automatically generated if not sent.* ***:type*** *listClientOrderId: str* ***:param*** *side: required* ***:type*** *side: str* ***:param*** *quantity: required* ***:type*** *quantity: decimal* ***:param*** *limitClientOrderId: A unique id for the limit order. Automatically generated if not sent.* ***:type*** *limitClientOrderId: str* ***:param*** *price: required* ***:type*** *price: str* ***:param*** *limitIcebergQty: Used to make the LIMIT\_MAKER leg an iceberg order.* ***:type*** *limitIcebergQty: decimal* ***:param*** *stopClientOrderId: A unique id for the stop order. Automatically generated if not sent.* ***:type*** *stopClientOrderId: str* ***:param*** *stopPrice: required* ***:type*** *stopPrice: str* ***:param*** *stopLimitPrice: If provided, stopLimitTimeInForce is required.* ***:type*** *stopLimitPrice: str* ***:param*** *stopIcebergQty: Used with STOP\_LOSS\_LIMIT leg to make an iceberg order.* ***:type*** *stopIcebergQty: decimal* ***:param*** *stopLimitTimeInForce: Valid values are GTC/FOK/IOC.* ***:type*** *stopLimitTimeInForce: str* ***:param*** *newOrderRespType: Set the response JSON. ACK, RESULT, or FULL; default: RESULT.* ***:type*** *newOrderRespType: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 Response ACK:  
  
 .. code-block:: python  
  
 {  
 }  
  
 Response RESULT:  
  
 .. code-block:: python  
  
 {  
 }  
  
 Response FULL:  
  
 .. code-block:: python  
  
 {  
 }* ***:raises****: BinanceRequestException, BinanceAPIException, BinanceOrderException, BinanceOrderMinAmountException, BinanceOrderMinPriceException, BinanceOrderMinTotalException, BinanceOrderUnknownSymbolException, BinanceOrderInactiveSymbolException  
  
 """* return self.\_post('order/oco', True, data=params)  
  
def order\_oco\_buy(self, \*\*params):  
 *"""Send in a new OCO buy order* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *listClientOrderId: A unique id for the list order. Automatically generated if not sent.* ***:type*** *listClientOrderId: str* ***:param*** *quantity: required* ***:type*** *quantity: decimal* ***:param*** *limitClientOrderId: A unique id for the limit order. Automatically generated if not sent.* ***:type*** *limitClientOrderId: str* ***:param*** *price: required* ***:type*** *price: str* ***:param*** *limitIcebergQty: Used to make the LIMIT\_MAKER leg an iceberg order.* ***:type*** *limitIcebergQty: decimal* ***:param*** *stopClientOrderId: A unique id for the stop order. Automatically generated if not sent.* ***:type*** *stopClientOrderId: str* ***:param*** *stopPrice: required* ***:type*** *stopPrice: str* ***:param*** *stopLimitPrice: If provided, stopLimitTimeInForce is required.* ***:type*** *stopLimitPrice: str* ***:param*** *stopIcebergQty: Used with STOP\_LOSS\_LIMIT leg to make an iceberg order.* ***:type*** *stopIcebergQty: decimal* ***:param*** *stopLimitTimeInForce: Valid values are GTC/FOK/IOC.* ***:type*** *stopLimitTimeInForce: str* ***:param*** *newOrderRespType: Set the response JSON. ACK, RESULT, or FULL; default: RESULT.* ***:type*** *newOrderRespType: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 See OCO order endpoint for full response options* ***:raises****: BinanceRequestException, BinanceAPIException, BinanceOrderException, BinanceOrderMinAmountException, BinanceOrderMinPriceException, BinanceOrderMinTotalException, BinanceOrderUnknownSymbolException, BinanceOrderInactiveSymbolException  
  
 """* params.update({  
 'side': self.SIDE\_BUY  
 })  
 return self.create\_oco\_order(\*\*params)  
  
def order\_oco\_sell(self, \*\*params):  
 *"""Send in a new OCO sell order* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *listClientOrderId: A unique id for the list order. Automatically generated if not sent.* ***:type*** *listClientOrderId: str* ***:param*** *quantity: required* ***:type*** *quantity: decimal* ***:param*** *limitClientOrderId: A unique id for the limit order. Automatically generated if not sent.* ***:type*** *limitClientOrderId: str* ***:param*** *price: required* ***:type*** *price: str* ***:param*** *limitIcebergQty: Used to make the LIMIT\_MAKER leg an iceberg order.* ***:type*** *limitIcebergQty: decimal* ***:param*** *stopClientOrderId: A unique id for the stop order. Automatically generated if not sent.* ***:type*** *stopClientOrderId: str* ***:param*** *stopPrice: required* ***:type*** *stopPrice: str* ***:param*** *stopLimitPrice: If provided, stopLimitTimeInForce is required.* ***:type*** *stopLimitPrice: str* ***:param*** *stopIcebergQty: Used with STOP\_LOSS\_LIMIT leg to make an iceberg order.* ***:type*** *stopIcebergQty: decimal* ***:param*** *stopLimitTimeInForce: Valid values are GTC/FOK/IOC.* ***:type*** *stopLimitTimeInForce: str* ***:param*** *newOrderRespType: Set the response JSON. ACK, RESULT, or FULL; default: RESULT.* ***:type*** *newOrderRespType: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 See OCO order endpoint for full response options* ***:raises****: BinanceRequestException, BinanceAPIException, BinanceOrderException, BinanceOrderMinAmountException, BinanceOrderMinPriceException, BinanceOrderMinTotalException, BinanceOrderUnknownSymbolException, BinanceOrderInactiveSymbolException  
  
