CS 553 Spring 2004

Web service descriptions

Table of contents

Overview Diagram

<table>
<thead>
<tr>
<th>Service</th>
<th>Responsible</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purchasing</td>
<td>Stan</td>
<td>A-1</td>
</tr>
<tr>
<td>Accounting</td>
<td>Jiangpen</td>
<td>C-1</td>
</tr>
<tr>
<td>Human Resources</td>
<td>Amit</td>
<td>H-1</td>
</tr>
<tr>
<td>Inventory</td>
<td>Yufei</td>
<td>I-1</td>
</tr>
<tr>
<td>Sales</td>
<td>Mike W.</td>
<td>S-1</td>
</tr>
<tr>
<td>Project Management</td>
<td>Vijay</td>
<td>P-1</td>
</tr>
<tr>
<td>Manufacturing</td>
<td>Scott</td>
<td>M-1</td>
</tr>
<tr>
<td>Deployment</td>
<td>Mike P.</td>
<td>D-1</td>
</tr>
<tr>
<td>Asset management</td>
<td>Rich P.</td>
<td>F-1</td>
</tr>
<tr>
<td>Trouble Ticketing</td>
<td>John</td>
<td>T-1</td>
</tr>
</tbody>
</table>
Services Overview Diagram

- Purchasing
  - Stan
- Accounting
  - Jiangpen
- Human Resources
  - Amit
- Inventory
  - Yufei
- Sales
  - Mike W
- Project Management
  - Vijay
- Manufacturing
  - Scott
- Deployment
  - Mike P.
- Asset Management
  - Rich P.
- Trouble Ticketing
  - John
PURCHASING

Types

*PurchaseOrder*

Purchasing requires keeping track of every purchase order either pending or complete.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PurchaseOrderNum</td>
<td>- int</td>
</tr>
<tr>
<td>VendorID</td>
<td>- int</td>
</tr>
<tr>
<td>VendorPartNum</td>
<td>- string</td>
</tr>
<tr>
<td>BuyerID</td>
<td>- int</td>
</tr>
<tr>
<td>ShippingCode</td>
<td>- int</td>
</tr>
<tr>
<td>OrderStatusCode</td>
<td>- int</td>
</tr>
<tr>
<td>OrderDate</td>
<td>- dateTime</td>
</tr>
<tr>
<td>PricePerUnit</td>
<td>- double</td>
</tr>
<tr>
<td>QuantityOrdered</td>
<td>- int</td>
</tr>
</tbody>
</table>

*OrderStatus*

Each order must be tracked and in a known state (completed, shipped, etc.).

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrderStatusCode</td>
<td>- int</td>
</tr>
<tr>
<td>StatusDescription</td>
<td>- string</td>
</tr>
</tbody>
</table>

*ShippingMethod*

The shipment method of each purchase must be tracked to anticipate arrivals.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShippingCode</td>
<td>- int</td>
</tr>
<tr>
<td>MethodDescription</td>
<td>- string</td>
</tr>
<tr>
<td>ArrivalLocation</td>
<td>- string</td>
</tr>
</tbody>
</table>
**Buyer**

Each purchase order must be linked with a buyer for billing and delivery purposes.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuyerID</td>
<td>- int</td>
</tr>
<tr>
<td>DepartmentID</td>
<td>- int</td>
</tr>
<tr>
<td>ContactName</td>
<td>- string</td>
</tr>
<tr>
<td>ContactPhone</td>
<td>- string</td>
</tr>
</tbody>
</table>

**Vendor**

Vendor information is necessary to track where each piece of equipment was purchased.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VendorID</td>
<td>- int</td>
</tr>
<tr>
<td>Name</td>
<td>- string</td>
</tr>
<tr>
<td>StreetAddress</td>
<td>- string</td>
</tr>
<tr>
<td>City</td>
<td>- string</td>
</tr>
<tr>
<td>State</td>
<td>- string</td>
</tr>
<tr>
<td>ContactName</td>
<td>- string</td>
</tr>
<tr>
<td>ContactPhone</td>
<td>- string</td>
</tr>
</tbody>
</table>

**Return**

Necessary to track the status of each return request.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReturnID</td>
<td>- int</td>
</tr>
<tr>
<td>PurchaseOrderNum</td>
<td>- int</td>
</tr>
<tr>
<td>ReturnDescription</td>
<td>- string</td>
</tr>
<tr>
<td>ReturnDate</td>
<td>- dateTime</td>
</tr>
</tbody>
</table>

Types needed from other services:

**Department** type with a unique Department ID field (int)
- Human Resources
Methods

Purchase Order Processing

CreatePurchaseOrder(VendorID, VendorPartNum, BuyerID, ShippingMethod, OrderStatus, OrderDate, PricePerUnit, QuantityOrdered)
- Create a new purchase order. Purchase order ID is automatically generated.

CancelPurchaseOrder(PurchaseOrderNum) - cancels the specified purchase

GetVendorID(PurchaseOrderNum) - returns the VendorID
SetVendorID(PurchaseOrderNum, VendorID) - sets the VendorID

GetVendorPartNumber(PurchaseOrderNum) - returns the Vendor part number
SetVendorPartNumber(PurchaseOrderNum, VendorPartNum) - sets the Vendor part number

GetBuyerID(PurchaseOrderNum) - returns the BuyerID
SetBuyerID(PurchaseOrderNum, BuyerID) - sets the BuyerID

GetShippingMethod(PurchaseOrderNum) - returns the shipping method code
SetShippingMethod(PurchaseOrderNum, ShippingMethod) - sets the shipping method code

GetOrderStatus(PurchaseOrderNum) - returns the order status
SetOrderStatus(PurchaseOrderNum, OrderStatus) - sets the order status

GetOrderDate(PurchaseOrderNum) - returns the purchase date
SetOrderDate(PurchaseOrderNum, OrderDate) - sets the purchase date

GetPrice(PurchaseOrderNum) - returns the purchase price per unit
SetPrice(PurchaseOrderNum, PricePerUnit) - sets the purchase price per unit

GetQuantityOrdered(PurchaseOrderNum) - returns the quantity ordered
SetQuantityOrdered(PurchaseOrderNum, QuantityOrdered) - sets the quantity ordered

DisplayPurchaseOrder(PurchaseOrderNum) - displays the specified purchase order info
DisplayAllPurchaseOrders( ) - displays all purchase orders

Vendors

AddVendor(Name, Street, City, State, Contact, ContactPhone) - adds a new vendor
RemoveVendor(VendorID) - removes the specified vendor

GetVendorName(VendorID) - returns the vendor name
SetVendorName(VendorID, Name) - sets the vendor name

GetVendorStreet(VendorID) - returns the vendors street address
SetVendorStreet(VendorID, StreetAddress) - sets the vendors street address

GetVendorCity(VendorID) - returns the vendors city
SetVendorCity(VendorID, City) - sets the vendors city

GetVendorState(VendorID) - returns the vendors state
SetVendorState(VendorID, State) - sets the vendors state

