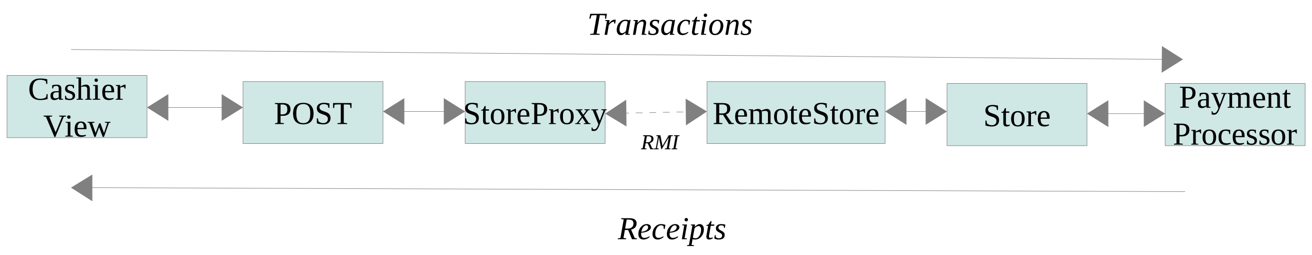
Client-Server Documentation

The connection between the client (POST) and server (Store) is established and maintained as follows:

- The server creates an object implementing the Store interface: This represents the store itself, and will ultimately be made available to clients. Note that Store provides the ability to retrieve a product catalog, a description of the store, and also permits transactions to be submitted to the store.
- The server creates an object implementing RemoteStore, which delegates all calls to the underlying Store object. The RemoteStore interface differs from Store only in extending Remote (to be eligible for RMI) and in declaring that its methods may throw RemoteExceptions. The RemoteStore is this bound via the RMI registry.
- The client, on start up, retrieves a reference to the RemoteStore from the server. This is wrapped in a StoreProxy object, which implements the plain Store interface by providing handling for RemoteExceptions. This includes queuing transactions that cannot be submitted, and caching copies of the product catalog and store description.
- The StoreProxy is given, as a Store, to a POST objects, which is then given to a CashierView (which represents the user interface.) The view assembles transactions via the post, and then these are submitted down the chain of classes until they reach the underlying Store at the server, which responds with a Receipt that makes its way back up to the view.

Note that a benefit of having the otherwise redundant Store and RemoteStore interfaces is that the majority of the system can be implemented to deal with what appears to be a plain Store, without having to worry about any details related to connecting remotely (such as catching RemoteExceptions).



The following table describes the communication of objects between client and server:

Main objects shipped from client (POST) to server (Store)	**Main objects shipped from server (Store) to client (POST)**
	RemoteStore*: Exposes functionality & state of the store to the client. This includes product catalog & store description, as well as a method for reporting back transactions from the POST.
Transaction: Transaction objects are reported from the POST to the Store in order to record a sale of items to a customer, as well as payment information.	**Receipt:** Receipt objects are given from the Store to POST in response to approved transactions.
LineItem: Describes the quantity of purchase for a given item. A Transaction contains one LineItem for each unique product purchased.	**ProductCatalog:** Provides a description of all products available. Sent to the POST on request as a complete copy (allowing it to be cached and consulted in the event that a connection is temporarily unavailable.)
Payment *(and subclasses CashPayment, CreditPayment, and CheckPayment):* Describes payment received at the point of sale. Submitted as part of a transaction.	**ProductSpecification:** An entry in the ProductCatalog. Describes UPC and unit price.
	StoreDescription: Contains descriptive information of the store, such as its name and address. Sent to the POST on request as a copy (allowing it to be cached and consulted in the event that a connection is temporarily unavailable)

* RemoteStore is published by the Server to the Client as a remote object via RMI; that is, calls to the RemoteStore's methods by the client invoke those methods on the server side. All other objects above are transmitted as copies.

The following table summarizes, by class, the types of objects located on the client (POST) and server (Store). Interfaces are italicized for distinction.

Objects located on client (POST)	**Objects located on server (Store)**	**Objects appearing on both**
post.client.Client	post.remote.RemoteStoreImpl	post.model.CashPayment
post.client.controller.POST	post.server.controller.PaymentProcessor	post.model.CheckPayment
post.client.controller.POSTImpl	post.server.controller.SalesLog	post.model.CreditPayment
post.client.controller.POSTInitializer	post.server.controller.StoreInitializer	post.model.LineItem
post.client.controller.StoreProxy	post.server.controller.StoreServerImpl	post.model.Payment
post.client.controller.TransactionBuilder	post.server.Server	post.model.PaymentOption
post.client.view.CashierView	post.model.ProductReader	post.model.ProductCatalog
post.client.view.FileBasedCashier		post.model.ProductSpecification
post.client.view.gui.CashierFrame		post.model.Receipt
post.client.view.gui.CashierGUI		post.model.StoreDescription
post.client.view.gui.CustomerArea		post.model.Transaction
post.client.view.gui.GUIMediator		*post.remote.RemoteStore*
post.client.view.gui.LineItemArea		*post.store.Store*
post.client.view.gui.LineItemModel		*post.store.TransactionReceiver*
post.client.view.gui.PaymentArea		
post.client.view.gui.ProductArea		
post.client.view.gui.ReceiptView		
post.client.view.gui.TimeArea		
post.client.view.gui.TransactionHandler		
post.client.view.gui.TransactionThread		
post.client.view.TransactionReader		

In general, the location of objects can be understood as follows:

- **Located on client (POST):** Objects related to the user interface of the POST; objects related to the internal implementation of the POST; and objects related to establishing a connection to the server.
- **Located on server (Store):** Objects related to creating and implementing the Store's internal behavior; objects related to the store's canonical inputs and outputs, such as the sales log; and objects related to making the Store available to clients.
- **Located on both:** Objects describing the essential business data, of mutual interest to the Store and the POST; interfaces which define the flow of transactions and receipts, which are common to both Store and POST.