## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>Description</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/01/2000</td>
<td>V2000</td>
<td>Generation for beta</td>
<td>Shawn Siemers</td>
</tr>
<tr>
<td>10/02/2000</td>
<td>V2000</td>
<td>Final release</td>
<td>Shawn Siemers</td>
</tr>
<tr>
<td>01/14/2003</td>
<td>V2003</td>
<td>Final Release</td>
<td>Alex Kutsick</td>
</tr>
<tr>
<td>05/20/2004</td>
<td>2004</td>
<td>Generation for beta</td>
<td>Alex Kutsick</td>
</tr>
</tbody>
</table>
Table of Contents

   1.1 Use-Case Realization - Run Payroll 5
      1.1.1 Run Payroll (with ss interface) 5
      1.1.2 Run Payroll (with Security) 8
      1.1.3 Run Payroll (with Distribution) 10
      1.1.4 Run Payroll (with OODBMS Persistency) 13
      1.1.5 Run Payroll (with everything) 16
   1.2 Use-Case Realization - Maintain Timecard 19
      1.2.1 Maintain Timecard (with ss interface) 19
      1.2.2 Maintain Timecard (with Security) 21
      1.2.3 Maintain Timecard (with Distribution) 24
      1.2.4 Maintain Timecard (with OODBMS Persistency) 27
      1.2.5 Maintain Timecard (with everything) 30
   1.3 Use-Case Realization - Login 37
      1.3.1 Login 37
      1.3.2 Login (with Security) 39
   1.4 Object Store Support 42

2. Exercise: Use-Case Design, Part 2 49
   2.1 Packages and Their Dependencies 49
      2.1.1 Package Descriptions 49
Payroll System Use-Case Design Solution

1. Exercise: Use-Case Design, Part 1

1.1 Use-Case Realization - Run Payroll

1.1.1 Run Payroll (with ss interface)

Run Payroll - Basic Flow (with ss interface)
Run Payroll - Basic Flow (with ss interface)

1. // start( )

1.1. // run payroll( )

1.1.1. // is payday?( )

1.1.2. // get pay amount( )

1.1.3. // create with amount(float)

1.1.4. // get pay amount( )

1.1.5. // print(Paycheck, String)

1.1.6. // get bank info( )

1.1.7. deposit(Paycheck, BankInformation)

1.1.1.1. // send transaction( )

1.1.2.1. // get timecard info( )

1.1.2.2. // get PO info( )

1.1.5.1. // print( )

1.1.7.1. // send transaction( )

1.1.2.3. // calculatePay( )

1.1.2.4. // get PO info( )

1.1.5. // print( )

1.1.6. // get bank info( )

1.1.3. // create with amount(float)

1.1.4. // get pay amount( )

1.1.5. // print(Paycheck, String)

1.1.6. // get bank info( )

1.1.7. deposit(Paycheck, BankInformation)

1.1.1. // send transaction( )

1.1.2.1. // get timecard info( )

1.1.2.2. // get PO info( )

1.1.5.1. // print( )

1.1.7.1. // send transaction( )

1.1.2.3. // calculatePay( )
Run Payroll - VOPC (with ss interface)
1.1.2 **Run Payroll (with Security)**

Run Payroll - Basic Flow (with security)

This is the same collaboration diagram as Run Payroll (with ss interface). There are no additional processing steps for Security for Run Payroll, as the PayrollController is meant to be “all-knowing” and “all-seeing” and thus, has open access to all secure data for Employees.

![Collaboration Diagram](image-url)
Run Payroll - VOPC (with Security)

This is the same VOPC as Run Payroll (with ss interface). There are no additional processing steps for Security for Run Payroll, as the PayrollController is meant to be "all-knowing" and "all-seeing" and thus, has open access to all secure data for Employees.

```
Employee
  name
  employee id : int
  bank info : BankInformation
  social security number
  address
  phone number
  email
  payment method
  // is payday()? 
  // get pay amount() 
  // get payment method() 
  // get bank info() : BankInformation 
  // calculatePay() 
  // add(timecard : Timecard) 
  // get Employee ID() : int 
  // getTimecard(forPayPeriod : Date) : Timecard 
  // add(thePaycheck : Paycheck)

HourlyEmployee
  hourlyRate
  + // getHourlyRate()

SalariedEmployee
  annualSalary
  + // getAnnualSalary()

CommissionedEmployee
  commissionRate
  + // getPurchaseOrders()
  + // getCommissionRate()

Timecard
  hours worked
  pay period
  // get timecard info()
  // new()
  // save()

Paycheck
  amount
  // create with amount(forAmount : float) : Paycheck

IBankSystem
  + deposit()

IPrintService
  + print()

SystemClockInterface
  // start()

PayrollController
  generatedPaychecks
  +paycheckPrinter
  +printCheck()

PurchaseOrder
  // get PO info()
```