GetVendorContact(VendorID) - returns the vendors contact name
SetVendorContact(VendorID, ContactName) - sets the vendors contact name

GetVendorContactPhone(VendorID) - returns the vendors contact phone number
SetVendorContactPhone(VendorID, ContactPhone) - sets the vendors contact phone #

DisplayVendor(VendorID) - displays the specified vendors info
DisplayAllVendors( ) - displays info for all vendors

Returns

ProcessReturn(PurchaseOrderNum, ReturnDescription, ReturnDate) - process a return on the given PO, ReturnID is automatically generated

GetReturnID(PurchaseOrderNum) - returns the ReturnID if it is not null

GetReturnDescription(ReturnID) - returns the reason for the return
SetReturnDescription(ReturnID, ReturnDescription) - sets the reason for the return

GetReturnDate(ReturnID) - returns the return date
SetReturnDate(ReturnID, ReturnDate) - sets the return date

DisplayReturn(ReturnID) - displays the specified return info
DisplayAllReturns( ) - displays all return info

OrderStatus

AddOrderStatus(StatusDescription) - adds a new order status, code automatically generated
RemoveOrderStatus(OrderStatusCode) - removes the specified order status

GetOrderStatusDescription(OrderStatusCode) - returns the status description
SetOrderStatusDescription(OrderStatusCode, StatusDescription) - sets the status description

DisplayOrderStatus(OrderStatusCode) - displays specified order status info
DisplayAllOrderStatus( ) - displays all order status info

ShippingMethods

AddShippingMethod(StatusDescription, ArrivalLocation) - adds a new shipping method
RemoveShippingMethod(ShippingCode) - removes the specified shipping method

GetShippingMethodDescription(ShippingCode) - returns shipping description
SetShippingMethodDescription(ShippingCode, StatusDescription) - returns shipping description
GetShippingArrival(ShippingCode) - returns the arrival location
SetShippingArrival(ShippingCode, ArrivalLocation) - sets the arrival location
DisplayShippingMethod(ShippingCode) - displays specified shipping info
DisplayAllShippingMethods( ) - displays all shipping method info

**Buyers**

AddBuyer(Department, ContactName, ContactPhone) - adds a new buyer
RemoveBuyer(BuyerID) - removes the specified buyer
GetBuyerDepartment(BuyerID) - returns the buyers department
SetBuyerDepartment(BuyerID, Department) - sets the buyers department
GetBuyerContactName(BuyerID) - returns the contact name for the buyer
SetBuyerContactName(BuyerID, ContactName) - sets the contact name for the buyer
GetBuyerContactNumber(BuyerID) - returns the contact phone number for the buyer
SetBuyerContactNumber(BuyerID, ContactPhone) - sets the contact phone number for the buyer
DisplayBuyer(BuyerID) - displays specified buyer info
DisplayAllBuyers( ) - displays info for all buyers

**Justification of Types and Methods**

The purchasing types and methods used in this web service are based on the business model found in the SAP tutorial in addition to commercial purchase order software packages. The Buyer and Purchase types and methods are based on those used in the Purchase Order software package by Cougar Mountain Software. Vendor and Shipping methods were also based on this package in addition to KDI Information Systems Support Purchase Orders documentation.

**Simulated Load**

The initial load of the purchasing service will be loaded from a flat file. This initial load will be predominantly populated with completed purchase orders but will also include new and in progress orders. A client program will simulate the daily interactions of the purchasing service. The client program will use a flat file containing a list of method calls and queries as input to simulate the various day to day interactions with the service.
Objects:

\textit{pense}

\textit{pense} represents expenses occurred during all purchases.

Assumptions:
- Amount is always paid in full (no partial payment)
- The vender specified by \textit{Vender} \textit{d} is the payee
- \textit{Vender} object is stored by the \textbf{Purchasing} \textit{S}
- Ignore late fee, etc.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExpenseId</td>
<td>int</td>
</tr>
<tr>
<td>PONumber</td>
<td>int</td>
</tr>
<tr>
<td>Amount</td>
<td>double</td>
</tr>
<tr>
<td>VenderId</td>
<td>int</td>
</tr>
<tr>
<td>EmployeeId</td>
<td>int</td>
</tr>
<tr>
<td>ScheduledDate</td>
<td>date</td>
</tr>
<tr>
<td>PaidDate</td>
<td>date</td>
</tr>
<tr>
<td>IsPaid</td>
<td>Boolean</td>
</tr>
<tr>
<td>CheckId</td>
<td>int</td>
</tr>
</tbody>
</table>

\textit{Invoice}

\textit{Invoice} represents invoices that are going to be sent to customers.

Assumptions:
- All payers are considered as customers
- Amount is always paid in full (no partial payment)
- The customer specified by \textit{us o er} \textit{d} is the payer
- \textit{us o er} object is stored by the \textbf{Sales} \textit{S}
- Ignore late fee, etc.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvoiceId</td>
<td>int</td>
</tr>
<tr>
<td>InvoiceNumber</td>
<td>int</td>
</tr>
<tr>
<td>Amount</td>
<td>double</td>
</tr>
<tr>
<td>DueDate</td>
<td>date</td>
</tr>
<tr>
<td>PaidDate</td>
<td>date</td>
</tr>
<tr>
<td>CustomerId</td>
<td>int</td>
</tr>
<tr>
<td>CustomerPONum</td>
<td>int</td>
</tr>
<tr>
<td>IsPaid</td>
<td>boolean</td>
</tr>
<tr>
<td>Reminders</td>
<td>date</td>
</tr>
<tr>
<td>CheckId</td>
<td>int</td>
</tr>
</tbody>
</table>
hec

hec represents checks that carry money.
Assumptions: Only consider the basic information of a check

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckId</td>
<td>int</td>
</tr>
<tr>
<td>CheckNumber</td>
<td>int</td>
</tr>
<tr>
<td>Amount</td>
<td>double</td>
</tr>
<tr>
<td>CheckDate</td>
<td>date</td>
</tr>
<tr>
<td>IsCustomerCheck</td>
<td>boolean</td>
</tr>
</tbody>
</table>

Methods:

counts Paya e

ScheduleE pense
   ➔ PONumber int
   ➔ ScheduledDate date
   ➔ Amount double
   ➔ VerderId int
   ➔ EmployeeId int
   ➔ (E penseld) int

PayE penseld
   ➔ E penseld int
   ➔ CheckNumber int

IsE penseldPaid
   ➔ E penseld int
   ➔ (IsPaid) boolean

GetE pensedPONumber
   ➔ E penseld int
   ➔ (PONum) int

GetE pensedAmount
   ➔ E penseld int
   ➔ (Amount) double

GetE pensedVenderId
   ➔ E penseld int
   ➔ (VenderId) int

GetE pensedScheduledDate
   ➔ E penseld int
   ➔ (ScheduledDate) date

GetE pensedPaidDate
   ➔ E penseld int
   ➔ (PaidDate) date

GetE pensedCheck
   ➔ E penseld int
(Check)

