

# CS 553 Spring 2004

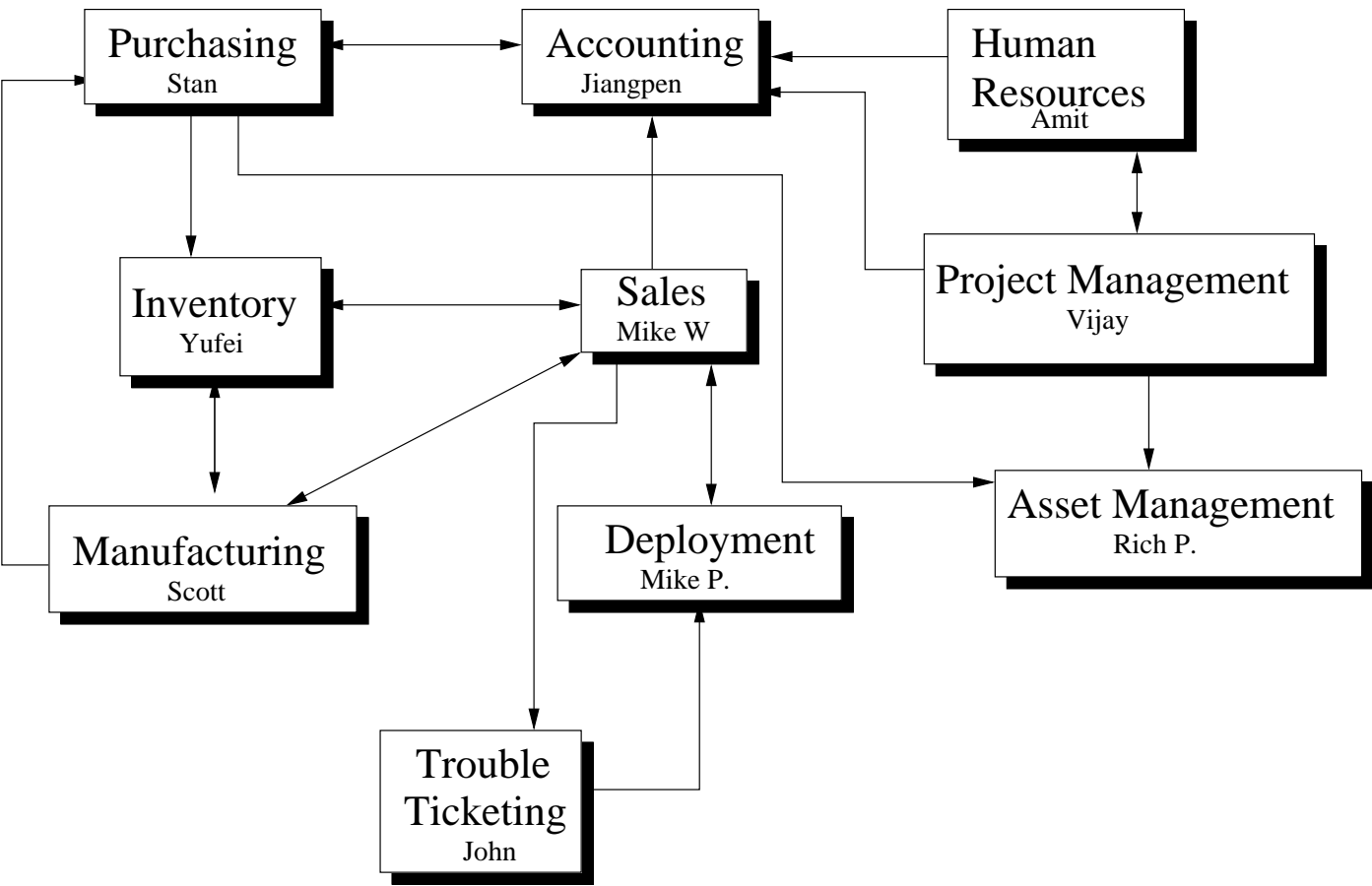
Web service descriptions

Table of contents

Overview Diagram

		1
Purchasing	Stan	A-1
Accounting	Jiangpen	C-1
Human Resources	Amit	H-1
Inventory	Yufei	I-1
Sales	Mike W.	S-1
Project Management	Vijay	P-1
Manufacturing	Scott	M-1
Deployment	Mike P.	D-1
Asset management	Rich P.	F-1
Trouble Ticketing	John	T-1