© Copyright IBM Corp. 2004 Page 9 of 50
1.1.3 **Run Payroll (with Distribution)**

Run Payroll - Basic Flow (with Distribution)

```
1. // start()
1.2. // run payroll()
1.2.1. // is payday()
1.2.2. // get pay amount()
1.2.2.1. // get timecard info()
1.2.2.2. // get PO info()
1.2.2.3. // calculatePay()
1.2.3. // create with amount(float)
1.2.4. // get payment method()
1.2.5. // print(Paycheck, String)
1.2.6. // get bank info()
1.2.7. deposit(Paycheck, BankInformation)
1.2.7.1. // send transaction()
1.2.7.1.1. // send transaction()
```

© Copyright IBM Corp. 2004
Run Payroll - Basic Flow (with Distribution)

1. start(
   1.1. lookup(String)
   1.2. run payroll()

2. System Clock
   1.2.1. is payday()
   1.2.2. get pay amount()
   1.2.4. get payment method()
   1.2.6. get bank info()

3. IPayrollController
   1.2.2.1. get timecard info()
   1.2.2.2. get PO info()
   1.2.2.3. calculatePay()

4. Employee
   1.2.2.4. get bank info()
   1.2.4. deposit(Paycheck, BankInformation)
   1.2.6. get bank info()

5. IBankSystem
   1.2.7. deposit(Paycheck, BankInformation)
   1.2.7.1. send transaction()

6. IPrintService
   1.2.5. print(Paycheck, String)
   1.2.5.1. print()

7. Timecard

8. PurchaseOrder

9. Paycheck

10. Bank System

11. Printer
Run Payroll - VOPC (with Distribution)

Employee
- name
- employee id : int
- bank info : BankInformation
- social security number
- address
- phone number
- email
- payment method
  - is payday?()
  - get pay amount()
  - get payment method()
  - get bank info() : BankInformation
  - calculatePay()
  - add(theTimecard : Timecard)
  - get Employee ID() : int
  - getTimecard(forPayPeriod : Date) : Timecard
  - add(thePaycheck : Paycheck)

HourlyEmployee
- hourlyRate
  - getHourlyRate()

SalariedEmployee
- annualSalary
  - getAnnualSalary()

CommissionedEmployee
- commissionRate
  - getPurchaseOrders()
  - getCommissionRate()

Paycheck
- amount
  - create with amount(forAmount : float) : Paycheck

Timecard
- hours worked pay period
  - getTimecardInfo()
  - update timecard()
  - new()
  - save()

IBankSystem
- deposit()
  - from External System Interfaces

PayrollController
- run payroll()
  - from Payroll

UnicastRemoteObject
- distributed
  - UnicastRemoteObject()
  - clone()
  - exportObject()

PayrollController is distributed

Serializable
- from java.io

Remote
- from java.rmi

UnicastRemoteObject()
- from Server

Naming.
- lookup()
  - from java.rmi

Remote
- from java.rmi

SystemClockInterface
- start()
  - from java.rmi

<<control>> PayrollController
- run payroll()
  - 0..1

<<entity>> Employee
- name
- employee id : int
- bank info : BankInformation
- social security number
- address
- phone number
- email
- payment method
  - is payday?()
  - get pay amount()
  - get payment method()
  - get bank info() : BankInformation
  - calculatePay()
  - add(theTimecard : Timecard)
  - get Employee ID() : int
  - getTimecard(forPayPeriod : Date) : Timecard
  - add(thePaycheck : Paycheck)

<<entity>> HourlyEmployee
- hourlyRate
  - getHourlyRate()

<<entity>> SalariedEmployee
- annualSalary
  - getAnnualSalary()

<<entity>> CommissionedEmployee
- commissionRate
  - getPurchaseOrders()
  - getCommissionRate()

<<entity>> PurchaseOrder
- get PO info()

<<boundary>> SystemClockInterface
- start()

<<control>> PayrollController
- run payroll()
  - 0..1

<<interface>> IPayrollController
- from Payroll

<<interface>> IBankSystem
- from External System Interfaces

<<interface>> IPayrollController
- run payroll()