SetExpensePONumber
  → ExpenseId int
  → PONum int
  ← void

SetExpenseAmount
  → ExpenseId int
  → Amount double
  ← void

SetExpenseVendorId
  → ExpenseId int
  → VendorId int
  ← void

SetExpenseScheduledDate
  → ExpenseId int
  → ScheduledDate date
  ← void

SetExpensePaidDate
  → ExpenseId int
  → PaidDate date
  ← Void

ReportAllExpenses
  → void
  ← (ExpenseId's) int[]

ReportExpensesOfPO
  → PONum int
  ← (ExpenseId's) int[]

ReportExpensesOfVendor
  → VendorId int
  ← (ExpenseId's) int[]

ReportExpensesOfEmployee
  → EmployeeId int
  ← (ExpenseId's) int[]

ReportAllPaidExpenses
  → void
  ← (ExpenseId's) int[]

ReportAllUnpaidExpenses
  → void
  ← (ExpenseId's) int[]

DumpExpenses
  → ExpenseIds int[]
  ← (Expenses) Expense[]
**Accounts Receivable**

Some Getters & Setters are omitted

**IssueInvoice**

- `CustomerId` int
- `CustomerPONum` int
- `DueDate` date
- `Amount` double
- `(InvoiceId)` int

**ReceivePayment**  
assuming always pay in full

- `InvoiceId` int
- `CustomerCheckNumber` int
- `CustomerCheckDate` date
- void

**RemindCustomer**

- `InvoiceId` int
- void

**IsInvoiceOverDue**

- `InvoiceId` int
- `(IsOverDue)` Boolean

**IsInvoicePaid**

- `InvoiceId` int
- `(IsPaid)` boolean

**ReportAllInvoices**

- void
- `(InvoiceId s)` int

**ReportInvoiceOfCustomerPO**

- `CustomerPONum` int
- `(InvoiceId s)` int

**ReportExpensesOfCustomer**

- `CustomerId` int
- `(InvoiceId s)` int

**ReportAllPaidInvoices**

- void
- `(InvoiceId s)` int

**ReportAllUnpaidInvoices**

- void
- `(InvoiceId s)` int

**DumpInvoices**

- `InvoiceIds` int
- `(Invoices)` Invoice

**Check Controlling**

**GetCheckInformation**

- `CheckNumber` int
- `(Check)` Check
The basic types I will model are:

**Employee**
This the main type which forms the basis of HR Management

<table>
<thead>
<tr>
<th>Field</th>
<th>XML RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee ID</td>
<td>String</td>
</tr>
<tr>
<td>firstName</td>
<td>String</td>
</tr>
<tr>
<td>middleName</td>
<td>String</td>
</tr>
<tr>
<td>lastName</td>
<td>String</td>
</tr>
<tr>
<td>dob</td>
<td>dateTime</td>
</tr>
<tr>
<td>sex</td>
<td>String</td>
</tr>
<tr>
<td>Job ID</td>
<td>int</td>
</tr>
<tr>
<td>departmentID</td>
<td>int</td>
</tr>
<tr>
<td>status/Field</td>
<td>boolean</td>
</tr>
</tbody>
</table>

**Department**
Tracks the Departments in the company

<table>
<thead>
<tr>
<th>Field</th>
<th>XML RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>departmentID</td>
<td>int</td>
</tr>
<tr>
<td>departmentName</td>
<td>String</td>
</tr>
<tr>
<td>employeeList</td>
<td>int</td>
</tr>
</tbody>
</table>

**Job Description**
Keeps a list of All the job Descriptions in the company.: PositionID is an instance of a particular job.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>positionID</td>
<td>int</td>
</tr>
<tr>
<td>departmentID</td>
<td>int</td>
</tr>
<tr>
<td>positionID</td>
<td>int</td>
</tr>
</tbody>
</table>
**Position Description**
Describes the specific position/job

<table>
<thead>
<tr>
<th>Field</th>
<th>XML RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PositionID</td>
<td>int</td>
</tr>
<tr>
<td>PositionTitle</td>
<td>String</td>
</tr>
<tr>
<td>SalaryGrade</td>
<td>int</td>
</tr>
<tr>
<td>Status</td>
<td>boolean</td>
</tr>
</tbody>
</table>

**Salary**
Keeps track of Salary Information for Each employee

<table>
<thead>
<tr>
<th>Field</th>
<th>XML RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmployeeID</td>
<td>int</td>
</tr>
<tr>
<td>SalaryGrade</td>
<td>int</td>
</tr>
<tr>
<td>SalaryAmount</td>
<td>int</td>
</tr>
<tr>
<td>BonusPlan ( yearlyAmt)</td>
<td>int</td>
</tr>
</tbody>
</table>

**Hiring/Promotions**
Keeps Hiring and Promotion Information for Each Employee

<table>
<thead>
<tr>
<th>Field</th>
<th>XML RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmployeeID</td>
<td>int</td>
</tr>
<tr>
<td>HireDate</td>
<td>dateTime</td>
</tr>
<tr>
<td>PromotionDates</td>
<td>dateTime</td>
</tr>
<tr>
<td>ReleaseDate</td>
<td>dateTime</td>
</tr>
</tbody>
</table>

**Benefits**
Keeps tracks of Benefits for Each Employee

<table>
<thead>
<tr>
<th>Field</th>
<th>XML RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmployeeID</td>
<td>int</td>
</tr>
<tr>
<td>SavingsPlan</td>
<td>String</td>
</tr>
<tr>
<td>MedicalPlan</td>
<td>String</td>
</tr>
<tr>
<td>DentalPlan</td>
<td>String</td>
</tr>
</tbody>
</table>
METHODS

AddEmployee(EmployeeID, firstName, middleName, lastName, DOB, Sex, obID, SalaryAmount, BonusPlan, HireDate, SavingsPlan, MedicalPlan, DentalPlan)-used to add employees to the system
DelEmployee(EmployeeID, ReleaseDate)-removes employee from the system: sets Status field to false
ListEmployees()-gives the list of employees

AddDepartment(DepartmentID, DepartmentName)-Add a department to the system
RemoveDepartment(DepartmentID)-removes a department
ListDepartments()-list all the departments
ListEmpDepartment(DepartmentID)-list employees working in a particular department

Add job ob( obID, PositionID, PositionTitle, Status, DepartmentID, SalaryGrade)-add a job to the system
Remove ob( obID)-remove a job from the system
List obs()-list the current active jobs
Open obs()-Lists open positions

ChangeSalary(EmployeeID, SalaryAmount, Bonus)-change the salary of a particular employee
ListSalaries()-generate a list of all employees with their salaries

AddPromotions(EmployeeID, PromoDate, NewSalary)-Assign a promotion
ListPromotions(EmployeeID)-List the Promotion dates for a particular employee
ListHireDate(EmployeeID)-List the Hire Date for a particular employee

ChangeBenefits(EmployeeID, Savings, Medical, Dental)-change the benefit plan
ListBenefits(EmployeeID)-list benefits for a particular employee
or populating jobs and employees to the system I will first generate a list of jobs and assign these jobs to a list of Employees.
I plan to use flat files to store my data structures
After there are sufficient employees in the system, the program will randomly call one of the methods to
i) change employee information: change job description, change salary information, change benefits information
ii) change job information: either to add new jobs, remove jobs from the system

