merX Server Guide

Last Modified: 06/22/2016
Current Version: 1.0.0.7

merX is an open source project, developed to enable manufacturers and distributors to receive and process many different types of transactions from their business partners. Designed originally to fill a need in the Powersports and Lawn & Garden Industries, the merX server and supporting Generics can be used to provide these services in an almost unlimited way.

merX is comprised of two main pieces. First and foremost, merX is a standardized protocol for communicating information between a manufacturer or supplier and a business system vendor. Secondly, the merX package contains a free server component that has been written in PHP and MYSQL that will allow any vendor to plug it in and start receiving orders and communicating with their clients. They could then simply get the information out of the database to process the orders and then insert records back into the database to push back to their clients.

The next element of the merX server is the support of what we call general purpose packages (Generics), which are built into merX to handle certain tasks. In its current version merX is comprised of:

- **sendorder** (sending purchase orders)
- **getorderstatus** (retrieving updates on previous orders sent)
- **sendpayment** (paying purchase orders electronically)
- **getcredits** (retrieving current credit information)
- **getinventory** (retrieving current inventory information on specific item)
- **getiteminfo** (retrieving dealer specific price/cost information)
- **getvendors** (returns a list of all vendors)
- **sendinventory** (customer sends their current inventory information)
- **sendsales** (customer sends their day's sales data)
- **getmodel** (Retrieves rebates, hold backs on particular model number)
- **senddoorswings** (sending door swing counts)
- **sendconsignmentinventory** (send current consignment inventory counts)
- **sendconsignmentsales** (send current sales data with respect to consignment parts)
- **getpricefile** (standardized updates of current parts, price and supersessions)
- **sendwarranty** (submitting warranty claim data)
- getvehicleinfo (returns model specific information on VIN/Serial)
- getvehiclewarranty (returns warranty, extended warranty, service contracts)
- getvehicleservice (returns known service history on unit)
- submitvehicleservice (allows submitting service history to vendor to share with other dealers)

merX is open source and freely available, therefore, adding support for additional communications can be done without restriction, and requests can be made to modify/add to the available Generics. Generics that are built into merX will remain standardized across the entire project to insure anyone that interfaces to them will always be able to reliably do so.

There is a merX server that is also freely available that is written in the PHP programming language, which makes it a very simple installation onto any platform. We highly recommend using Linux as the primary server platform but anything you have will work. PHP is a proven robust programming language with multi-threaded processes baked into the language itself. This allows merX to handle extreme customer loads in a very simple and straight forward way, keeping the source code understandable and flexible. However, **it is perfectly acceptable to run a merX server on any web-server such as Apache or NGINX using PHP, PERL or any other language you choose**. There is also a GOLang version of the product which would allow for the solution with no web-server installation at all.

The primary communication mechanism for merX to your back-end business system is through the **MariaDB** (MySQL fork) tables themselves. We will provide example queries to send/receive data from these tables and the merX team can also provide additional paid services to assist in the back-end integration of merX into your business system if needed.

Because **MariaDB** (MySQL) can be accessed from all programming languages that run on most popular operating systems, there is little worry that you will have problems communicating with it directly. The **MariaDB** platform utilizes the standard MySQL libraries.

If you wish to replicate the environment that merX was developed in, the following info should help you: Ubuntu v14.04 LTS (linux), MariaDB v5.5, and NGINX (pronounced engine X) v1.4.6 Web Server, PHP v5.5.

Future versions of merX will also have a web based administrative front end that will allow for setting up and maintaining the merX server. Presently, **ALL** merX configurations are done in the Command Line Interface. Currently these services are provided by a set of scripts that run on the server itself. Examples of these scripts are for supplying the server with a list of your current dealers so that it can verify that its communicating with only those dealers/customers.

Given its open source nature, there may be other companies that will eventually offer those services or each vendor/supplier can support their own system given its simplistic nature.

It is also possible to contract **nizeX**, Inc. to host your merX server where orders will enter the system and you would simply use REST/JSON calls to pull each order down and to send updated information back. Each server will stand on its own and will not share information with any other vendor.

Authentication:

Before any packages can be sent into the merX system, you must first authenticate with it. There are two main methods for authenticating based on whether you are a Systems Vendor or a user.