<<interface>> ISystemClockInterface
- start()

<<interface>> IPrintService
- print()

<<interface>> IPayrollController
- from Payroll

<<boundary>> SystemClockInterface
- start()
1.1.4 Run Payroll (with OODBMS Persistency)

Run Payroll - Basic Flow (with OODBMS Persistency)

Perform these steps for each employee:

1. // start()
2.1. // run payroll()
2.1.2. // is payday()
2.1.3. // get pay amount()
2.1.3.1. // get timecard info()
2.1.3.2. // get PO info()
2.1.4. // create with amount (calculatePay)
2.1.5. // add(Paycheck)
2.1.6. save(Paycheck, Employee)
2.1.7. // (get payment method)
2.1.8. print(Paycheck, String)
2.1.9. // (get bank info())
2.1.10. deposit(Paycheck, bank Information)
2.1.11. // send transaction()

Example:

When the Employee is retrieved from the database, the Timecards and PurchaseOrders are retrieved as well.

1. // start()
2.1. // run payroll()
2.1.2. // is payday()
2.1.3. // get pay amount()
2.1.3.1. // get timecard info()
2.1.3.2. // get PO info()
2.1.4. // create with amount (calculatePay)
2.1.5. // add(Paycheck)
2.1.6. save(Paycheck, Employee)
2.1.7. // (get payment method)
2.1.8. print(Paycheck, String)
2.1.9. // (get bank info())
2.1.10. deposit(Paycheck, bank Information)
2.1.11. // send transaction()
Run Payroll - Basic Flow (with OODBMS Persistency)

1. // start(

1.1. // run payroll(

1.1.1. getEmployee(string)

1.1.2. // is payday?( )

1.1.3. // get pay amount( )

1.1.4. // create with amount (calculatedPay)

1.1.10. deposit(Paycheck, BankInformation)

1.1.10.1. // send transaction( )

1.1.1.6. save(Paycheck, Employee)

1.1.8. print(Paycheck, String)

1.1.8.1. // print( )

1.1.3. // get timecard info( )

1.1.3.1. // get timecard info( )

1.1.3.2. // get PO info( )

1.1.3.3. // calculatePay( )

1.1.5. // add(Paycheck)

1.1.7. // get payment method( )

1.1.1.9. // get bank info( )

1.1.9. // get bank info( )

1.1.1.7. // get payment method( )

1.1.1.8. print(Paycheck, String)

1.1.1.8.1. // print( )

1.1.1.8.2. // print( )

1.1.1.1. getEmployee(string)

1.1.1.6. save(Paycheck, Employee)
Run Payroll - VOPC (with OODBMS Persistency)
1.1.5 **Run Payroll (with everything)**

Run Payroll - Basic Flow (with everything)

```plaintext
1.1. lookup(String)
1.2. // run payroll()

Perform these steps for each employee:

1.2.1. getEmployee(String)
1.2.2. // is payday()
1.2.3. // get pay amount()
1.2.3.1. // get timecard info()
1.2.3.2. // get PO info()
1.2.3.3. calculatePay()
1.2.4. // create with amount (calculatedPay)
1.2.6. add(Paycheck, Employee)
1.2.7. // get payment method()
1.2.8. print(Paycheck, String)
1.2.9. // get bank info()
1.2.10. deposit(Paycheck, BankInformation)

Sequence Diagram:
OODBMS Support
PayrollDBManager - Get Employee

Sequence Diagram:
OODBMS Support
PayrollDBManager - Get Employee
```

All calls are forwarded to the remote object.

System Clock

When the Employee is retrieved from the database, the Timecards and PurchaseOrders are retrieved as well.

© Copyright IBM Corp. 2004
Run Payroll - Basic Flow (with everything)

1. // start()
   1.1. lookUp(string)
   1.2. // run payroll()
      1.2.1. getEmployee(string)
      1.2.2. // is payday()
      1.2.3. // get pay amount()
      1.2.4. // create with amount (calculatedPay)
      1.2.5. // addPaycheck
      1.2.6. // savePaycheck
      1.2.7. // get payment method()
      1.2.8. // print(Paycheck, String)
      1.2.9. // get bank info()
      1.2.10. // send transaction()
      1.2.10.1. // send transaction()

   1.2.8.1. // print()
   1.2.8.2. // print()
   1.2.8.3. // print()