In order to keep payroll salary information for each employee I will need to interact with the Payroll webservice, to keep Department information I would need to track changes such as creation deletion of departments in the company
Web service descriptions

Table of contents

Overview Diagram

<table>
<thead>
<tr>
<th>Service</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purchasing</td>
<td>Stan</td>
<td>A-1</td>
</tr>
<tr>
<td>Accounting</td>
<td>Jiangpen</td>
<td>C-1</td>
</tr>
<tr>
<td>Human Resources</td>
<td>Amit</td>
<td>H-1</td>
</tr>
<tr>
<td>Inventory</td>
<td>Yufei</td>
<td>I-1</td>
</tr>
<tr>
<td>Sales</td>
<td>Mike W.</td>
<td>S-1</td>
</tr>
<tr>
<td>Project Management</td>
<td>Vijay</td>
<td>P-1</td>
</tr>
<tr>
<td>Manufacturing</td>
<td>Scott</td>
<td>M-1</td>
</tr>
<tr>
<td>Deployment</td>
<td>Mike P.</td>
<td>D-1</td>
</tr>
<tr>
<td>Asset management</td>
<td>Rich P.</td>
<td>F-1</td>
</tr>
<tr>
<td>Trouble Ticketing</td>
<td>John</td>
<td>T-1</td>
</tr>
</tbody>
</table>
Interaction with other services:

Data Types:

### ProductType

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PartNum</td>
<td>string</td>
</tr>
<tr>
<td>Name</td>
<td>string</td>
</tr>
<tr>
<td>Description</td>
<td>string</td>
</tr>
</tbody>
</table>

**Field specification:**
- PartNum: a unique identification number standing for the product type
- Name: a string giving the human-readable name for the type
- Description: a string describing the product-specific properties

### ProductItem

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SerialNum</td>
<td>int</td>
</tr>
<tr>
<td>BarCode</td>
<td>string</td>
</tr>
</tbody>
</table>
PartNum string
LotID int
LocationID int

Field specification:
- SerialNum: a unique serial number for a product item.
- BarCode: a unique bar code for a product item
- PartNum: the ID of the type of the product item
- LocationID: the id of the location where item is stored currently. -1 means that item is on the way.

ItemHistoryEntry
Field XML-RPC Type
SerialNum int
Action string
LocationID int
Time dateTime

Field specification:
- SerialNum: the serial number of item.
- Action: the defined action is “REMOVE” and “ADD”.
- LocationID: the id of location involved.
- Time: the time when action is taken

Lot
Field XML-RPC Type
LotID int
ItemArray array of string

Field specification:
- LotID: the id of lot, which is a group of items.
- ItemArray: the serial numbers of items in the lot

Location
Field XML-RPC
LocationID int
Site int
Building string
Floor string
Room string

Field specification
I just keep the same definition of location used in Asset Service.
Methods:

BarCoding

Boolean assignBarCode(string SerialNum, string barCode)
string queryBarCode(string SerialNum)
int queryItemSN(string barcode)

Warehouse Management

Boolean removeItem(string SerialNum, dateTime time)
Boolean storeItem(string SerialNum, int locationID, dateTime time)
Boolean transferToDeployment(string SerialNum, dateTime time)
int queryLocation(string SerialNum)
ItemHistory[] queryItemHistory(string SerialNum)

Lots tracking

int queryLot(string SerialNum)
Boolean assignLot(string SerialNum, Lot lot)
int[] getItemsInLot(Lot lot)

Type tracking

string queryProductType(string SerialNum)

Quantity tracking

int queryQuantity(string PartNum)

Indirection query

ProductType getProductType(string PartNum)
ProductItem getProductItem(string SerialNum)
Lot getLot(int lotID)
Location getLocation(int locationID)

Junk Query

int[] getAllItems()
int[] getAllTypes()
int[] getAllLots()
int[] getLocations()

Simulated load generation:

I will generate about 100 locations, 100 production types, 1,000 - 10,000 items for each type. Also, I will execute about 2 random movements (from one location to another location) for each item.
My project is to develop a Sales Management package for Fubar, Inc. Here are the data types and methods I propose.

**DATA TYPES**

**Product**
The system needs to know about Fubar’s products so that sales reps can enter orders without having to fill in all the details. The product information maintained here will likely be different from that maintained by the Inventory and Manufacturing services.

- **PartNumber** string
- **ProductLineID** int
- **Description** string -- the product name
- **BasePrice** double
- **DiscountCodes** array of strings
- **DiscountRates** array of doubles
- **SubstitutePartNumber** string -- what to substitute if this item is out of stock
- **IsActive** boolean -- set false for discontinued items

**Customer**
We must store the customer’s address for shipping and returns purposes. The status value tells us if a contract job is ongoing, if the customer is no longer valid, etc.

- **CustomerID** int
- **CustomerName** string
- **CustomerPhone** string
- **BillingStreetAddress** string
- **BillingTown** string
- **BillingState** string
- **BillingZip** int
- **ShippingStreetAddress** string
- **ShippingTown** string
- **ShippingState** string
- **ShippingZip** int
- **AccountBalance** double -- sales or accounts receivable
- **Status** string

**Sale**
The Sale object comprises the information a sales rep needs in order to fill out an invoice and complete a sale.

- **ReferenceNo** int
- **CustomerID** int
- **SalespersonID** string
- **DateAndTime** dateTime.iso
- **LineItems** array of SalesLineItem objects
- **SalesTax** double
ShippingCharge    double
DeliveryTax    double    -- related to the customer's location
Total          double
AmountPaid     double
Status         Boolean    -- shipped yet (Y/N)
TrackingNo     int

SalesLineItem
BillingRate    double    applies to contract jobs, in which case Quantity will be the hours billed.
PartNumber     string
SerialNumbers  array of string
BillingRate    double
HoursBilled    double
DiscountCode   string
SoftwareKey    string

Return
Keeps a record of all merchandise returns. Fubar's decision to accept or reject the return (based on condition, etc.) and the amount credited to the customer's account.
ReferenceNo    int    -- local key
SalesReferenceNo int    -- foreign key into the Sales database
PartNumber     string
QuantityReturned array of SerialNums
DateReturned   dateTime.iso8601
IsAccepted     array of Boolean
AmountCredited array of double
Comments       string
ReplacementReferenceNo string