There will be a test server setup at www.merxedi.com that will handle all currently supported API's and will respond immediately with appropriate response codes and data to simulate communicating with a vendor.

1) The dealer setup is a process, and starts with the vendor pushing in a list of their dealer numbers they wish to be able to communicate with. The BSV will then push that dealer number into the merX system with an order or other Generic. merX will check to see if the dealer has already been given a dealer key (UUID) and if not (this would be the very first time a dealer sent an order), then one would be generated and sent back to them. This UUID would then be required for all future communications with the vendor and should be stored locally by the BSV.

Note: It is the combination of Dealer Number and UUID along with the SSL encryption that secures the system. This ensures only the proper dealer is capable of placing an order and that no one else can communicate on their behalf.

Note: Dealer UUID will be stored locally by their respective BSV and passed in with all future communications. Only the very first communication will succeed without a proper UUID.

Note: Vendors will be required to reset Dealer keys using a server script that runs on the merX server if needed.

Note: Dealer numbers will be preloaded by vendors using a script in the merX folder called *loaddealers*. This program will take a single parameter of the file name and will be looking for a .csv list of “dealernumber”,”dealername” on each line. This serves as the initial step in setting up dealers to be able to start sending orders.

Note: Price file data can be communicated using deltas and the built-in processing scripts. merX will have a standardized method for communicating price file data back to the dealer's system, but within each package you can have non-standard fields to pass back additional information to the BSV to communicate with the dealers.

Note: Because price and item data can be sent back in real-time as delta releases, merX keeps up with what version each dealer has, supersession data should be less of an issue and dealer prices should be more up to date. The merX system attempts to strip out all unnecessary data to enable these updates to process quickly. As an example, if an item price changes but the description has not, the system will only send across the part number and its price.

Note: When using price codes, discounts must be in decimal form. Example. 10% off list would be -.1

sendorder:

This transaction type is used to send a Purchase Order to a selected vendor. The response can be a simple HTTP response code that it was received or it could be rejected due to the order having already been sent. Error codes are sent in the message body as the first word in the sentence and will be listed in Appendix B for each event. HTTP status codes are used to tell the dealer's BSV if the request was successful or not. See Appendix A for the list of valid HTTP response codes.

Note: This is currently the first process that must be completed in order to generate a clientkey. A ClientKey is required for all other transaction types and will be returned on the initial call to send order.

Request Type: POST

URL Path: /sendorder

Required POST Variables:

[AccountNumber]	**=> Dealer's account, provided by supplier/vendor.**
[ClientKey]	**=> Client's UUID that was passed back in original order (blank on very first communication)**
[Data]	**=> JSON field that contains all of the purchase order details**

Data JSON Object Format:

[PONumber]*	**=> The dealer's Purchase Order Number (max 20 characters)**
[OrderType]*	**=> 1=For Regular, 2=Seasonal Order**
[Status]	=> Status of shippable items when order is submitted: 0 = Pending, 1 = Ordered
[ShipToFirstName]	=> (max 50 characters)
[ShipToLastName]	=> (max 50 characters)
[ShipToCompany]	=> (max 50 characters)
[ShipToAddress1]	=> (max 50 characters)
[ShipToAddress2]	=> (max 50 characters)
[ShipToCity]	=> (max 50 characters)
[ShipToState]	=> (max 5 characters)
[ShipToZip]	=> (max 15 characters)
[ShipToCountry]	=> (max 3 characters)
[ShipToPhone]	=> (max 15 characters)
[ShipToEmail]	=> (max 50 characters)
[ShipMethod]	=> 1=Regular, 2=2 Day, 3=Next Day
[PaymentMethod]	=> 1=Account, 2=Credit Card, 3=COD
[LastFour]	=> Last 4 of card to use. Card must already be on account

[Items] (Array)
[0] => (Object)

[VendorID]*	**=> Vendor the part belongs to**
[ItemNumber]*	**=> Vendor's part number (max 30 characters)**
[Qty]*	**=> Quantity ordered**
[FillStatus]	=> Not Required but sending 1 in this field tells vendor ONLY ACCEPT IF YOU CAN FILL COMPLETELY and sending 2 means send me what you have but DO NOT BACKORDER anything you don't have.