   1.2. // run payroll()
      1.2.1. getEmployee(string)
      1.2.2. // is payday()
      1.2.3. // calculatePay()
      1.2.4. // create with amount (calculatedPay)
      1.2.5. // addPaycheck
      1.2.6. // savePaycheck
      1.2.7. // get payment method()
      1.2.8. // print(Paycheck, String)
      1.2.9. // get bank info()
      1.2.10. deposit(Paycheck, BankInformation)

   1.2. // run payroll()
      1.2.1. getEmployee(string)
      1.2.2. // is payday()
      1.2.3. // calculatePay()
      1.2.4. // create with amount (calculatedPay)
      1.2.5. // addPaycheck
      1.2.6. // savePaycheck
      1.2.7. // get payment method()
      1.2.8. // print(Paycheck, String)
      1.2.9. // get bank info()
      1.2.10. deposit(Paycheck, BankInformation)

   System Clock
Run Payroll - VOPC (with everything)
1.2 Use-Case Realization - Maintain Timecard

1.2.1 Maintain Timecard (with ss interface)

Maintain Timecard - Basic Flow (with ss interface)
Maintain Timecard - Basic Flow (with ss interface)

1. // maintain timecard() 
   1.1. // get current timecard() 
   1.1.1. // getTimecard(Date) 
   1.3. // get charge codes() 
      1.3.1. getChargeNumbers(String) 
      1.3.1.1. get charge numbers() 
   2. // enter hours for charge numbers() 
   2.1. // update timecard() 
      2.1.1. // update timecard() 
   3. // save timecard() 
      3.1. // save() 
      3.1.1. // save( )

Maintain Timecard - VOPC (with ss interface)

<<boundary>>
TimecardForm

<<control>>
TimecardController

<<entity>>
Timecard

<<entity>>
Employee

© Copyright IBM Corp. 2004
1.2.2 **Maintain Timecard (with Security)**

Maintain Timecard - Basic Flow (New Timecard with Security)

1. `Maintain Timecard()`
   
   **The secure user session for the employee established at Login must be propagated to all forms and:**

   1.1. `open(ISecureUser)`
   
   A current timecard does not exist

   1.1.1. `create()`
   
   A new Timecard is created and added to the Employee

   1.1.2. `get current timecard()`
   
   The employee is given "open" permissions for his/her timecard

   1.1.3. `get current timecard()`
   
   1.1.4. `display timecard()`
   
   1.1.5. `get charge codes()`

   1.1.5.1. `getChargeNumbers(String)`

   1.1.5.1.1. `get charge numbers()`

   1.1.6. `display charge codes()`

   10. `enter hours for charge numbers()`

   10.1. `update timecard()`

   10.1.1. `update timecard()`

   No need to check Timecard access permissions, the TimecardController already knows it is the Employee's Timecard.

   11. `save timecard()`

   11.1. `save()`

   © Copyright IBM Corp. 2004
Maintain Timecard - Basic Flow (New Timecard with Security)

1.1.4. // display timecard()  
1.1.6. // display charge codes()  

1.1.3. // get current timecard()   
1.1.5. // get charge codes()   
10.1. // update timecard()   
11.1. // save timecard()  

10. // enter hours for charge numbers()   
11. // save timecard()  

1.1. // maintain timecard()  
1.1.1. // create()   
1.1.2. // setSession(ISecureUser)   
1.1.5.1. // get charge numbers()  
1.1.5.1.1. // get charge numbers()  

2. // getTimecard(Date)  
4. // add(Timecard)  
10.1.1. // update timecard()  
11.1.1. // save()  

3. // new()  
10.1. // update timecard()  
11.1. // save()  

9. setAccess(Timecard, TimecardAccess)  
5. new()   
6. makeReadable()   
7. makeWriteable()   
8. makeDeleteable()  

1.1.5.1. getChargeNumbers(String)  

© Copyright IBM Corp. 2004
Maintain Timecard - VOPC (with Security)

- MAINEMPLOYEEFORM
  - // maintain timecard()

- TIMECARDFORM
  - // display timecard()
  - + // open()
  - + // enter hours for charge numbers()
  - + // display charge codes()
  - + // save timecard()

- TIMECARDCONTROLLER
  - + // get current timecard()
  - + // get charge codes()
  - + // update timecard()
  - + // setSession()
  - + // create()
  - + // save timecard()

- ISECUREUSER
  - + setAccess()
  - + getAccess()
  - + getUserId()
  - + new()