# Services Overview Diagram



Stan Rajan  
CS 553 – Internet Services

## **PURCHASING**

### **Types**

#### ***PurchaseOrder***

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

<b>Field</b>	<b>XML-RPC Type</b>
PurchaseOrderNum	- int
VendorID	- int
VendorPartNum	- string
BuyerID	- int
ShippingCode	- int
OrderStatusCode	- int
OrderDate	- dateTime
PricePerUnit	- double
QuantityOrdered	- int

#### ***OrderStatus***

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

<b>Field</b>	<b>XML-RPC Type</b>
OrderStatusCode	- int
StatusDescription	- string

#### ***ShippingMethod***

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

<b>Field</b>	<b>XML-RPC Type</b>
ShippingCode	- int
MethodDescription	- string
ArrivalLocation	- string

### ***Buyer***

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

<b>Field</b>	<b>XML-RPC Type</b>
BuyerID	- int
DepartmentID	- int
ContactName	- string
ContactPhone	- string

### ***Vendor***

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

<b>Field</b>	<b>XML-RPC Type</b>
VendorID	- int
Name	- string
StreetAddress	- string
City	- string
State	- string
ContactName	- string
ContactPhone	- string

### ***Return***

Necessary to track the status of each return request.

<b>Field</b>	<b>XML-RPC Type</b>
ReturnID	- int
PurchaseOrderNum	- int
ReturnDescription	- string
ReturnDate	- dateTime

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.

CS533 – Internet Services  
Proposal for Web Service # – Accounts Payable Accounts Receivable

**Objects:**

*pense*

*pense* represents expenses occurred during all purchases.

Assumptions:

- Amount is always paid in full (no partial payment)
- The vendor specified by *VendorId* is the payee
- *Vendor* object is stored by the **Purchasing** S
- Ignore late fee, etc.

<b>Field</b>	<b>Type</b>
<u>ExpenseId</u>	int
PONumber	int
Amount	double
VendorId	int
EmployeeId	int
ScheduledDate	date
PaidDate	date
IsPaid	Boolean
CheckId	int

*invoice*

*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 *CustomerId* is the payer
- *Customer* object is stored by the **Sales** S
- Ignore late fee, etc.

<b>Field</b>	<b>Type</b>
<u>InvoiceId</u>	int
InvoiceNumber	int
Amount	double
DueDate	date
PaidDate	date
CustomerId	int
CustomerPONum	int
IsPaid	boolean
Reminders	date
CheckId	int



**hec**

*hec* represents checks that carry money.

Assumptions: Only consider the basic information of a check

Field	Type
<u>CheckId</u>	int
CheckNumber	int
Amount	double
CheckDate	date
IsCustomerCheck	boolean

**Methods:**

*ccounts Payable*

ScheduleE pense

→ PONumber	int
→ ScheduledDate	date
→ Amount	double
→ VerderId	int
→ EmployeeId	int
← (E penseId)	int

PayE pense

→ E penseId	int
← CheckNumber	

IsE pensePaid

→ E penseId	int
← (IsPaid)	boolean

GetE pensePONumber

→ E penseId	int
← (PONum)	int

GetE penseAmount

→ E penseId	int
← (Amount)	double

GetE penseVenderId

→ E penseId	int
← (VenderId)	int

GetE penseScheduledDate

→ E penseId	int
← (ScheduledDate)	date

GetE pensePaidDate

→ E penseId	int
← (PaidDate)	date

GetE penseCheck

→ E penseId	int
-------------	-----

← (Check)	Check
SetE pensePONumber	
→ E penseId	int
→ PONum	int
← void	
SetE penseAmount	
→ E penseId	int
→ Amount	double
← void	
SetE penseVenderId	
→ E penseId	int
→ VenderId	int
← void	
SetE penseScheduledDate	
→ E penseId	int
→ ScheduledDate	date
← void	
SetE pensePaidDate	
→ E penseId	int
→ PaidDate	date
← Void	
ReportAllE penses	
→ void	
← (E penseId s)	int
ReportE pensesOfPO	
→ PONum	int
← (E penseId s)	int
ReportE pensesOfVender	
→ VenderId	int
← (E penseId s)	int
ReportE pensesOfEmployee	
→ EmployeeId	int
← (E penseId s)	int
ReportAllPaidE penses	
→ void	
← (E penseId s)	int
ReportAllUnpaidE penses	
→ void	
← (E penseId s)	int
DumpE penses	
→ E penseIds	int
← (E penses)	E pense