METHODS
Sales order processing
CreateOrder (customerID, salesperson) – sets up a new order invoice for the sales rep.
AddItemToOrder (SalesLineItem) – adds a line item to an invoice. Fills in product
description, software key (if applicable)
RemoveFromOrder (productID, quantity) – deletes a line item from an invoice
PostOrder () – Creates and returns a ReferenceNo for this order. Posts the order invoice
so that the warehouse can fulfill it and it can be shipped.
DisplayOrder (ReferenceNo) – Displays a simple list of the sales line items, tax, total,
and customer name ID
CreateLineItem (productID, quantity) – initializes a new SalesLineItem object
GetSetItemQuantity(SalesLineItem)
GetSetSoftwareKey(SalesLineItem)
SetDiscountCode(SalesLineItem)
CalculateTax (Sale object), CalculateTotal(Sale object)
DisplayBalance (CustomerID) – returns the balance on an account
CreateProduct (description, price, discount rate) – adds a new product to the database
creates and returns a product ID
Get Set product attribute (productID, attributeValue) – for those attributes that should be readable writable by a salesperson
DiscontinueProduct (productID)
DisplayProduct (productID) – Displays the product information maintained in the sales database
ProductName ID(productID), ProductID Name(description)
CreateCustomer (Name, Shipping.Billing Address, Town, State, ip ) – adds a new customer to the database
Get Set customer attribute (CustomerID, attributeValue) – for those attributes that should be readable writable by a salesperson
DeactivateCustomer (CustomerID) – marks a customer as no longer valid active
DisplayCustomerHistory (CustomerID, Date) – shows the purchases on record for this customer since da e
DisplayCustomer(CustomerID) – Displays the customer information maintained in the sales database
CustomerName ID(), CustomerID Name()
ApplyCharge(CustomerID, Amount) – returns the account balance after the charge
ApplyCredit(CustomerID, Amount) – returns the account balance after the credit

Quotations
GetPriceOrRate (ProductID, discountCode, bool includeTax ) – returns the full or discounted price of an item, or the rate for contract work

Invoices and orders display different collections of sale information in different ways
DisplayInvoice (ReferenceNo) – displays the full invoice pertaining to a specific order

Shipping
GrabItem (productID, quantity) – obtains the item needed for shipping and decrements the inventory obtains substitute items if necessary.
SetShippingCost (ReferenceNo) – establishes the cost of shipping the order
GetShippingCost (ReferenceNo) – returns the shipping cost for the order
PostShipment (ReferenceNo) – informs the sales database that a shipment has been performed by recording and returning a tracking number

Returns
AddReturn (SalesReferenceNo, Comments) – adds a return record to the database
AcceptReturn (bool IsAccepted, double AmountCredited) – marks return as accepted rejected, credits the customer s account
RunningTotalReturns(Date) – returns the total value of all returns accepted since da e

SIMULATED LOAD
A temporary database of customers and products will be set up by a client program that calls the Create() methods several times over. Then the actual testing will occur as the client simulates the real-world activity of taking and fulfilling orders, making shipments, and accepting returns and synchronizing these transactions with the Inventory Service. The client will invoke some logical sequence of these operations, and before terminating it will call DisplayCustomerHistory() on each customer RunningTotalSales() and RunningTotalReturns() for purposes of auditing and verification. I will also obtain statistics from Inventory Management to ensure Sales and Inventory are properly synchronized.
Based on my understanding of the topic and the ways to interpret it, I chose to break it down into 2 broad components –

1) Project Management is often used to track the progress of and manage the resources (i.e. people, equipment, subcontractors, etc.) used in complex projects.
2) From a customer’s perspective, it involves preparing bills (including pricing) and tracking orders.

This web service may interact with Asset management (which may place asset requests), Purchasing (where the asset requests will be checked and redirected), and HR (place staffing requests)

These are the various types I feel the need for, as I make an initial design of the system:

**Asset Requirements**
This type would get information from Asset Management when the need is felt for asset purchases and the information would be passed on to the Purchases Department

- asset_req_id int
- asset_type string
- quantity int
- requirements_text string
- date_required_by datetime

**Department**
Various departments in the company that can place requests for staff

- department_id string
- department_name string
- requirement_text string
- requirement_num int

**Staffing Requirements**
This type would get information regarding staffing requirements from any department and the matter would be referred to HR.

- staff_req_id int
- department_id string
- requirements_text string
- number_required int
- date_required_by datetime

**Bill**
Get order information and generate a bill for customer.
CustomerID    string
Purchase Order Number    int

All classes/tables created by Stan Rajan for Purchasing will be crucial to my implementation. Maybe, the `return products` implementation is better suited in my project – since Project Management deals with customer interaction and status checking.

//in Customer table/entity, “Boolean payment_received” to be included for the purposes of my project.

**Methods:**

submit_asset_req(type, number, date, notes)
send_asset_req(asset_req_id)
remove_staffing_req(asset_req_id)
remove_staffing_req(staff_req_id)
send_staffing_req(staff_req_id)
create_bill(cust_id, order_id)
update_status(order_id, status_text, shipping_date, delivery_date)
create_return(order_id, return_reason, date)
track_return(return_id)
credit_payment(return_id)

**Simulated Load**

A program would create a simulated system with a bunch of customers, orders, and departments. Then, the asset management department will place various asset requests, various other departments will place many staffing requests, the status of various orders would be updated, bills generated, customer tracking requests placed, and the ability of the web service to handle such multiple simultaneous requests correctly will be tested.
TYPES:

Bill of Material

This type would represent the product and the parts that make up the product in the manufacturing process.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>int</td>
</tr>
<tr>
<td>Product_Name</td>
<td>String</td>
</tr>
<tr>
<td>PartNum</td>
<td>String</td>
</tr>
<tr>
<td>Parts</td>
<td>Array of ints</td>
</tr>
</tbody>
</table>

Engineering Change Notice

This would represent an Engineering Change Notice request.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>int</td>
</tr>
<tr>
<td>Product_Name</td>
<td>String</td>
</tr>
<tr>
<td>PartNum</td>
<td>String</td>
</tr>
<tr>
<td>partChanges</td>
<td>array of structs of ids</td>
</tr>
</tbody>
</table>

GenericPart

This represents the common part needed (i.e. screw but not say a screw by Home Depot).

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>int</td>
</tr>
<tr>
<td>Amount</td>
<td>int</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
</tr>
</tbody>
</table>

Vendor

This represents the manufacturer of the specific part.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
</tr>
<tr>
<td>vendorID</td>
<td>int</td>
</tr>
<tr>
<td>Address</td>
<td>String</td>
</tr>
<tr>
<td>State</td>
<td>String</td>
</tr>
<tr>
<td>Zip Code</td>
<td>String</td>
</tr>
</tbody>
</table>
City
Contact Name
Contact Number

**Product**

This represents the product type (i.e. Temperature Sensor).