- ISECUREDATA
  - + setAccess()
  - + getAccess()
  - + getUserID()
  - + new()

- IPROJECTMANAGEMENTDATABASE
  - + getChargeNumbers()
1.2.3 **Maintain Timecard (with Distribution)**

**Maintain Timecard - Basic Flow (with distribution)***

```plaintext
1. // open()
   1.1. lookup(String)
   1.2. // get current timecard()
       1.2.1. // getTimecard(Date)
   1.3. // display timecard()
   1.4. // get charge codes()
       1.4.1. getChargeNumbers(String)
   1.5. // display charge codes()

2. // enter hours for charge numbers()
   2.1. // update timecard()
       2.1.1. // update timecard()

3. // save timecard()
   3.1. // save timecard()
       3.1.1. // save()
```

Lookup remote object by specifying it's URL. This returns a reference to the remote object interface. All calls are forwarded to the remote object.

Get the Timecard for the current pay period.
Maintain Timecard - Basic Flow (with distribution)

1.3. // display timecard( )
1.5. // display charge codes( )
Maintain Timecard - VOPC (with Distribution)

**Distributed class client**

**<<boundary>>**

**TimecardForm**
(from Employee Activities)

- // display timecard()
+ // open()
+ // enter hours for charge numbers()
+ // display charge codes()
+ // save timecard()

**Naming.**
(from java.rmi)

+ lookup()

**TimecardController**

is distributed

**<<interface>>**

**ITimecardController**
(from Employee Activities)

+ // get current timecard()
+ // get charge codes()
+ // update timecard()
+ // setSession(forUser : ISecureUser)
+ // create() : ITimecardController
+ // save timecard()

**Employee**
(from Payroll Artifacts)

- name
- employee id : int
+ // add()

**<<entity>>**

**Timecard**
(from Payroll Artifacts)

- hours worked
- pay period
+ // get timecard info()
+ // update timecard()
+ // new()
+ // save()

**<<control>>**

**TimecardController**
(from Employee Activities)

+ // get current timecard()
+ // get charge codes()
+ // update timecard()
+ // create() : TimecardController
+ // save timecard()

**<<interface>>**

**IProjectManagementDatabase**
(from External System Interfaces)

+ getChargeNumbers()

**<<interface>>**

**UnicastRemoteObject**
(from Server)

# UnicastRemoteObject()
+ clone()
+ exportObject()

**<<Interface>>**

**ITimecardController**
(from Employee Activities)

+ // get current timecard()
+ // get charge codes()
+ // update timecard()
+ // setSession(forUser : ISecureUser)
+ // create() : ITimecardController
+ // save timecard()

**Timecard**
(from Payroll Artifacts)

- hours worked
- pay period
+ // get timecard info()
+ // update timecard()
+ // new()
+ // save()

**Serializable**
(from java.io)

Remote
(from java.rmi)

Serialization.
(+ lookup())

**Timecard must be passed between distributed objects**
1.2.4 Maintain Timecard (with OODBMS Persistence)

Maintain Timecard - Basic Flow (with OODBMS Persistency)

1. // open()
   1.1. // create()
   1.2. // get current timecard()
      1.2.1. getTimecard(Employee, Date)
   1.3. // display timecard()
   1.4. // get charge codes()
      1.4.1. getChargeNumbers(String)
   1.5. // display charge codes()
2. // enter hours for charge numbers()
   2.1. // update timecard()
      2.1.1. // update timecard()
3. // save timecard()
   3.1. // save()
      save(Timecard, Employee)
Maintain Timecard - Basic Flow (with OODBMS Persistency)

1.1. // create( )
1.2. // get current timecard( )
1.4. // get charge codes( )
1.5. // display charge codes( )
1.6. // display timecard( )

2. // enter hours for charge numbers( )
3. // save timecard( )

1.1.1. // create( )
1.2.1. // update timecard( )
3.1. // save timecard( )

1.4.1. // get charge numbers( )
1.4.1.1. // get charge numbers( )
2.1.1. // update timecard( )
3.1.1. // save( )

© Copyright IBM Corp. 2004
Maintain Timecard - VOPC (with OODBMS Persistency)

**PayrollDBManager**
(from ObjectStore Support)

- **+ save()**
- **+ getTimecard()**

**Timecard**
(from Payroll Artifacts)

- **- hours worked**
- **- pay period**
- **+ // get timecard info()**
- **+ // update timecard()**
- **+ // new()**
- **+ // save()**