[Units] (Array)
[0] => (Object)

[VendorID]*	**=> Vendor the part belongs to**
[ModelNumber]*	**=> Vendor's model number or order code**
[Qty]*	**=> Quantity ordered**
[OrderCode]	=> If vendor has a unique part number to reference this unit, it goes here
[Year]	=> Year of unit if required
[Colors]	=> Primary color of unit

* = Required Field

[Details]	=> Special notes or detail of units desired	

If [ShipTo] information is passed, then you should use what is on file with their account. [ShipTo] information different than dealership will assume a drop ship situation.

Response JSON Array:

```
[PONumber]          => The dealer's purchase order number.
[InteralID]         => The purchase order ID assigned by vendor or internal ID of received data.
[ClientKey]         => Returned only if merX server generated a new key which is only done on initial call
[Items] (Array)
          [0]       => (Object)
                    [VendorID]        => Vendor part belongs to
                    [ItemNumber]      => Part number requested to order
                    [Superseded]      => Superseded Part number that is being ordered in place of original
                    [SubVendor]       => Vendor for substitution part
                    [SubNumber]       => Part number item has been substituted to
                    [NLA]             => 1 = item is no longer available
                    [Closeout ]       => 1 = item only available until inventory depleted, then becomes NLA
                    [BackorderQty]    => Qty of items not in stock that will need to be ordered
                    [MSRP]            => Suggested Retail Price of item
                    [Cost]            => Current cost for specific dealer placing order

[Units] (Array)
          [0]       => (Object)
                    [ModelNumber]*    => Vendor's model number or order code
                    [VendorID]        => Vendor part belongs to
                    [OrderCode]       => If vendor has a unique part number to reference this unit it goes here
                    [NLA]             => Unit is no longer available
                    [BackorderQty]    => Qty needing to be ordered by vendor and not readily available
                    [MSRP]            => Suggested Retail Price of unit
                    [Cost]            => Cost of unit to current dealer
```

getorderstatus:

This package type requests an update from merX related to a specific vendor for an order that has already been sent. A response can include the number of boxes and tracking number for each and can even get as detailed as telling which and how many of each item is in each box. HTTP status codes are used to tell the dealer's BSV if the request was successful or not. See Appendix A for the list of valid HTTP response codes.

Request Type: GET

URL: /getorderstatus

Required GET Variables:

[AccountNumber]	**=> Client's account, provided by supplier/vendor.**
[ClientKey]	**=> Client's UUID**
[InternalID]	**=> The Internal ID generated when the original order was sent**

getorderstatus Response JSON Object:

```
[InternalID]              => merX purchase order id
[PONumber]                => Dealer's purchase order number
[Discount]                => Blanket Discounts provided
[ExpectedDelivery]        => Used for seasonal orders to estimate when the product will arrive at dealership
[PayByDiscDate]           => Discount if invoice paid by certain date
[PayByDiscAmt]            => If PayByDiscDate is used, this field used if discount is a dollar amount
[PayByDiscPercent]        => If PayByDiscDate is used, this field used if discount is a percentage amount
[Status]                  => 0=Pending, 1=Ordered, 2=Processing, 3=Pulling, 4=Staging, 5=Shipping, 6=Completed

[Boxes]
        [BoxNumber]               => Box Number (1,2,3...)
        [ShipVendor]              => Shipper (UPS, FedEx, etc..)
        [TrackingNumber]          => The packages tracking number
        [VendorInvoice]           => Vendor's invoice number to reference this box
        [DueDate]                 => Due Date for payment for this specific box/invoice number
        [ShipCost]                => Used if paying shipping per box shipped
        [ShipDate]                => When box was shipped
        [Items]   (Array)
                  [0]       => (Object)
                            [VendorID]                => Vendor the part belongs to
                            [ItemNumber]              => ItemNumber
                            [SupersessionNumber]      => Only exist is super was shipped in place of original
                            [CrossRefNumber]          => Only exist is super was shipped in place of original
                            [CrossRefVendorID]        => Vendor ID to identify origin of cross referenced item
                            [QtyShipped]              => Qty
                            [Cost]                    => Cost

[Units] (Array)
        [0]       => (Object)
                  [VendorID]*       => Vendor the part belongs to
                  [ModelNumber]*    => Vendor's model number or order code
                  [ShipVendor]      => Shipper (UPS, FedEx, etc..)
                  [TrackingNumber]  => The packages tracking number
                  [OrderCode]       => Vendor's unique number for this model unit
                  [Year]            => Year of unit if required
                  [Colors]          => Primary color(s) of unit
                  [Details]         => Special notes or detail of units desired
                  [SerialVIN]       => ID number of unit
```

* = Required Field

```
                    [Cost]              => Cost of unit
                    [ShipCharge]        => Cost of shipping for this specific unit(s)
                    [EstShipDate]       => Estimated date to fulfillment
                    [ShipDate]          => Actual ship date

[Backorder] (Array)
          [0]       => (Object)
                    [ItemNumber]        => ItemNumber
                    [VendorID]          => Vendor the part belongs to
                    [QtyPending]        => Qty needing to be ordered by vendor and not readily available
                    [EstShipDate]       => Estimated date fulfillment will happen
                    [ShipNote]          => Info about shipping
```

sendpayment:

This transaction type is used to send a Purchase Order payment detail to a selected vendor. The response will be a standard HTTP status code for success. Error codes are sent in the message body as the first word in the sentence and will be listed in Appendix B for each event. HTTP status codes are used to tell the dealer's BSV if the request was successful or not. See Appendix A for the list of valid HTTP response codes.

Request Type: POST

URL Path: /sendpayment

Required POST Variables:

[AccountNumber]	**=> Client's account, provided by supplier/vendor**
[ClientKey]	**=> Client's UUID that was passed back in original order (blank on very first communication)**
[Data]	**=> JSON field that contains all of the purchase order details**

Data JSON Object Format:

```
    [POs]     (Array)
              [0]       => (Object)
                        [PONumber]             => Purchase Order Being Paid
                        [InvoiceNumber]        => Vendor Invoice Number
                        [AmountToPay]          => Amount to pay on this purchase order
                        [Units]    (Array)
                                   [0]        => (Object)
                                              [VIN-Serial]      => VIN-Serial that we are paying for
                                              [AmountToPay]     => amount we are paying on this unit


    [CreditMemo] (Array)
              [0]       => (Object)
                        [CreditMemo]      => Credit memo number being used as payment
                        [CreditAmount]    => Amount of this credit memo used on this particular payment


    [PaymentMethods] (Array)
              [0]       => (Object)
                        [CheckNumber]     => Check number being sent
                        [CreditCard]      => Last 4 of credit card to use for payment. (Must be setup with vendor in advance)
                        [ACHTransaction]  => Transaction ID of Electronic Funds Transfer if available  (999 if unknown)
                        [Amount]          => Amount of Payment using this method
```

getcredits:

This package type requests an updated list of available credits that can be used by the dealer. These credits could be for Warranty claims, rebates, holdbacks or any other type. See Appendix A for the list of valid HTTP response codes.

Request Type: GET

URL: /getcredits

Required GET Variables:

[AccountNumber]	**=> Client's account, provided by supplier/vendor.**
[ClientKey]	**=> Client's UUID**

GetCredits Response JSON Object:

```
[Credits] (Array)
        [0]          => (Object)
                [Type]             => 1=Refund, 2= Rebate, 3= Holdback, 4= Warranty
                [TransactionNo]    => Distinct Transaction number for type: warranty would hold warranty claim number, refunds
                would hold vendor invoice number refund was posted against. Rebates and hold backs would contain a VIN-Serial
                number associated with.
                [Amount]           => Amount being given as credit.
                [CreditMemo]       => The credit memo number being given as reference to vendor credit.
```

getiteminfo:

This package allows for the quick retrieval of minimal price and availability information for an item.