### *ccounts 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  
← (Invoicess)                    Invoice

### *Check Controlling*

#### GetCheckInformation

→ CheckNumber                    int  
← (Check)                         Check



Amit Gaur  
CS 553-Internet Services  
Professor Martin

Process eb Service – Human Resource Management

The basic types I will model are:

### **mployee**

This the main type which forms the basis of HR Management

<b>Field</b>	<b>XML RPC Type</b>
Employee ID	String
irstName	String
MiddleName	String
LastName	String
DOB	dateTime
Se	String
obID	int
DepartmentID	int
Status ield	boolean

### **epa tment**

Tracks the Departments in the company

<b>Field</b>	<b>XML RPC Type</b>
DepartmentID	int
DepartmentName	String
EmployeeList	int

### **Jo escription**

Keeps a list of All the ob Descriptions in the company.:PositionID is an instance of a particular ob

<b>Field</b>	<b>XML RPC Type</b>
obID	int
DepartmentID	int
PositionID	int

**Position Description**

Describes the specific position job

<b>Field</b>	<b>XML RPC Type</b>
PositionID	int
PositionTitle	String
SalaryGrade	int
Status field	boolean

**Salary**

Keeps track of Salary Information for Each employee

<b>Field</b>	<b>XML RPC Type</b>
EmployeeID	int
SalaryGrade	int
SalaryAmount	int
BonusPlan(earlyAmt)	int

**Hiring Promotions**

Keeps Hiring and Promotion Information for Each Employee

<b>Field</b>	<b>XML RPC Type</b>
EmployeeID	int
HireDate	dateTime
PromotionDates	dateTime
ReleaseDate	dateTime

**Benefits**

Keeps tracks of Benefits for Each Employee

<b>Field</b>	<b>XML RPC Type</b>
EmployeeID	int
SavingsPlan	String
MedicalPlan	String
DentalPlan	String

## METHODS

AddEmployee(EmployeeID, FirstName, MiddleName, LastName, DOB, Sex, JobID, 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(JobID, PositionID, PositionTitle, Status, DepartmentID, SalaryGrade)-add a job to the system