**IProjectManagementDatabase**
(from External System Interfaces)

- **+ getChargeNumbers()**

**TimecardController**
(from Employee Activities)

- **+ // get current timecard()**
- **+ // get charge codes()**
- **+ // update timecard()**
- **+ // create()**
- **+ // save timecard()**

**TimecardForm**
(from Employee Activities)

- **- // display timecard()**
- **+ // open()**
- **+ // enter hours for charge numbers()**
- **+ // display charge codes()**
- **+ // save timecard()**
1.2.5 Maintain Timecard (with everything)

Maintain Timecard - Basic Flow (with everything)

1. // open(ISecureUser)

1.1. lookup(String)

1.2. // current()

1.3. // setSession(ISecureUser)

1.4. // get current timecard()

1.5. // display timecard()

1.6. // get charge codes()

1.7. // display charge codes()

2.1. // update timecard()

2.1.1. // updated timecard()

3.1. // save timecard()

3.1.1. save(Timecard, Employee)

The secure user session for the employee established at Login must be propagated to all forms and controllers, so access can be checked for secure data.

A new Timecard is created and added to the Employee.

The employee is given "open" permissions for his/her timecard.

No need to check Timecard access permissions, the TimecardController already knows that it is the Employee's Timecard.

© Copyright IBM Corp. 2004
Maintain Timecard - Basic Flow (with everything), Part 1

- Employee
  - TimecardForm
  - Naming
  - TimecardController
  - PayrollDBManager
  - Timecard
  - Employee
  - TimecardAccess
  - EmployeeSession

The secure user session for the employee established at Login must be propagated to all forms and controllers, so access can be checked for secure data.

1. // open(ISecureUser)
1.5. // display timecard()
1.1. lookup(String)
1.2. // create()
1.3. // setSession(ISecureUser)
1.4. // get current timecard()

All calls are forwarded to the remote object.

1.4.1. getTimecard(Employee, Date)
1.4.2. // close()
1.4.3. // add(Timecard)
1.4.4. new()
1.4.5. makeReadable()
1.4.6. makeWriteable()
1.4.7. makeDeleteable()
1.4.8. setAccess(Timecard, TimecardAccess)

A new Timecard is created and added to the Employee.

The employee is given "open" permissions for his/her timecard.

1.5. // display timecard()
Maintain Timecard - Basic Flow (with everything), Part 2

Continued from Maintain Timecard - Basic Flow (with everything), Part 1

1. // get charge codes( )
2. // display charge codes( )
3. // enter hours for charge numbers( )
4. // save timecard( )

3.1. // update timecard( )
4.1. // save timecard( )

// save Timecard in the Payroll Database

No need to check Timecard access permissions, the TimecardController already knows that it is the Employee's Timecard.

© Copyright IBM Corp. 2004
Maintain Timecard - Basic Flow (with everything), Part 1

1. // open(ISecureUser)
   1.1. lookup(String)
   1.2. // create()
   1.3. // setSession(ISecureUser)
   1.4. // get current timecard()

Employee

TimecardForm

Naming

EmployeeSession : ISecureUser

ITimecardController

PayrollDBManager

Employee

Timecard

Employee

TimecardAccess : SecurityAccess

1.5. // display timecard()

1.1.4. setAccess(Timecard, TimecardAccess)

1.4.1. getTimecard(Employee, Date)

1.4.2. // new()

1.4.3. // add(Timecard)

1.4.4. new()

1.4.5. makeReadable()

1.4.6. makeWriteable()

1.4.7. makeDeleteable()
Maintain Timecard - Basic Flow (with everything), Part 2

1. // get charge codes()
   1.1.1. // get charge numbers()  
2. // display charge codes()
3. // enter hours for charge numbers()  
4. // save timecard()  
3.1. // update timecard()  
4.1. // save timecard()  
3.1.1. // update timecard()  
4.1.1. save(Timecard, Employee)
Maintain Timecard - Basic Flow (with everything)

1.5. // display timecard()
1.7. // display charge codes()
Maintain Timecard - VOPC (with everything)
1.3  Use-Case Realization - Login

1.3.1  Login

Login - Basic Flow

: Any User  : LoginForm

1. // enter username and password( )

  1.1. // validate username and password( )

: LoginForm

1. // enter username and password( )