Request Type: GET

URL: /getiteminfo

Required GET Variables:

[AccountNumber]	**=> Client's account, provided by supplier/vendor.**
[ClientKey]	**=> Client's UUID**
[VendorID]	**=> Vendor the part belongs to**
[ItemNumber]	**=> Part number we'd like to look up**

Response JSON Object:

[MSRP]	=> Suggested Retail Price
[Cost]	=> Cost of item
[MAP]	=> Minimum Advertised Price
[Category]	=> Category of the item
[Closeout]	=> 0=no, 1=yes
[OrigManufName]	=> OEM of the item itself
[OrigManufNumber]	=> OEM number for the item itself
[NLA]	=> Item is No longer Available
[SupersessionNumber]	=> Identifier of the alternate item number

getinventory:

This method allows a dealer to retrieve from a distributor or manufacturer, current pricing and inventory information on a specific part. See Appendix A for the list of valid HTTP response codes.

Request Type: GET

URL: /getinventory

Required GET Variables:

[AccountNumber]	**=> Client's account, provided by supplier/vendor**
[ClientKey]	**=> Client's UUID that was passed back in original order (blank on very first communication)**
[VendorID]	**=> Vendor the part belongs to**
[ItemNumber]	**=> Part number we'd like to look up**

Response JSON Object:

```
[MSRP]                  => Suggested Retail Price
[Cost]                  => Cost of Item For Requesting Dealer
[Category]              => Category Information For Item
[NLA]                   => (0 not set, 1 no longer available) Is the item no longer available?
[SupersessionNumber]    => Part Number of Superseded Item
[MAP]                   => Minimum Advertised Price of item
[OrigManufName]         => OEM's name for the item
[OrigManufNumber]       => OEM's part number for the item
[Warehouses]
        [0] => Array
        [WarehouseName]         => Label of warehouse or distribution center
        [WarehouseState]        => State warehouse is located in
        [Qty]                   => Number of items
        [DaysToArrive]          => Days in transit


[Images]                => URL for thumbnail of item
        [0]=>array
        [ImageURL]      => web URL to image if available
        [ImageSize]     => 1=thumbnail, 2=medium, 3=large
```

* = Required Field

getvendors:

This package returns a list of supported vendors and vendor id's to use when communicating with this supplier. An example might be we are communicating with a distributor that actually handles multiple brands. It will return the id's they are expecting you to use to represent each of their brands.

Request type: GET

URL: /getvendors

Required Variables:

[AccountNumber]	**=> Dealer's account, provided by supplier/vendor**
[ClientKey]	**=> Client's UUID that was passed back in original order (blank on very first communication)**

JSON response object:

[VendorID]	=> Vendor ID
[VendorName]	=> Associated name of vendor

sendinventory:

This method allows a dealer to send current inventory values to a vendor and can be done within any time constraint agreed upon between the two. For example: The “Lizzy” BMS system is capable of sending real time inventory feeds to vendor's as items are being sold, or it can provide a monthly dump if that is desired. The merX system supports any type time constraint needed. See Appendix A for the list of valid HTTP response codes.

Request Type: POST

URL: /sendinventory

Required POST Variables:

```
        [AccountNumber]        => Client's account, provided by supplier/vendor
        [ClientKey]            => Client's UUID that was passed back in original order (blank on very first communication)
        [Data]                 => Object for holding the JSON Array elements themselves
```

Data JSON Object Format:

```
        [Items]    (Array)
                [0]        => (Object)
                        [VendorID]          => Vendor Code of Item
                        [ItemNumber]        => Part Number
                        [Qty]               => Current Stock Qty
                        [Cost]              => Cost of inventory
                        [Allocated]         => How many of the item are already spoken for?
                ...
        [Serialized] (Array)
                [0]        => (Object)
                        [VendorID]          => Vendor Code for unit
                        [OrderCode]         => Vendor specific part number for this unit model
                        [ModelNumber]       => Model Number
                        [VIN-Serial]        => VIN or serial of unit
                        [NURD]              => 1 New, 2 Used, 3 Rental, 4 Demo ( and yes, we know we spelled NURD :-)
                        [Cost]              => Amount paid for the unit
                        [Assembled]         => Is the unit assembled yet or is it still in the crate?
                        [OnDeal]            => Is the unit currently on a pending deal for a customer?
```

sendsales:

This method allows a dealer to send sales data to a vendor and can be done within any time constraint agreed upon between the two. For example: The "Lizzy" BMS system is capable of sending real time sales feeds to vendor's as items are being sold, or it can provide a monthly dump if that is desired. The merX system supports any type time constraint needed. See Appendix A for the list of valid HTTP response codes.