Remove Job(JobID)-remove a job from the system

List Jobs()-list the current active jobs

Open Jobs()-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

## **SIMULATI N**

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



# CS 553 Spring 2004

Web service descriptions

Table of contents

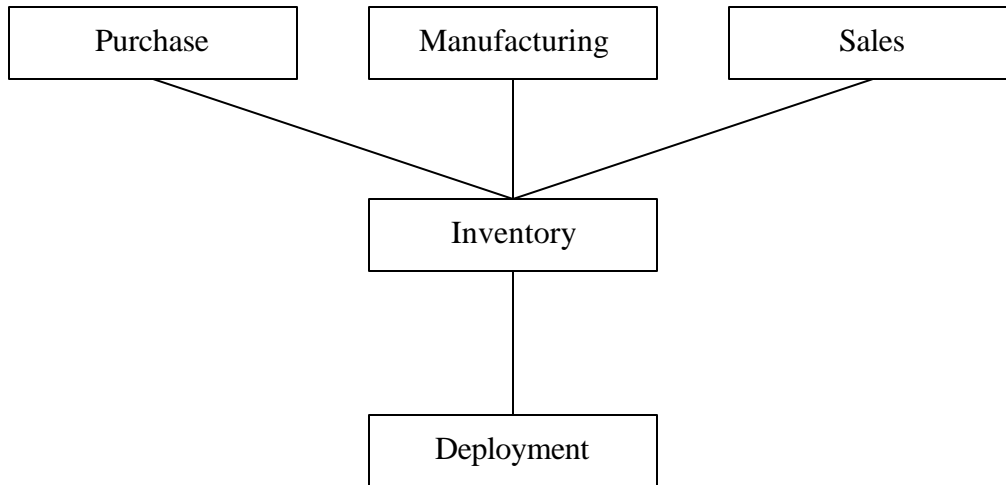
Overview Diagram

		1
Purchasing	Stan	A-1
Accounting	Jiangpen	C-1
Human Resources	Amit	H-1
Inventory	Yufei	I-1
Sales	Mike W.	S-1
Project Management	Vijay	P-1
Manufacturing	Scott	M-1
Deployment	Mike P.	D-1
Asset management	Rich P.	F-1
Trouble Ticketing	John	T-1

# CS553: Inventory service design

Yufei Pan

## Interaction with other services:



## Data Types:

### ProductType

Field	XML-RPC Type
PartNum	string
Name	string
Description	string

### 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

Field	XML-RPC Type
SerialNum	int
BarCode	string

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.

Michael Wood  
CS 553 Web Service Proposal

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	string
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.iso8601
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* 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  
Get/SetItemQuantity(SalesLineItem)  
Get/SetSoftwareKey(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  
ProductName2ID(productID), ProductID2Name(description)  
CreateCustomer (Name, {Shipping,Billing}{Address, Town, State, Zip}) – 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 *date*  
DisplayCustomer(CustomerID) – Displays the customer information maintained in the sales database  
CustomerName2ID(), CustomerID2Name()  
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

### Invoicing

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

*Vijay Lakshminarayanan*  
*mailvj@paul*

**CS 553 – Internet Services**  
**Dr. Richard Martin**

**Project – Stage I Description**  
**Topic: Project Management**

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)  
submit\_staffing\_req(dept, 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.

<b>Field</b>	<b>XML-RPC Type</b>
Id	int
Product_Name	String
PartNum	String
Parts	Array of ints

***Engineering Change Notice***

This would represent an Engineering Change Notice request.

<b>Field</b>	<b>XML-RPC Type</b>
Id	int
Product_Name	String
PartNum	String
partChanges	array of structs of ids

***GenericPart***

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

<b>Field</b>	<b>XML-RPC Type</b>
Id	int
Amount	int
Name	String

***Vendor***

This represents the manufacturer of the specific part.

<b>Field</b>	<b>XML-RPC Type</b>
Name	String
vendorID	int
Address	String
State	String
Zip Code	String

City	String
Contact Name	String
Contact Number	String

### ***Product***

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

<b>Field</b>	<b>XML-RPC Type</b>
Id	int
Name	String

### ***ProductInstance***

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

<b>Field</b>	<b>XML-RPC Type</b>
Parts	array of vendorPartNum
SerialNumber	String
ManufacturedDate	dateTime.iso8601
Product_Name	String
Part_Number	String

### ***SpecificPart***

This represents a specific part type made by a manufacturer.

<b>Field</b>	<b>XML-RPC Type</b>
Id	int
Vendor_ID	int
VendorPartNum	String
Cost	double
Name	String
Amount	int

```

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, double cost);
    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 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 webservices in order to keep the information up to date.

## **Objects:**

### ***Product***

<b>Field</b>	<b>XML-RPC Type</b>
Product Name	- String
Part Number	- String
Serial Number	- String
Customer ID	- Int
Software Version	- String
Operational Status	- boolean
Recall Notice	- boolean
Manufacture Date	- dateTime
End of Maintenance	- dateTime
End of Life	- dateTime

### ***Customer***

<b>Field</b>	<b>XML-RPC Type</b>
Customer ID	- Int
Customer Name	- String
Street Address	- String
City	- String
State	- String
Postal Code	- String
Country	- String
Contact Name	- String
Contact Email	- String
Contact Phone Number	- String

### ***Software***

<b>Field</b>	<b>XML-RPC Type</b>
Product Name	- String
Customer ID	- Int
Software Version	- String
Update Available	- Boolean
Recall Notice	- Boolean
End of Maintenance	- dateTime
End of Life	- dateTime

## 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  
getEndOfMaintenance(productName, customerID);  
updateMaintenance(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 its output to a log file for verification upon completion. After a given amount of time the main program will then ask for statistics 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.

<b>Field</b>	<b>XML-RPC Type</b>
Employee ID	- string
First Name	- string
Last Name	- string
Department Number	- int

### *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.

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

### *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.

<b>Field</b>	<b>XML-RPC Type</b>
Vendor ID	- int
Vendor Name	- string
Vendor Street Address	- string
Vendor Town	- string
Vendor State	- string
Vendor Country	- string
Vendor Contact	- string
Vendor Contact Phone #	- string



### ***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.

<b>Field</b>	<b>XML-RPC Type</b>
Location ID	- int
Site	- string
Building	- string
Floor	- string
Room	- string

### ***Department***

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

<b>Field</b>	<b>XML-RPC Type</b>
Department Number	- int
Department Name	- string

### ***Depreciation***

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

<b>Field</b>	<b>XML-RPC Type</b>
Asset Classification	- string
Depreciation Percentage per year	- double
Expected lifetime in years	- int

## 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.

John Francisco  
Internet Services  
Rich Martin  
19 February, 2004

## 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:

<u>Attribute:</u>	<u>XML Type:</u>	<u>Description:</u>
Ticket ID	dateTime.iso8601	Time ticket was opened, primary key
Owner	string	Owner of the faulty product
PartNumber	string	Part # of faulty product
CloseDate	dateTime.iso8601	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 = -1  
- newly created with little or no Incident Reports assigned to it

Trouble Ticket:      Active TT      Status = 0  
- complete and active Ticket

Closed Ticket:      Inactive TT      Status = 1  
- a Ticket for a problem that is no longer an issue

## **Incident Report:**

<u>Attribute:</u>	<u>XML Type:</u>	<u>Description:</u>
Ticket ID	dateTime.iso8601	Ticket this IR is associated with
Incident ID	dateTime.iso8601	Time this IR was generated, primary key
Description	string	Description of the problem
SerialNumber	string	Serial # / version of the faulty product
PartNumber	string	Part # of the faulty product
Owner	string	Owner of faulty product

## **Bug Report:**

<u>Attribute:</u>	<u>XML Type:</u>	<u>Description:</u>
Bug ID	dateTime.iso8601	Time this BR was created
PartNumber	string	Part # of buggy product
SerialNumber	string	Serial # of product bug was initially found in
Description	string	Description of bug
Workaround	string	Workaround, if any, for this bug

## **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

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 all Tickets for a date range

GetTicket(Ticket ID)

-returns Ticket

GetBugReport(Bug ID)

-returns Bug Report

GetAllBugReports()

-returns all Bug Reports

GetAllBugReports(Bug ID, Bug ID)

-returns all Bugs for a date range

GetAllIncidentReports()

-returns all Incident Reports

GetAllIncidentReports(Incident ID, Incident ID)

-returns all Incidents for a date range

GetAllActiveTickets()

-returns all Active Tickets

GetAllActiveTickets(Ticket ID, Ticket ID)

-returns all Active Tickets for a date range

GetAllInactiveTickets()

-returns all Inactive Tickets' Ticket IDs

GetAllInactiveTickets(Ticket ID, Ticket ID)

-returns all Inactive Tickets for a date range

GetAllBugReports()

-returns all Bug Reports

GetAllBugReports(Bug ID, Bug ID)

-returns all Bug Reports for a date range

GetAllIncidentReportsByOwner(Owner)

-returns all Incident Reports for a specific Owner

GetAllActiveTicketsByOwner(Owner)

-returns all Active Tickets for a specific Owner

GetAllInactiveTicketsByOwner(Owner)

-returns all Inactive Tickets for a specific Owner

GetAllIncidentReportsByOwner(Owner)

-returns all Incident Reports for a specific Owner

GetAllIncidentReportsByPart(PartNumber)

-returns all Incident Reports for a specific part

GetAllActiveTicketsByPart (PartNumber)

-returns all Active Tickets for a specific part

GetAllInactiveTicketsByReporter (PartNumber)

-returns all Inactive Tickets for a specific part

GetAllBugReportsByPart(PartNumber)

-returns 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 / Incident ID / Bug 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.