 """* params.update({  
 'side': self.SIDE\_SELL  
 })  
 return self.create\_oco\_order(\*\*params)  
  
def create\_test\_order(self, \*\*params):  
 *"""Test new order creation and signature/recvWindow long. Creates and validates a new order but does not send it into the matching engine.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md#test-new-order-trade* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *side: required* ***:type*** *side: str* ***:param*** *type: required* ***:type*** *type: str* ***:param*** *timeInForce: required if limit order* ***:type*** *timeInForce: str* ***:param*** *quantity: required* ***:type*** *quantity: decimal* ***:param*** *price: required* ***:type*** *price: str* ***:param*** *newClientOrderId: A unique id for the order. Automatically generated if not sent.* ***:type*** *newClientOrderId: str* ***:param*** *icebergQty: Used with iceberg orders* ***:type*** *icebergQty: decimal* ***:param*** *newOrderRespType: Set the response JSON. ACK, RESULT, or FULL; default: RESULT.* ***:type*** *newOrderRespType: str* ***:param*** *recvWindow: The number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 .. code-block:: python  
  
 {}* ***:raises****: BinanceRequestException, BinanceAPIException, BinanceOrderException, BinanceOrderMinAmountException, BinanceOrderMinPriceException, BinanceOrderMinTotalException, BinanceOrderUnknownSymbolException, BinanceOrderInactiveSymbolException  
  
  
 """* return self.\_post('order/test', True, data=params)

def get\_order(self, \*\*params):  
 *"""Check an order's status. Either orderId or origClientOrderId must be sent.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md#query-order-user\_data* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *orderId: The unique order id* ***:type*** *orderId: int* ***:param*** *origClientOrderId: optional* ***:type*** *origClientOrderId: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 .. code-block:: python  
  
 {  
 "symbol": "LTCBTC",  
 "orderId": 1,  
 "clientOrderId": "myOrder1",  
 "price": "0.1",  
 "origQty": "1.0",  
 "executedQty": "0.0",  
 "status": "NEW",  
 "timeInForce": "GTC",  
 "type": "LIMIT",  
 "side": "BUY",  
 "stopPrice": "0.0",  
 "icebergQty": "0.0",  
 "time": 1499827319559  
 }* ***:raises****: BinanceRequestException, BinanceAPIException  
  
 """* return self.\_get('order', True, data=params)  
  
def get\_all\_orders(self, \*\*params):  
 *"""Get all account orders; active, canceled, or filled.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md#all-orders-user\_data* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *orderId: The unique order id* ***:type*** *orderId: int* ***:param*** *limit: Default 500; max 500.* ***:type*** *limit: int* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 .. code-block:: python  
  
 [  
 {  
 "symbol": "LTCBTC",  
 "orderId": 1,  
 "clientOrderId": "myOrder1",  
 "price": "0.1",  
 "origQty": "1.0",  
 "executedQty": "0.0",  
 "status": "NEW",  
 "timeInForce": "GTC",  
 "type": "LIMIT",  
 "side": "BUY",  
 "stopPrice": "0.0",  
 "icebergQty": "0.0",  
 "time": 1499827319559  
 }  
 ]* ***:raises****: BinanceRequestException, BinanceAPIException  
  
 """* return self.\_get('allOrders', True, data=params)  
  
def cancel\_order(self, \*\*params):  
 *"""Cancel an active order. Either orderId or origClientOrderId must be sent.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md#cancel-order-trade* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *orderId: The unique order id* ***:type*** *orderId: int* ***:param*** *origClientOrderId: optional* ***:type*** *origClientOrderId: str* ***:param*** *newClientOrderId: Used to uniquely identify this cancel. Automatically generated by default.* ***:type*** *newClientOrderId: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 .. code-block:: python  
  
 {  
 "symbol": "LTCBTC",  
 "origClientOrderId": "myOrder1",  
 "orderId": 1,  
 "clientOrderId": "cancelMyOrder1"  
 }* ***:raises****: BinanceRequestException, BinanceAPIException  
  
 """* return self.\_delete('order', True, data=params)  
  
def get\_open\_orders(self, \*\*params):  
 *"""Get all open orders on a symbol.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md#current-open-orders-user\_data* ***:param*** *symbol: optional* ***:type*** *symbol: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 .. code-block:: python  
  
 [  
 {  
 "symbol": "LTCBTC",  
 "orderId": 1,  
 "clientOrderId": "myOrder1",  
 "price": "0.1",  
 "origQty": "1.0",  
 "executedQty": "0.0",  
 "status": "NEW",  
 "timeInForce": "GTC",  
 "type": "LIMIT",  
 "side": "BUY",  
 "stopPrice": "0.0",  
 "icebergQty": "0.0",  
 "time": 1499827319559  
 }  
 ]* ***:raises****: BinanceRequestException, BinanceAPIException  
  
 """* return self.\_get('openOrders', True, data=params)  
  
# User Stream Endpoints  
def get\_account(self, \*\*params):  
 *"""Get current account information.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md#account-information-user\_data* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 .. code-block:: python  
  
 {  
 "makerCommission": 15,  
 "takerCommission": 15,  
 "buyerCommission": 0,  
 "sellerCommission": 0,  
 "canTrade": true,  
 "canWithdraw": true,  
 "canDeposit": true,  
 "balances": [  
 {  
 "asset": "BTC",  
 "free": "4723846.89208129",  
 "locked": "0.00000000"  
 },  
 {  
 "asset": "LTC",  
 "free": "4763368.68006011",  
 "locked": "0.00000000"  
 }  
 ]  
 }* ***:raises****: BinanceRequestException, BinanceAPIException  
  
 """* return self.\_get('account', True, data=params)  
  
def get\_asset\_balance(self, asset, \*\*params):  
 *"""Get current asset balance.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md#account-information-user\_data* ***:param*** *asset: required* ***:type*** *asset: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: dictionary or None if not found  
  
 .. code-block:: python  
  
 {  
 "asset": "BTC",  
 "free": "4723846.89208129",  
 "locked": "0.00000000"  
 }* ***:raises****: BinanceRequestException, BinanceAPIException  
  
 """* res = self.get\_account(\*\*params)  
 # find asset balance in list of balances  
 if "balances" in res:  
 for bal in res['balances']:  
 if bal['asset'].lower() == asset.lower():  
 return bal  
 return None  
  
def get\_my\_trades(self, \*\*params):  
 *"""Get trades for a specific symbol.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md#account-trade-list-user\_data* ***:param*** *symbol: required* ***:type*** *symbol: str* ***:param*** *limit: Default 500; max 500.* ***:type*** *limit: int* ***:param*** *fromId: TradeId to fetch from. Default gets most recent trades.* ***:type*** *fromId: int* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 .. code-block:: python  
  
 [  
 {  
 "id": 28457,  
 "price": "4.00000100",  
 "qty": "12.00000000",  
 "commission": "10.10000000",  
 "commissionAsset": "BNB",  
 "time": 1499865549590,  
 "isBuyer": true,  
 "isMaker": false,  
 "isBestMatch": true  
 }  
 ]* ***:raises****: BinanceRequestException, BinanceAPIException  
  
 """* return self.\_get('myTrades', True, data=params)  
  
def get\_system\_status(self):  
 *"""Get system status detail.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/wapi-api.md#system-status-system* ***:returns****: API response  
  
 .. code-block:: python  
  
 {  
 "status": 0, # 0: normal，1：system maintenance  
 "msg": "normal" # normal or System maintenance.  
 }* ***:raises****: BinanceAPIException  
  
 """* return self.\_request\_withdraw\_api('get', 'systemStatus.html')  
  
def get\_account\_status(self, \*\*params):  
 *"""Get account status detail.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/wapi-api.md#account-status-user\_data* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 .. code-block:: python  
  
 {  
 "msg": "Order failed:Low Order fill rate! Will be reactivated after 5 minutes.",  
 "success": true,  
 "objs": [  
 "5"  
 ]  
 }* ***:raises****: BinanceWithdrawException  
  
 """* res = self.\_request\_withdraw\_api('get', 'accountStatus.html', True, data=params)  
 if not res['success']:  
 raise BinanceWithdrawException(res['msg'])  
 return res

def get\_trade\_fee(self, \*\*params):  
 *"""Get trade fee.  
  
 https://github.com/binance-exchange/binance-official-api-docs/blob/master/wapi-api.md#trade-fee-user\_data* ***:param*** *symbol: optional* ***:type*** *symbol: str* ***:param*** *recvWindow: the number of milliseconds the request is valid for* ***:type*** *recvWindow: int* ***:returns****: API response  
  
 .. code-block:: python  
  
 {  
 "tradeFee": [  
 {  
 "symbol": "ADABNB",  
 "maker": 0.9000,  
 "taker": 1.0000  
 }, {  
 "symbol": "BNBBTC",  
 "maker": 0.3000,  
 "taker": 0.3000  
 }  
 ],  
 "success": true  
 }* ***:raises****: BinanceWithdrawException  
  
 """* res = self.\_request\_withdraw\_api('get', 'tradeFee.html', True, data=params)  
 if not res['success']:  
 raise BinanceWithdrawException(res['msg'])  
 return res