Request Type: POST

URL: /sendsales

Required POST Variables:

```
        [AccountNumber]          => Dealer's account, provided by supplier/vendor
        [Password]               => Dealer's password, set when dealer registration request was processed
        [BSVKey]                 => BSV authentication key
        [Data]                   => Holds JSON Object of sales data
```

Data JSON Object Format:

```
        [Items] (Array)
                [0]     => (Object)
                        [VendorID]              => Vendor the part belongs to
                        [ItemNumber]            => Part number sold
                        [Qty]                   => Quantity sold
                        [SoldFor]               => Individual Price Sold For. (Must multiply Qty * SoldFor if you want extended)
                        [SoldOn]                => 1=Serialized Sale, 2=Over Counter, 3=Internet Sale, 4=Service

        [Serialized] (Array)
                [0]     => (Object)
                        [InvoiceNumber]   => Dealers Invoice Number the Unit was sold on
                        [VendorID]        => Vendor Code of Unit
                        [OrderCode]       => Vendor specific part number for this unit model
                        [ModelNumber]     => Model Number of Unit
                        [VIN-Serial]      => VIN or serial number of sold unit
                        [NURD]            => 1 New, 2 Used, 3 Demo
                        [Year]            => Model year of sold unit

                        [SoldFor]         => Price Unit was Sold For
                        [AttachmentTotal] => Sum of attachments added to deal

                        [Attachments] (Array)
                                [0]     => (Object)
                                        [VendorName]            => Name of vendor
                                        [ItemNumber]            => Part number sold
                                        [Qty]                   => Quantity sold

                        [SoldTo]=> (Object)
                                [ContactID]       => Unique ID for the customer
                                [Type]            => Customer type that purchased the unit ( business, landscaper, racer...)
                                [BusinessName]    => Name of company if company purchased unit
                                [FirstName]       => Customer first name
                                [MiddleName]      => Middle name of customer
                                [LastName]        => Last name of customer
                                [Address1]        => Address of primary purchaser
                                [Address2]        => Secondary Address
                                [City]            => City of customer
                                [State]           => State or Province of customer
```

* = Required Field

[PostalCode]	=> Zip code or postal code of customer
[Country]	=> Country of the customer

getmodel:

This method allows dealers to get updated model numbers and rebate information. See Appendix A for the list of valid HTTP response codes.

Request Type: GET

URL: /getmodel

Required GET Variables:

[AccountNumber]	**=> Clients's account, provided by supplier/vendor**
[ClientKey]	**=> Client's UUID that was passed back in original order (blank on very first communication)**
[VendorID]	**=> Vendor that you want to retrieve**
[Year]	**=> Model year that you want to retrieve**

Optional GET Variable:

[OrderCode]	=> Vendor specific part number for this unit model
[ModelNumber]	=> Specific model To Lookup (Leaving blank will return list of models for year)

Response JSON Object:

```
[OrderCode]                 => Item number referring to this specific model for ordering purposes
[ModelName]                 => Name for model, if applicable
[NLA]                       => No Longer Available
[CloseOut]                  => Closeout status of the unit
[Cost]                      => Model cost
[MSRP]                      => Suggested Retail Price
[Description]               => Model description

[Images]                                  => URL for thumbnail of item
        [0]=>array
                [ImageURL]          => web URL to image if available
                [ImageSize]         => 1=thumbnail, 2=medium, 3=large

[Rebate] ( Array )
        [0]     => (Object)
                [Name]              => Name of rebate program
                [DealerPercent]     => Percentage of rebate that goes to dealer
                [CustomerPercent]   => Percentage of rebate that goes to customer
                [Amount]            => Rebate amount
                [StartDate]         => Date rebate starts
                [EndDate]           => Date rebate ends
                …
[Colors] (Array)
        [0]     => (Object)
                [PrimaryColor]      => Primary color
                [SecondaryColor]    => Secondary color
        ...
```

senddoorswings:

This method allows dealers to send their door swings to a vendor and basically includes the date and the swing count. See Appendix A for the list of valid HTTP response codes.

Request Type: POST

URL: /senddoorswings

Required POST Variables:

AccountNumber	**=> Client's account, provided by supplier/vendor**
ClientKey	**=> Clients's UUID that was passed back in original order (blank on very first communication)**
DateOfSwings	**=> Date the swings were recorded**
SwingCount	**=> Swing count for that day**

sendconsignmentinventory:

This method allows dealers to send their current consignment inventory counts to the supplier which is basically the unsold items they've been given to sell. See Appendix A for the list of valid HTTP response codes.

Request Type: POST

URL: /sendconsignmentinventory

Required POST Variables:

[AccountNumber]	**=> Client's account, provided by supplier/vendor.**
[ClientKey]	**=> Client's UUID that was passed back in original order (blank on very first communication)**
[Data] (Array)	
[0]	=> (Object)
[ItemNumber]	=> Item Number on Consignment
[Qty]	=> Current Qty in stock for the item at the dealership

* = Required Field

sendconsignmentsales:

This method allows dealers to send their current consignment sales counts to the supplier which is basically the sold items they've been given to sell since the last time sent. See Appendix A for the list of valid HTTP response codes.

Request Type: POST

URL: /sendconsignmentsales

Required POST Variables:

```
    [AccountNumber]         => Client's account, provided by supplier/vendor
    [ClientKey]             => Client's UUID that was passed back in original order (blank on very first communication)
    [Data] (Array)
        [0]     => (Object)
                [ItemNumber]            => Item Number on Consignment
                [QtySold]               => Current Qty Sold since last payment was made for the item at the dealership
```

getpricefile:

This method allows dealers to request price file updates from the system. See Appendix A for the list of valid HTTP response codes.

Note: merX maintains a database of parts data along with dealer specific costs based on the requirements of the vendor. These tables can be updated daily and it maintains the last date each item has been effected or added to the system. merX supports calling into this system and passing in a date in order to receive a delta response on everything that has changed for that dealer since the passed in date.
Each merX server has customized scripts to handle these price file updates. These scripts may be specialized by each vendor and are not themselves part of the core merX system. While the price file update process itself is considered a “Generic”, the process by which the deltas are built is not.

Request Type: GET

URL: /getpricefile

Required POST Variables:

[AccountNumber]	**=> Client's account, provided by supplier/vendor**
[ClientKey]	**=> Client's UUID that was passed back in original order (blank on very first communication)**
[RequestDate]	**=> Date to work from for the deltas**

Response JSON Object:

```
Object
        (
        [<VendorID>] => Object
                (
                [<Parts>] => Object
                        (
                        [ItemNumber]        => Number of item
                        [Description]       => Description of item
                        [MSRP]              => Suggested Retail
                        [Cost]              => Dealer specific cost
                        [Weight]            => weight of item
                        [Supersession]      => Part item supersedes to
                        [NLA]               => 1 passed in if item has been obsoleted
                        [Size]              => if item is related to a size then what that size is
                        [Color]             => If item has a color then what is the color?
                        [Category1]         => if category information is available then what is it?
                        [Category2]         => if additional category is available then what is it?
                        [Category3]         => if additional category is available then what is it?
                        [ImageURL]          => If image data is available then where can it be located?
                        )
                )
        )
```

Appendix A

HTTP Status Codes:
500 – Internal Server Error, there was a problem with the merX server contact support.
401 – Unauthorized, dealer authentication failed, details will be in the response body.
400 – Bad Request, request was not formatted correctly, details will be in the response body.
201 – Created, record created body will not contain any data.
200 – OK, response body will contain a JSON object with the requested data.

Appendix B

Error Codes (returned in HTTP message body):
1000 – UUID Not Valid
1001 – BSV Key Not Valid
1002 – Dealer Number Not Valid

2000 – Purchase Order already sent and pulled by vendor.

Appendix C

A test inventory is included complete with detailed parts for confirming your installation. Part numbers as follows:
53-04855, 550-0138, 730003, 2-B10HS, 87-9937