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>int</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
</tr>
</tbody>
</table>

**ProductInstance**

This represents a specific product instance, i.e. a specific Temperature Sensor.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parts</td>
<td>array of vendorPartNum</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>String</td>
</tr>
<tr>
<td>ManufacturedDate</td>
<td>dateTime.iso8601</td>
</tr>
<tr>
<td>Product_Name</td>
<td>String</td>
</tr>
<tr>
<td>Part_Number</td>
<td>String</td>
</tr>
</tbody>
</table>

**SpecificPart**

This represents a specific part type made by a manufacturer.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>int</td>
</tr>
<tr>
<td>Vendor_ID</td>
<td>int</td>
</tr>
<tr>
<td>VendorPartNum</td>
<td>String</td>
</tr>
<tr>
<td>Cost</td>
<td>double</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
</tr>
<tr>
<td>Amount</td>
<td>int</td>
</tr>
</tbody>
</table>
public interface InventoryManager
{
    public Collection getOutOfStockParts();
    public Collection getLowStockParts();
    public Collection getPartsWithStockLessThan(int amount);
    public Collection getPartsWithStockGreaterThan(int amount);
    public Collection getSpecificParts(int id);
    public int incrementPartAmount(String partId, int amount)
        throws MaximumAmountException;
    public int decrementPartAmount(String partId, int amount)
        throws MinimumAmountException;
    public GenericPart addPartType(int part_id);
    public GenericPart addSpecificPart(String partId, int mid, double cost, String name, int count);
    public SpecificPart updateSpecificPartCost(String partId);
    public boolean removeSpecificPart(String partId);
    public boolean removeGenericPart(int id);
    public Collection getManufacturers();
    public Manufacturer ManufacturerById(int id);
    public Collection getPartsByManufacturer(int mid);
    public double getAveragePartPrice(int pid);
    public double[] getPartPriceHistory(String partId);
    public double getAverageProductCost(int bom);
}

NOTE: The InventoryManager will be required to interface with the Purchasing Service

public interface ManufacturingManager
{
    public ProductInstance buildSensor(String productID)
        throws NotEnoughMaterialsException;
    public Collection buildSensors(String productID, int amount)
        throws NotEnoughMaterialsException;
    public Collection buildSensorsToStock(int id);
    public ProductInstance increaseStage(String serialNumber)
        throws FinishedProductException;
    public Collection increaseStage(Collection collection);
    public boolean updateBillOfMaterialsBasedOnECN(int nid);
}

NOTE: The ManufacturingManager will most likely need to talk with the Sales to determine how much product to make.

public interface MaterialRequirementsPlanningManager
{
    public Collection getListOfRecommendedPartsToReOrder();
    public void addProductToMasterProductionSchedule(Product product, int quantity, Date estimatedStartDate);
}
public void removeProductFromMasterProductionSchedule(Product product, Date estimatedStartDate);

Note: There will also be other methods to do things such as addBillOfMaterial, etc. that will essentially be the data entry into the database.

**Simulated Load**

Data will be entered in to the database. This will either be done using a script to enter directly into the database, or via the methods provided. A program will then be created that will simulate building up an inventory of parts and then continually building sensors and updating parts via simulation of both build-to-stock and build-to-order. This should simulate the day-to-day build-up and use of parts as well as the creation of new products. At the end the reporting methods can be used to see what was created and used.
A deployment web service, as researched, is best be described as unifying system of post inventory management and support services. The methods described below should be adequate to allow for the support service to get information on individual, and the overall states of the deployed products. As such this service should be fed by the sales and trouble ticket webservice in order to keep the information up to date.

**Objects:**

### Product

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Name</td>
<td>String</td>
</tr>
<tr>
<td>Part Number</td>
<td>String</td>
</tr>
<tr>
<td>Serial Number</td>
<td>String</td>
</tr>
<tr>
<td>Customer ID</td>
<td>Int</td>
</tr>
<tr>
<td>Software Version</td>
<td>String</td>
</tr>
<tr>
<td>Operational Status</td>
<td>boolean</td>
</tr>
<tr>
<td>Recall Notice</td>
<td>boolean</td>
</tr>
<tr>
<td>Manufacture Date</td>
<td>dateTime</td>
</tr>
<tr>
<td>End of Maintenence</td>
<td>dateTime</td>
</tr>
<tr>
<td>End of Life</td>
<td>dateTime</td>
</tr>
</tbody>
</table>

### Customer

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer ID</td>
<td>Int</td>
</tr>
<tr>
<td>Customer Name</td>
<td>String</td>
</tr>
<tr>
<td>Street Address</td>
<td>String</td>
</tr>
<tr>
<td>City</td>
<td>String</td>
</tr>
<tr>
<td>State</td>
<td>String</td>
</tr>
<tr>
<td>Postal Code</td>
<td>String</td>
</tr>
<tr>
<td>Country</td>
<td>String</td>
</tr>
<tr>
<td>Contact Name</td>
<td>String</td>
</tr>
<tr>
<td>Contact Email</td>
<td>String</td>
</tr>
<tr>
<td>Contact Phone Number</td>
<td>String</td>
</tr>
</tbody>
</table>

### Software

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Name</td>
<td>String</td>
</tr>
<tr>
<td>Customer ID</td>
<td>Int</td>
</tr>
<tr>
<td>Software Version</td>
<td>String</td>
</tr>
<tr>
<td>Update Available</td>
<td>Boolean</td>
</tr>
<tr>
<td>Recall Notice</td>
<td>Boolean</td>
</tr>
<tr>
<td>End of Maintenence</td>
<td>dateTime</td>
</tr>
<tr>
<td>End of Life</td>
<td>dateTime</td>
</tr>
</tbody>
</table>
Methods:

**Product**
addProduct(Product, Customer, dateTime)
removeProduct(serialNumber)
updateProductStatus(serialNumber, operationalStatus)
getProductStatus(serialNumber)

**Software**
addSoftware(Software, Customer, dateTime)
removeSoftware(productName, customerId)
updateSoftwareAvailable(productName)
getSoftwareAvailable(productName)
  * Calls support services to check if an update is available
updateSoftwareVersion(productName, softwareVersion);

**Shared**
getRecall(productName);
  * Calls support services to check for a recall
updateRecall(String productName, Boolean status)
  - Announce/cancel recall of specified product
getEndOfMantenence(productName, customerId);
updateMantence(productName, CustomerID)
getEndOfLife(String productName);
  * Calls support service to check for EndofLife
updateEndOfLife(productName, dateTime)

**Informative Queries**
getTotalDeployed(productName, dateTime, dateTime)
  - Returns the total amount of product deployed between a
given date
getTotalCustomers(customerID, productName);
  - Returns the total amount of customers with the specified
  product
getTotalFailedProducts(productName);
getFailedProducts(productName, dateTime, dateTime)
  - Returns product(s) that failed during the time period
getCustomersByProducts(productName);
  - Returns all products a customer has
getProductsByCustomer(customerID);
  - Returns all customers that have a product

Simulated Load:

A basic main program will randomly create sales and service events then update
the system appropriately while also dumping it's output to a log file for verification upon
completion. After a given amount of time the main program will then ask for statics
from the service which can be checked against the output file to ensure proper
execution.
For my job, one of my projects is the deployment of an enterprise asset management system for my division. Based on my familiarity with the system and the data stored in the system, I put together the following types and methods.