: Any User
Login - VOPC

<table>
<thead>
<tr>
<th>LoginForm</th>
</tr>
</thead>
<tbody>
<tr>
<td>(from GUI Framework)</td>
</tr>
<tr>
<td>+ open()</td>
</tr>
<tr>
<td>+ enterUserName()</td>
</tr>
<tr>
<td>+ validateUserIDPassword()</td>
</tr>
<tr>
<td>+ enterPassword()</td>
</tr>
<tr>
<td>+ logInUser()</td>
</tr>
<tr>
<td>+ setupSecurityContext()</td>
</tr>
<tr>
<td>+ getUserContext()</td>
</tr>
</tbody>
</table>
1.3.2 Login (with Security)

Login - Basic Flow (Employee Login with Security)

The MainEmployeeForm retains the Employee's session for later processing by the user.
Login - Basic Flow (Employee Login with Security)

1. start()
   2. enterUserName()
   3. enterPassword()
   4. loginUser()

: Employee

: MainEmployeeForm
   1.1. open()
   1.2. getUserContext()
   1.3. close()

: LoginForm
   4.1. validateUserIDPassword()
   4.2. setupSecurityContext()

: Login

: Employee Session:
   ISecureUser

4.1. validateUserIDPassword()
Login - VOPC (with Security)
1.4 ObjectStore Support

The following diagrams demonstrate the design of the PayrollDBManager class operations. These are included to supplement the use-case realization diagrams provided above. For the use-case realization diagrams that involve OODBMS persistency, there are references to the diagrams in this section.

**PayrollDBManager - Save Timecard**

1. save(Timecard, Employee)
   
   a read-only transaction to ensure that the Employee isn't changed while we're reading it
   
   1.1. begin()

   1.2. // get Employee ID()

   1.3. get (EmployeeID)

   Returns an Employee

   [ Employee does not exist]

   1.4. put(EmployeeID, Employee)

   Save the Employee (and all associated data, including Timecards) in the database

   [ Employee does exist]

   1.5. // add(Timecard)

   If the Employee already has a Timecard for that pay period, the existing Timecard is updated with the given Timecard.

   A separate put() to the Map is not necessary as the get() operation returns a reference to the Employee and any changes to that Employee (like adding a Timecard), if made in the context of a transaction, are automatically committed to the database when the transaction is committed.
PayrollDBManager - Get Timecard

1. getTimecard(Employee, Date)
   1.1. begin()
   1.2. // get Employee ID()
   1.3. get(EmployeeID)
   1.4. // getTimecard(Date)

2. commit(RETAIN_HOLLOW)

Specify the RETAIN_HOLLOW option on the commit(), so that the reference to the retrieved Timecard can be used outside of the transaction.
PayrollDBManager - Get Employee

Specify the RETAIN_HOLLOW option on the commit(), so that the reference to the retrieved Student can be used outside of the transaction.
Initialization must occur before any persistent class can be accessed.

Once the session has been created and joined, the PayrollDBManager must open and create the new database.

To create the database, the PayrollDBManager creates a new transaction and creates the "root" of the database with the "createRoot()" operation. In our example, the root will be the EmployeeMap data structure. It will contain instances of the Employee class and all "reachable" classes (including Timecards and Purchase Orders). Remember, the root is the entry point into the Payroll Database. It is a "special" data structure. Any changes to this data structure that occur within the context of a transaction will be applied to the associated Payroll ObjectStore Database.

Once the root has been created, the transaction is committed
PayrollDBManager - Shutdown

To shutdown the database, the PayrollDBManager must close the database and terminate the session.

PayrollDBManager - Save Paycheck

A read-only transaction to ensure that the Employee isn't changed while we're reading it

Save the Employee (and all associated data, including Paychecks) in the database

A separate put() to the Map is not necessary as the get() operation returns a reference to the Employee and any changes to that Employee (like adding a Paycheck), if made in the context of a transaction, are automatically committed to the database when the transaction is committed.
Main

Operations to access Timecard and Purchase Order were added for easier design incorporation. The PayrollDBManager manages the details of loading the appropriate Employee (the selected root class) before accessing Timecard and Purchase Order.