**TYPES:**

**Employee**

The asset tracking module would need to track some basic information on employees since it is necessary to know who owns the equipment.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee ID</td>
<td>- string</td>
</tr>
<tr>
<td>First Name</td>
<td>- string</td>
</tr>
<tr>
<td>Last Name</td>
<td>- string</td>
</tr>
<tr>
<td>Department Number</td>
<td>- int</td>
</tr>
</tbody>
</table>

**Asset**

The asset tracking module would need to track detailed information on all of Fubar’s assets. This includes information related to the purchase, installation, and disposal of the assets.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset ID</td>
<td>- int</td>
</tr>
<tr>
<td>Asset Description</td>
<td>- string</td>
</tr>
<tr>
<td>Asset Classification</td>
<td>- string</td>
</tr>
<tr>
<td>Model #</td>
<td>- string</td>
</tr>
<tr>
<td>Serial #</td>
<td>- string</td>
</tr>
<tr>
<td>Owner – Employee ID</td>
<td>- int</td>
</tr>
<tr>
<td>Department Number</td>
<td>- int</td>
</tr>
<tr>
<td>Purchase Date</td>
<td>- dateTime</td>
</tr>
<tr>
<td>Purchase Price</td>
<td>- double</td>
</tr>
<tr>
<td>Installation Date</td>
<td>- dateTime</td>
</tr>
<tr>
<td>Location ID</td>
<td>- int</td>
</tr>
<tr>
<td>Vendor ID</td>
<td>- int</td>
</tr>
<tr>
<td>Disposal Date</td>
<td>- dateTime</td>
</tr>
<tr>
<td>Status</td>
<td>- string</td>
</tr>
</tbody>
</table>

**Vendor**

The asset tracking module would need to track some basic information on vendors since it is necessary to know who sold the equipment to Fubar.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor ID</td>
<td>- int</td>
</tr>
<tr>
<td>Vendor Name</td>
<td>- string</td>
</tr>
<tr>
<td>Vendor Street Address</td>
<td>- string</td>
</tr>
<tr>
<td>Vendor Town</td>
<td>- string</td>
</tr>
<tr>
<td>Vendor State</td>
<td>- string</td>
</tr>
<tr>
<td>Vendor Country</td>
<td>- string</td>
</tr>
<tr>
<td>Vendor Contact</td>
<td>- string</td>
</tr>
<tr>
<td>Vendor Contact Phone #</td>
<td>- string</td>
</tr>
</tbody>
</table>
**Location**

The asset tracking module would need to track the exact location of the equipment. In order to achieve this level of detail, the location type would be required.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location ID</td>
<td>- int</td>
</tr>
<tr>
<td>Site</td>
<td>- string</td>
</tr>
<tr>
<td>Building</td>
<td>- string</td>
</tr>
<tr>
<td>Floor</td>
<td>- string</td>
</tr>
<tr>
<td>Room</td>
<td>- string</td>
</tr>
</tbody>
</table>

**Department**

The asset tracking module would need to track some basic information on the departments within the company.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Department Number</td>
<td>- int</td>
</tr>
<tr>
<td>Department Name</td>
<td>- string</td>
</tr>
</tbody>
</table>

**Depreciation**

The asset tracking module would need to track important values to be used in the depreciation calculations.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML-RPC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset Classification</td>
<td>- string</td>
</tr>
<tr>
<td>Depreciation Percentage per year</td>
<td>- double</td>
</tr>
<tr>
<td>Expected lifetime in years</td>
<td>- int</td>
</tr>
</tbody>
</table>
Methods

**Detailed Tracking**

AddEmployee(FirstName, LastName, EmployeeID, DepartmentNumber) – Adds a new employee to the employee table.

RemoveEmployee(EmployeeID) – Removes an employee from the employee table.

GetAllEmployees() – Returns an iterator to access all employees.

GetAllAssets() – Returns an iterator to access all assets.

AddVendor(Name, StreetAddress, Town, State, Country, Contact, ContactPhone) – Adds a new vendor to the vendor table.

RemoveVendor(VendorID) – Removes the vendor from the vendor table.

GetAllVendors() – Returns an iterator to access all vendors.

AddLocation(Site, Building, Floor, Room) – Adds a new location to the location table.

RemoveLocation(LocationID) – Removes the location from the location table.

GetAllLocations() – Returns an iterator to access all locations.

AddDepartment(DepartmentNumber, DepartmentName) – Adds a new department to the department table.

RemoveDepartment(DepartmentNumber) – Removes a department from the department table.

GetAllDepartments() – Returns an iterator to access all departments.

AddDepreciationValue(Class, Percentage, Lifetime) – Adds a new depreciation value to the table.

RemoveDepreciationValue(Class, Percentage, Lifetime) – Removes the depreciation value from the table.

GetAllDepreciationValues() – Returns an iterator to access all depreciation values.

CreateAsset(Description, Classification, EmployeeID, PurchasePrice, PurchaseDate, Department, LocationID, VendorID) – Create a new asset record with the required fields provided.

GetAssetDescription(AssetID) – returns a description of the asset with the given asset ID

SetAssetDescription(AssetID, Description) – sets the description of the asset

GetAssetClassification(AssetID) – returns the classification of the asset with the given asset ID

SetAssetClassification(AssetID, Classification) – sets the classification of the asset

GetModelNumber(AssetID) – returns the model number of the asset with the given asset ID

SetModelNumber(AssetID, ModelNumber) – sets the model number of the asset

GetSerialNumber(AssetID) – returns the serial number of the asset with the given asset ID

SetSerialNumber(AssetID, SerialNumber) – sets the serial number of the asset

GetPurchaseDate(AssetID) – returns the purchase date of the asset

SetPurchaseDate(AssetID, PurchaseDate) – sets the purchase date of the asset

GetPurchasePrice(AssetID) – returns the purchase price of the asset

SetPurchasePrice(AssetID, PurchasePrice) – sets the purchase price of the asset

GetInstallationDate(AssetID) – returns the installation date of the asset

SetInstallationDate(AssetID, InstallationDate) – sets the installation date of the asset

GetLocationID(Site, Building, Floor, Room) – returns the location id for the entered location

GetLocation(AssetID) – returns the Site + “\-“ + Building + “\-“ + Floor + “\-“ + Room of the given asset ID

SetLocation(AssetID, LocationID) – sets the location id of the asset

GetVendorId(VendorName) – returns the vendor id for the given vendor name

GetVendor(AssetID) – returns the Vendor Name for the given asset

SetVendor(AssetID, VendorID) – sets the vendor id for the given asset

**Depreciation and Gain/Loss Detail**

CalculateDepreciation(AssetID) – returns the depreciated value for the given asset using the depreciation percentages in the Depreciation table.

RunningTotal() – returns the total value of assets that have not been disposed

**Ownership**

GetEmployeeID(FirstName, LastName) – returns the employee identifier for the given employee

GetOwnerName(AssetID) – returns the first and last name of the employee that owns the asset

SetOwnerName(AssetID, EmployeeID) – sets the employee id for the owner of the asset

GetDepartment(AssetID) – returns the department that owns the asset

SetDepartment(AssetID, DepartmentNumber) – sets the department that owns the asset
**Disposal**

- `GetCurrentStatus(AssetID)` – returns the current status of the asset
- `SetCurrentStatus(AssetID, Status)` – sets the current status of the asset (In Service, Broken, Disposed)
- `GetDisposalDate(AssetID)` – returns the date that the asset was disposed
- `DisposeOfAsset(AssetID, DisposalDate)` – sets the disposal date of the asset to `DisposalDate` and updates the status to disposed
Simulated Load

There will be a main client program that initially generates new assets. This will simulate the initial population of assets into the asset management system. After there are sufficient assets in the system, the program will randomly choose different get and set methods to represent daily asset operations. This will simulate the day to day queries and updates that would take place in Fubar. During this time, new assets would be sporadically added and other items would be disposed. The function calls would allow for the correct system operation to be verified. At the end of the program, the RunningTotal() function would be run to determine the total value of Fubar’s assets.
Trouble Ticketing Web Service Revised API

Types

There are three data types in the Trouble Ticketing API (TT-API): the Trouble Ticket (TT), Incident Report (IR), and Bug Report (BR).

Trouble Ticket

Attribute:       XML Type:                   Description:
Ticket ID         dateTime.iso              Time ticket was opened, primary key
Owner             string                   Owner of the faulty product
PartNumber        string                   Part # of faulty product
CloseDate         dateTime.iso            Time ticket is closed
Closer            string                   Person who closes the ticket
Status            int                      Determines status of ticket
Description       string                   Description of problem

Logical Ticket Types:

Pending Ticket: Newly created TT  Status  -
- newly created with little or no Incident Reports assigned to it

Trouble Ticket: Active TT  Status
- complete and active Ticket

Closed Ticket: Inactive TT  Status
- a Ticket for a problem that is no longer an issue
**Incident Report**

<table>
<thead>
<tr>
<th>Attribute</th>
<th><em>ML Type</em></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ticket ID</td>
<td>dateTime.iso</td>
<td>Ticket this IR is associated with</td>
</tr>
<tr>
<td>Incident ID</td>
<td>dateTime.iso</td>
<td>Time this IR was generated, primary key</td>
</tr>
<tr>
<td>Description</td>
<td>string</td>
<td>Description of the problem</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>string</td>
<td>Serial # / version of the faulty product</td>
</tr>
<tr>
<td>PartNumber</td>
<td>string</td>
<td>Part # of the faulty product</td>
</tr>
<tr>
<td>Owner</td>
<td>string</td>
<td>Owner of faulty product</td>
</tr>
<tr>
<td>Live</td>
<td>boolean</td>
<td>True if product is still afield</td>
</tr>
</tbody>
</table>

**Bug Report**

<table>
<thead>
<tr>
<th>Attribute</th>
<th><em>ML Type</em></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug ID</td>
<td>dateTime.iso</td>
<td>Time this BR was created</td>
</tr>
<tr>
<td>PartNumber</td>
<td>string</td>
<td>Part # of buggy product</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>string</td>
<td>Serial # of product bug was initially found in</td>
</tr>
<tr>
<td>Description</td>
<td>string</td>
<td>Description of bug</td>
</tr>
<tr>
<td>workaround</td>
<td>string</td>
<td>orkaround, if any, for this bug</td>
</tr>
</tbody>
</table>

**Methods**

CreateIncidentReport(Owner, ProductNumber, SerialNumber, Description)
- create a new Incident Report  Ticket ID and Incident ID are set automatically

CreateTicket(Owner, ProductNumber, Description)
- create a new Trouble Ticket  Ticket ID and Status are set automatically

OpenTicket(Ticket ID)
- opens a Pending or Closed Ticket and makes it an Active Ticket

CloseTicket(Ticket ID, Closer)
- closes an Active Ticket and makes it an Inactive Ticket

DeleteTicket(Ticket ID)
- removes an Inactive Ticket from the system
DeleteIncident(Incident ID)
-removes an Incident Report from the system that either is not associated with a Ticket, or whose Ticket has been deleted

DeleteBug(Bug ID)
-removes a Bug Report from the system

ReturnProduct(SerialNumber)
-updates all Incident Reports to reflect returning of product

GetAllLiveIncidents()
-returns the Incident IDs of all Incident Reports for products in the field

GetAllDeadIncidents()
-returns the Incident IDs of all Incident Reports for products recalled or returned

IsLive(IncidentID)
-returns true if the product is still in the field, false if not

CreateBugReport(PartNumber, SerialNumber, Description, Workaround)
-creates a new Bug Report  Bug ID is set automatically

MakeBugReport(Incident ID, Workaround)
-makes a Bug Report out of the Incident Report specified

SetIncidentTicketID(Incident ID)
-set the Ticket ID that an Incident Report belongs to

GetIncidentTicketID(Incident ID)
-returns the Ticket ID that an Incident Report belongs to

GetAll()
-returns all Tickets and Reports

GetAll(Ticket ID, Ticket ID)
-returns an array of all Tickets for a date range

GetTicket(Ticket ID)
-returns Ticket

GetAllBugs()
-returns Tickets IDs of all Bug Reports

GetAllIncidentReports()
-returns all Incident Reports  Ticket IDs
GetAllActiveTickets()
-returns all Active Tickets' Ticket IDs
GetAllInactiveTickets()
-returns all Inactive Tickets' Ticket IDs
GetAllActiveTickets(Ticket ID, Ticket ID)
-returns all Active Tickets for a date range
GetAllInactiveTickets(Ticket ID, Ticket ID)
-returns all Inactive Tickets for a date range
GetAllBugReports(Bug ID, Bug ID)
-returns all Bug Reports for a date range
GetAllIncidentReports(Incident ID, Incident ID)
-returns all Incident Reports for a date range
GetAllIncidentReportsByOwner(Owner)
-returns the Incident IDs of all Incident Reports for a specific Owner
GetAllActiveTicketsByOwner(Owner)
-returns the Tickets IDs of all Active Tickets for a specific Owner
GetAllInactiveTicketsByOwner(Owner)
-returns the Tickets IDs of all Inactive Tickets for a specific Owner
GetAllIncidentReportsByOwner(Owner)
-returns the Incident IDs of all Incident Reports for a specific Owner
GetAllIncidentReportsByPart(PartNumber)
-returns the Tickets IDs of all Incident Reports for a specific part
GetAllActiveTicketsByPart(PartNumber)
-returns the Tickets IDs of all Active Tickets for a specific part
GetAllInactiveTicketsByReporter(PartNumber)
-returns the Tickets IDs of all Inactive Tickets for a specific part
GetAllBugReportsByPart(PartNumber)
-returns the Tickets IDs of all Bug Reports for a specific part
GetOwner(Ticket ID, Incident ID)
-returns the Owner of a Ticket or Incident
GetPart(Ticket ID   Incident ID)
-returns the PartNumber of a Ticket or Incident
GetDescription(Ticket ID)
-returns the Description of a Ticket, Incident or Bug
GetNewestIncident()
-returns the most recently logged Incident Report
GetOldestIncident()
-returns the oldest logged Incident Report

**Load**

In order to simulate accesses to the service, a client program will be written to first generate Incident Reports. As the database begins to be populated it will create less Incident Reports while executing the other web service status-changing and lookup functions more often.