### PayrollDBManager

- initialize()
- shutdown()
- newTimecard(forEmployee : Employee) : Timecard
- deleteTimecard(forEmployee : Employee)
- updateTimecard(forEmployee : Employee)
- getTimecard(forEmployee : Employee)
- newEmployee() : Employee
- deleteEmployee()
- updateEmployee()
- saveEmployee()
- getEmployee(withID : string) : Employee
- newPO(forEmployee : CommissionedEmployee) : PurchaseOrder
- deletePO(thePO : PurchaseOrder, forEmployee : CommissionedEmployee)
- updatePO(thePO : PurchaseOrder, forEmployee : CommissionedEmployee)
- getPO(thePO : PurchaseOrder, forEmployee : CommissionedEmployee)
- getEmployee(withID : string) : Employee

PayrollDBManager: For the Payroll System, there is one ObjectStore database, the Payroll Database, that contains employee, timecard, and purchase order information for the company. There is one PayrollDBManager (i.e., this class is a singleton).

This class is responsible for providing access to the persistent objects in the Payroll Database. It provides a single entry point into the Payroll Database. It contains operations to access entities in the database.

The PayrollDBManager class contains most of the database-specific code, such as starting and ending transactions. There are no PayrollDBManager objects stored in the database, which means that the PayrollDBManager class is not required to be persistence-capable.

The PayrollDBManager class has a static members that keep track of the database that is open. It also has a number of static methods, each of which executes a transaction in the ObjectStore database.

**Session**: The class that represents a database session. A session must be created in order to access the database and any persistent data.

A session is the context in which PSE/PSE Pro databases are created or opened, and transactions can be executed. Only one transaction at a time can exist in a session.

**Map**: A persistent map container classes that stores key/value pairs.

**Database**: The Database class represents an ObjectStore database.

Before you begin creating persistent objects, you must create a database to hold the objects. In subsequent processes, you open the database to allow the process to read or modify the objects. To create a database, you call the static create() method on the Database class and specify the database name and an access mode.

**Transaction**: An ObjectStore transaction. Manages a logical unit of work. All persistent objects must be accessed within a transaction.

**ObjectStore**: Defines system-level operations that are not specific to any database.

**HourlyEmployee**: An employee that is paid by the hour.
SalariedEmployee: An employee that receives a salary.

CommissionedEmployee: An employee that receives a commission.

PurchaseOrder: A record of a sale made by an employee.

Timecard: The timecard contains information regarding the hours worked by an employee for a given time period.

Employee: A person that works for the company.

Paycheck: A record of how much an employee was paid for a given pay period.
2. Exercise: Use-Case Design, Part 2

2.1 Packages and Their Dependencies

Package Dependencies Diagram

2.1.1 Package Descriptions

Administration: Contains the design elements that support the Payroll Administrator's applications.
BankSystem Subsystem: Encapsulates the details involved in communicating with external bank systems.

Base Reuse: Basic reusable design elements.

com.odi: The com.odi package contains the design elements that support the OODBMS persistency mechanism. The name of the package in the model reflects the naming convention for 3rd party Java software. The convention is to use the reverse of the domain name, so if Rational had a Java package called "util" they’d call it com.rational.util. This com.odi has nothing to do with Microsoft COM/DCOM, they are totally separate. There is nothing COM/DCOM related when using CORBA, RMI, or ObjectStore.

Employee Activities: Contains the design elements that support the Employee's applications.

External System Interfaces: Contains the interfaces that support access to external systems. This is so that the external system interface classes can be version controlled independently from the subsystems that realize them.

GUI Framework: This package comprises a whole framework for user interface management.

It has a ViewHandler that manages the opening and closing of windows, plus window-to-window communication so that windows do not need to depend directly upon each other.

This framework is security-aware, it has a login window that will create a server-resident user context object. The ViewHandler class manages a handle to the user context object.

The ViewHandler also starts up the controller classes for each use case manager.

java.awt: The java.awt package contains the basic GUI design elements for Java.

java.io:

java.lang: The package contains some basic Java design elements.

java.rmi: The java.rmi package contains the classes that implement the RMI distribution mechanism. This package is commercially available with most standard JAVA IDEs.

java.sql: The package that contains the design elements that support RDBMS persistency.

ObjectStore Support: Contains the business-specific design elements that support the OODBMS persistency mechanism. This includes the DBManager. The DBManager class must contain operations for every OODBMS persistent class.

Payroll: Contains the design elements that support the execution of the payroll processing.

Payroll Artifacts: Contains the core payroll abstractions.

PrintService Subsystem: Provides utilities to produce hard-copy.

ProjectManagementDatabase Subsystem: Encapsulates the interface to the legacy database containing information regarding projects and charge numbers.

Secure Interfaces: Contains the interfaces that provide clients access to security services.

Security: Contains design elements that implement the security mechanism.

Security Manager Subsystem: Provides the implementation for the core security services.

Server: